Hindawi
Security and Communication Networks
Volume 2019, Article ID 1798391, 15 pages
https://doi.org/10.1155/2019/1798391

Research Article

Dynamic Resource Provisioning on Fog Landscapes

Hoang-Nam Pham-Nguyen [1,2] **and Quang Tran-Minh** [1]

[1] *Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, VNU-HCM, Vietnam*
[2] *Faculty of Information Technology, Industrial University of Ho Chi Minh City, Vietnam*

Correspondence should be addressed to Hoang-Nam Pham-Nguyen; nampham.sdh2018@hcmut.edu.vn

Received 30 November 2018; Revised 8 March 2019; Accepted 15 April 2019; Published 2 May 2019

Academic Editor: Petros Nicopolitidis

A huge amount of smart devices which have capacity of computing, storage, and communication to each other brings forth fog computing paradigm. Fog computing is a model in which the system tries to push data processing from cloud servers to "near" IoT devices in order to reduce latency time. The execution orderings and the deployed places of services make significant effect on the overall response time of an application. Beside new research directions in fog computing, e.g., fog-cloud collaboration, service scalability, fog scalability, mobile fog computing, fog federation, trade-off between energy consumption and communication efficiency, duration of storing data locally, storage security and communication security, and semantic-aware fog computing, the service deployment problem is one of the attractive research fields of fog computing. The service deployment is a multiobjective optimization problem; there are so many proposed solutions for various targets, such as response time, communication cost, and energy consumption. In this paper, we focus on the optimization problem which minimizes the overall response time of an application with awareness of network usage and server usage. Then, we have conducted experiments on two service deployment strategies, called cloudy and foggy strategies. We analyze numerically the overall response time, network usage, and server usage of those two strategies in order to prove the effectiveness of our proposed foggy service deployment strategy.

1. Introduction

Cloud computing is a suitable computing model for all users to deploy their applications into their customers because there is no need for investing much on physical equipment and there is no limitation on customers' geographic locations. End users are able to rent cloud services with various configurations (e.g., storage, computing, and cost); hence, there is no limitation on applications' processing because service providers will expand logical or physical servers according to their customers' requirements. Therefore, communication cost of applications from end-user devices to servers plays an important role in the response time of an application. Nowadays, the amazingly fast development of devices transforms from "thin sensors" to "smart devices". "Thin devices" are sensors which only detect or measure the changes of conditional environment. "Smart devices" are devices which have additional features, such as computational, storage, and networking resources. When devices have effective communication methods to each other, they are able to join owned Edge computing networks in order to share computing, storage ability. Edge computing networks integrate with cloud computing into fog computing.

Fog computing is in the beginning phase of its life time. Firstly, F. Bonomi et al. give a first definition on fog computing with three characteristics (e.g., highly virtualized, resided between IoT devices and cloud, and not exclusively located at the edge) [1]. Then, L. M. Vaquero et al. also state another definition on fog computing as networks consisted of heterogeneous, ubiquitous, and decentralized devices connection, without third party invention, and running in a sand-boxed environment [2]. When National Institute of Standards and Technology (NIST) releases an official definition of the fog computing in [3] and fog computing conceptual model in [4], there are many surveys [5, 6] on fog computing and similar technologies (e.g., mobile cloud computing, Edge computing, and mobile edge computing) in order to find out fog computing characteristics and problems deeply. Many works have been done to quantify the layer-based concept of fog architecture, as three layers [7, 8],

four layers [9], or five layers [10] of the fog architecture. Because fog computing is formed from the combination of cloud computing and Edge computing, fog computing makes use of advantages of two paradigms in order to overcome drawbacks of those. Fog computing utilizes its owned Edge computing networks in order to push data processing tasks totally or partially from cloudy-level servers to smart devices in the middle layer, i.e., fog devices. If an application requires computing ability more than an Edge computing network's ability, this Edge computing network will push data processing tasks totally or partially to other fit Edge computing networks or to cloudy-level servers in order to not exceed application's deadline. Therefore, fog computing is appropriate for applications which are not only processed on the middle computing networks to meet low latency requirement but also processed on cloudy-level for complex, difficult requirements. Therefore, others try to prove that fog computing and its characteristics are effective on many specified applications, e.g., improving healthcare systems and their performance [11–13], vehicular networks and road safety [14, 15], provisioning 5G mobile networks [16, 17], surveillance video stream processing [18], and saving energy in cloud computing [19]. Finally, others focus on specified problems in fog computing environment, e.g., survey on security issues [20, 21], building security schema [22, 23], and building simulation framework named iFogSim [24] and MyiFogSim [25].

An application's service execution plans have a significant effect on overall processing of this application. There are several optimization solutions for service deployment problem, each of them focuses on different expected target variables, e.g., time (computation time, communication time, deadline), data (data size, data flow), cost (networking cost, deployment cost, execution cost), and energy consumption.

In this paper, we focus on optimization solutions for minimizing overall response time of an application. Our solution is based on checking status of services and fog devices on real-time for finding best deployments. Our work considers the application which is regulated services' execution priority. The main contribution of our work is that we combine three components (response time, network congestion, and server usage) into one objective function. The best service deployment, we proposed, is an execution plan which has trade-off among minimizing application's response time, minimizing network congestion, and minimizing cloud layer's server usage. Then, we do six experiments on characteristics: application type and the number of services for two types of service deployment policy, called cloudy and foggy strategies. Basing on two metrics, overall response time and network congestion, we do compare cloudy to foggy strategies and give analytic on the results for proofing our effective method.

The remaining of this paper is organized as follows: after related work in Section 2 and preliminaries in Section 3, we formulate and describe by algorithms our proposed optimization strategy on resource provisioning on fog landscapes in Section 4. Finally, we explain and analyze our experimental results in Section 5 and present conclusions and future research in Section 6.

2. Related Work

Service Deployment Optimization is a classical problem in computer science. This is a multiobjective optimization without perfect solutions. There are several optimization solutions for service deployment problem, each of them focusing on different expected target variables, e.g., time (computation time, communication time, deadline), data (data size, data flow), cost (networking cost, deployment cost, execution cost), and energy consumption.

O. Skarlat et al. in [26] try to find solution for maximizing the number of services deployed on fog devices. Their research focuses on Quality of Service (application response times). Therefore, the main idea of their solution is maximizing the utilization of existing resources on the fog landscape. Each service will be executed on a device (current fog colony, or neighbour fog colony, or cloud server) if capacities of this device are greater or equal than this service's demands and estimated execution time is less than the application's deadline. Then, authors improve their work by applying heuristics to solve their fog Service Placement Problem in [27]. First, the system finds a device which fits for a service's execution. Then, they swap services placed on near fog devices in order to reduce their transfer time. Authors assume that each service of an application can be executed independently. It is NOT realistic, because the output of a service sometimes is used as the input data of other services. Therefore, the priority of service executing should be considered conceptually.

N. Mostafa et al. in [28] utilize Fog-to-Fog communication and historical executions in order to minimize delay in IoT environment. Authors propose a new algorithm called Fog Resource Selection which is introduced as a run-time prediction for resource selection. Then, they implement this proposed algorithm as the Fog Resource Selection (FResS) module in the fog layer to manage requests from IoT devices towards fog entities. Every request from IoT devices will be sent to FResS module. FResS algorithm is executed in FResS module through two phases. Firstly, FResS module makes prediction on execution cost, required resources based on historical executions. Then, FResS module queries a list of current available resources in order to make prediction on finding the best resource(s) for execution. To reduce prediction cost on FResS module, results of every execution are stored as an execution history log file and are subsequently utilized for mining, discovering knowledge, and updating stored knowledge. They use Artificial Neural Networks (ANNs) as a learning-based method for their system, because ANNs can work with such very large number of resources on fog/cloud computing. The more the number of historical executions, the more the accuracy of predictions. The prediction results are stored in a separate database called the Prediction Model Databases. When a new task appears, the prediction model database is searched locally for sending the result which is an already deployed similar task. After a task is completed, the parameters are stored in the prediction model database for future executions. If not, Fog-to-Fog communication will take place to execute the task. The more the number of historical executions, the faster the predictions.

L. Liu et al. in [29] focus on computation offloading problem in mobile cloud computing, an instance of fog computing paradigm. Mobile Cloud Computing can provide computing resources at the Edge computing networks which is made from mobile devices. Sharing computing and storage capacity in mobile cloud computing brings executing location more closer to the mobile devices. They find that the mobile devices can offload its data or computational expensive tasks to the fog node within its proximity, instead of distant cloud. The most important characteristic of mobile devices is limited energy. However, it sometimes incurs additional delay and generates related cost for enjoying the cloud service. For example, minimizing the delay time by executing close to source mobile devices can be applied by running the services at the local Edge computing networks. On the other side, running too many services locally may take large amount of energy of the mobile devices. The payment cost of users will go down if their application utilizes the resources of Edge computing networks. Therefore, authors address service provisioning as energy consumption, Delay and Payment (E&D&P) optimization problem which is the trade-off among energy consumption, delay performance, and payment cost when deploying services. They propose Interior Point Method (IPM) for transforming multiobjective optimization to single objective optimization by finding the optimal offloading probability and transmit power for each mobile device.

J. Zhang et al. [30] consider fog devices as smart mobile devices (SMDs) which have limited computation resources (e.g., central process unit (CPU) frequency and memory) and battery lifetime. SMDs connect to the cloud sever which has higher computation capacity and storage in order to have capacity on executing computing-intensive applications which require higher computing capacity and more energy than traditional applications on SMDs. It comes into being mobile cloud computing (MCC). As the enormous popularity with the emerging of mobile technologies like Internet of Things (IoTs) and wearable devices and new architectures and key technologies for 5G networks, mobile edge computing (MEC) is presented as a platform for relocating the cloud computation resource close to SMDs because the cloud severs are spatially far from SMDs, which causes high transmission latency and obstructs the latency-sensitive applications. Therefore, computing on SMDs causes high energy cost with the limited battery lifetime. Therefore, authors give definition on the weighting factors of energy consumption and latency in order to consider trade-off of them in making efficient offloading decisions in MEC Networks. Authors focus on finding the solution which optimizes local computing frequency scheduling, channel allocation, power allocation, and computation offloading.

Y. Xiao et al. in [31] consider fog computing as a wireless network-supported system. Fog nodes can offload part or all the workload from the cloud data centers in order to improve the quality-of-experience (QoE) of users. Therefore, authors focus on the workload offloading problem for fog computing networks. QoE is introduced recently in [32] as one of the main guiding paradigms for service quality of the cloud computing networks. In fog computing, authors consider and investigate the relationship between two performance metrics: the users' QoE and the fog nodes' power efficiency. Authors propose a novel cooperation strategy referred to as offload forwarding, which describes how each fog node can forward part or all of its unprocessed workload to its neighbouring fog nodes instead of always sending the request to and receiving the returned result from cloud data centers. They use a given power efficiency constraint in order to help fog nodes deciding the optimal partitions of workload to be forwarded to other fog nodes. When investigating the trade-off between the users' QoE and the fog nodes' power efficiency, authors propose a distributed alternating direction method of multipliers (ADMM) to approach the global optimal workload allocation.

In our work, we try to make an execution plan which is aware of three issues (application's response time, network congestion, and server usage). We minimize the application's response time because the minimum application's response time indicates the deploying services on devices are a good strategy with low latency. The communication time which is used for sending and receiving date among devices, plays an important role in total response time of an application. Other proposed methods try to deploying service near devices which send the data for minimizing the overall response time of an application. Therefore, the state of devices near deployment process is unavailable. The second considered issue is minimizing the amount of data which is sent and received on the network environment. The final considered issue is server usage. The service usage is computed based on the amount of cloud layer's server usage for execution application. It means the cost which a user has to charge when using utilities of an application. Therefore, our proposed method is minimizing the server usage parameter in order to reduce overall cost of an application. So, we combine three components (response time, network congestion, and server usage) into one objective function. The best service deployment, we proposed, is an execution plan which has trade-off among minimizing application's response time, minimizing network congestion, and minimizing cloud layer's server usage.

3. Preliminaries

3.1. Service and Application. In this work, we define a service as a computing function of an application. Each service has three basic components, *input*, *processing*, and *output*. In Figure 1, there are many complex services which are made from ordered list of services. We assume that a service is an undividable processing unit. It means that a service has to be deployed only on a single device. In practice, we model a service s as a 4-tuple, ($Instructor(s)$, $Dema(RAM,s)$, $Dema(Storage,s)$, and $Dema(Type,s)$), described as follows:

(i) $IInstructor(s)$ is content of service s, a set of instructions which have to be executed on a single device, measured on mega-instruction (MI);

(ii) $Dema(RAM,s)$ is the demand on RAM (memory) of service s, measured on megabyte (MB);

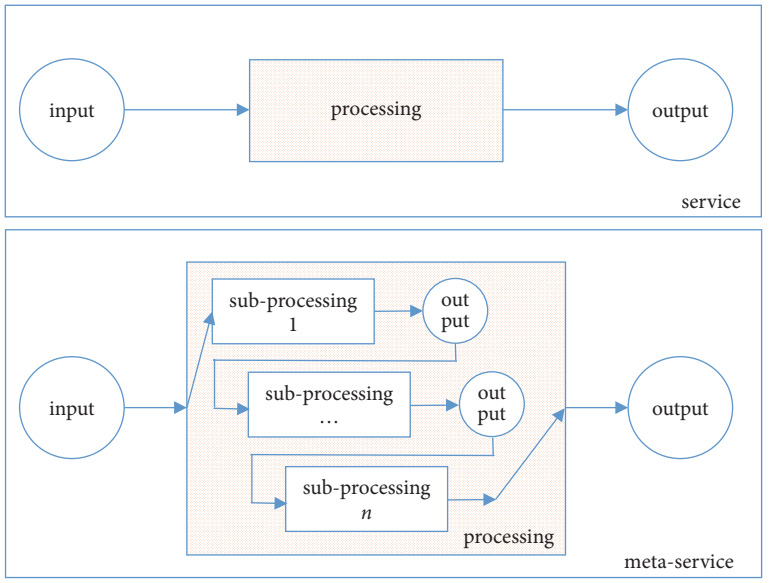

Figure 1: Structure of a service.

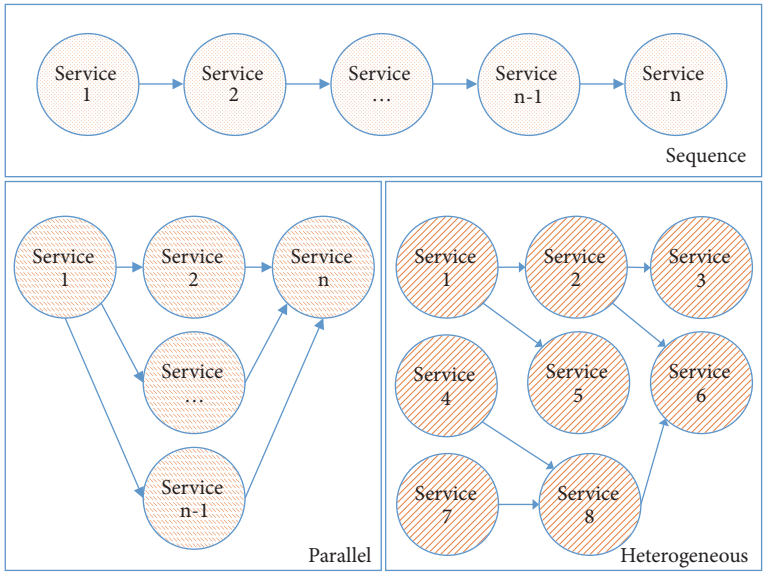

Figure 2: Types of a service string.

(iii) *Dema*(*Storage*,*s*) is the demand on *Storage* of service *s*, measured on megabyte (MB);

(iv) *Dema*(*Type*,*s*) is the demand on *Type* of service *s*. This value indicates the type of device where the service *s* can be executed.

Sometimes, the output of a service is used as input of other services. Therefore, we regulate the executing priority between two services. Services having relationship together create a string of services. Relationship of each pair of service s_i and s_j is one of three types, including the following:

(i) Service s_i executes after service s_j.

(ii) Service s_i will execute before service s_j.

(iii) There is no rule on execution order between service s_i and service s_j.

Figure 2 shows us three types of a service string. They are sequence, parallel, and heterogeneous. In the top section of Figure 2, a set of five dotted services is sequence type; they will be executed in sequence. In the left side of Figure 2, there is a set of five dashed downward diagonal services; they will be executed parallelly with each other. Finally, a list of eight light upward diagonal services in the Right-Bottom side of Figure 2 is of heterogeneous type; it is executed complexly based on its regulated execution orders. For example, in the bottom-right section of Figure 2, s_6 will be ready to execute if service s_2 and service s_8 finish their execution already.

At one moment, every service has to get only one state. We define two functions in order to navigate to a service based on its *Service Relationships*. $Prev(s_i)$ is a set of services which have to be executed before s_i, and $Next(s_i)$ is a set of services which can be deployed only if s_i have already been executed. When a service changes its state, it will be able to cause state changes of related services. List of states of a service s_i is described follows:

(i) *Waiting*: when there exists at least one service in the set of $Prev(s_i)$ services which have not been executed yet,

(ii) *Ready*: when all services of the set of $Prev(s_i)$ services have already been executed, or there do not exist services which have to execute before s_i,

(iii) *Executing*: when service s_i starts executing on its hosting device,

(iv) *Executed*: when service s_i finishes executing on its hosting device,

(v) *Released*: when service s_i starts releasing resources on its hosting device.

For example in Figure 2, *service 2* and *service 5* are in the *ready* status, which means they are ready to execute when *service 1* is executed completely. We do not define execution order between *service 2* and *service 4*. This means that *service 2* is able to be executed before or after *service 4*.

An *application* is a list of service strings. An *application* is asked to be executed from a specific source device and assigned to send returned result to a specific destination device. We define an *application* as a 4-tuple, ($Devi_0$, $Devi_{n+1}$, S, SR), where

(i) $Devi_0$: a device from which such an application sends the request,

(ii) $Devi_{n+1}$: a device which will receive the returned result of the considering application,

(iii) S: a set of n *services* of the application,

(iv) SR: a set of 2-tuple *service relationships*, (s_i, $si + 1$), describing the executing priority between each two services. s_{i+1} can be deployed only if s_i have already been executed when there exists a relationship (s_i, $si + 1$).

3.2. Fog Node Architecture System. This paper uses the architecture of fog computing in [33]. *Fog cell* (also called *fog node*) is a software component installed on a device. It takes devices to have capacity on sharing computation, communication with each other. A set of fog cells combining with others create a small Edge computing network, called a *fog colony*. The most powerful fog cell in a fog colony is chosen to become the *orchestration node*. The orchestration node has roles on managing other fog cells in its colony. Fog cell f is represented in a 4-tuple, ($Capa(Comp, f)$, $Capa(RAM, f)$, $Capa(Storage, f)$, $Capa(Type, f)$), consisting of the following:

(i) $Capa(Comp, f)$: *Computing* capacity of fog cell f, measured on mega-instruction-per-second (MIPS),

(ii) $Capa(RAM, f)$: *RAM* capacity of fog cell f, measured on megabyte (MB),

(iii) $Capa(Storage, f)$: *Storage* capacity of fog cell f, measured on megabyte (MB),

(iv) $Capa(Type, f)$: *Type* capacity of fog cell f. This value indicates that fog cell f can execute a service which has the same *type* value.

The orchestration node monitors the changes of fog cells in its fog colony in order to create new relationships or remove unused relationships. A *relationship* between two fog cells f_i and f_j is a 2-tuple, ($Dist(f_i, f_j)$, $Velo(f_i, f_j)$), consisting of the following:

(i) $Dist(f_i, f_j)$: physical *distance* between fog node f_i and f_j, measured on meter (m),

(ii) $Velo(f_i, f_j)$: average *velocity* between fog node f_i and f_j, measured on megabyte-meter-per-millisecond (MBm/ms).

4. Dynamic Resource Provisioning on Fog Landscape

4.1. Notations and Variables. Given f_i is a fog cell, F_i is f_i's fog colony. Capacity of fog colony F_i is equal to its orchestration node. $C(C, f_i)$, $C(R, f_i)$, $C(S, f_i)$, $C(T, f_i)$ is *Computation*, *RAM*, *Storage*, and *Type* capacity of fog node f_i, respectively, described in Table 1.

Given application a with owned set of n services S, service s_i is a service of application a. Service s_i is described with a tuple ($Dema(Comp, s_i)$, $Dema(RAM, s_i)$, $Dema(Storage, s_i)$, and $Dema(Type, s_i)$), consisting of set of *Computation*, *RAM*, *Storage*, and *Type* demand of service s_i.

Because each service is an undividable processing unit, service s_i will be deployed on a single device, following a 4-tuple execution plan (s_i, f_i, t, c). An execution plan describes a service s_i which is deployed on device f_i at time t (millisecond) using c computing values (mega-instruction-per-second).

4.2. Objective Function. Given an application a, with S is a set of n services. The requester starts from device d_0, and the requester expects returned results to be sent to device $dn + 1$ given d_i is a device on which service s_i will be deployed. Our objective function is a combination of three components, $Time(Resp, a)$-the response time, $NU(a)$-the network usage, and $SU(a)$-server usage of application a. The objective function of our proposed strategy on choosing execution plan is described as

$$\min\left(\alpha * Time\left(Resp, a\right) + \beta * NU\left(a\right) + \delta * SU\left(a\right)\right) \quad (1)$$

4.2.1. Response Time. Given $Time(Resp, s_i)$ is the response time of service s_i, the response time of a service is the overall time using for data processing, e.g., waiting time, input transferring time, service placement time, service execution

Table 1: Notations.

Notation	Definition
f_i	Fog node f_i
$Capa(Comp, f_i)$	Current *Computation* capacity of fog node f_i
$Capa(\text{RAM}, f_i)$	Current *RAM* capacity of fog node f_i
$Capa(\text{Storage}, f_i)$	Current *Storage* capacity of fog node f_i
$Capa(\text{Type}, f_i)$	Current *Type* capacity of fog node f_i
$Dist(f_i, f_j)$	*Distance* between fog node f_i and f_j
$Velo(f_i, f_j)$	*Velocity* between fog node f_i and f_j
F_i	Fog colony of fog node f_i
$Capa(Comp, F_i)$	Current *Computation* capacity of orchestration node of fog colony F_i
$Capa(RAM, F_i)$	Current *RAM* capacity of orchestration node of fog colony F_i
$Capa(\text{Storage}, F_i)$	Current *Storage* capacity of orchestration node of fog colony F_i
$Capa(\text{Type}, F_i)$	Current *Type* of orchestration node of fog colony F_i
$Dist(F_i, F_j)$	*Distance* between orchestration node of fog colony F_i and F_j
$Velo(F_i, F_j)$	*Velocity* between orchestration node of fog colony F_i and F_j
a	Application a
S	List of n services of application a
s_i	Service s_i of application a
d_0	Device from which application a send the request
d_i	Device on which service s_i will be deployed
d_{n+1}	Device which will receive the returned result of application a
$Dema(\text{Comp}, s_i)$	*Computation* demand of service s_i
$Dema(\text{RAM}, s_i)$	*RAM* demand of service s_i
$Dema(\text{Storage}, s_i)$	*Storage* demand of service s_i
$Dema(\text{Type}, s_i)$	*Type* demand of service s_i
$Prev(s_i)$	Set of services of service s_i which have to execute before service s_i
$Next(s_i)$	Set of services of service s_i which have to execute after service s_i
$Time(Resp, a)$	Response time of application a
$Time(Resp, s_i)$	Response time of service s_i
$Time(Exec, s_i, n_j)$	Execution time when service s_i executes on fog node n_j
$Time(Tran, s_i, n_j, n_k)$	Transfer time when service s_i transfers from fog node n_j to fog node n_k

time, and returned result transferring. Let $Time(Resp, a)$ be the response time of application a. In the full-parallel case, all services s_i of application a are executed on device d_i parallelly; the overall response time of application a is equal to the maximum response time of each service s_i as

$$Time(Resp, a) = \max_{s_i \in S} (Time(Resp, s_i)) \quad (2)$$

If all services s_i of application a are placed and are executed on device d_i parallelly, then the overall response time $Time(Resp, a)$ of application a will be greater than or equal to the maximum services' response time. If all services s_i of application a are placed and are executed on device d_i in sequence, then the overall response time $Time(Resp, a)$ of application a will be less than or equal to the total of services' response time. Therefore, the limitation of the overall response time $Time(Resp, a)$ of application a is regulated in

$$Time(Resp, a) \leq \sum_{s_i \in S} (Time(Resp, s_i)) \quad (3)$$

The first subobjective function is in response time factor and is the overall response time $Time(Resp, a)$ of application a.

4.2.2. Network Congestion. The communication time which is used for sending and receiving data among devices plays an important role in the overall application response time. Therefore, other proposed methods try to deploying services closer to devices, which send input/receive results, for minimizing the overall response time of an application. Data is spreading around devices which make requests for service execution. The availability of devices, which are within the effected locations, becomes decreased significantly. Moreover, the communication time within the effected locations also decreases because of the wasted bandwidth. Thus, the other considered constraint is minimizing the amount of data which is sent and received on the network environment.

Let $NU(s_i, d_i)$ be the network usage of service s_i when s_i will deploy on device d_i given s_k is a service in the set of s_i's previous services. $Time(Tran, s_k, d_k, d_i)$ is the time for transferring service s_k output from device d_k to device d_i. We compute the network usage $NU(s_i, d_i)$ of service s_i deployed

on device d_i based on the total of the product of s_k's output data size (denoted $Dema(s_k)$) and the time for transferring s_k's output from device d_k to device d_i with each s_k in s_i's previous services, described as

$$NU(s_i, d_i) = \sum_{s_k \in Prev(s_i)} (Dema(Storage, s_k) * Time(Tran, s_k, d_k, d_i)) \quad (4)$$

The overall network usage $NU(a)$ of application a is computed by sum of network usage of each owned services. When making decision on choice of an execution plan, we try to reduce network usage of an application as Eq. (5)- the second subobjective function in network usage factor.

$$\min(NU(a)) = \min\left(\sum_{s_i \in a} NU(s_i, d_i)\right) \quad (5)$$

4.2.3. Server Usage. Let $SU(a)$ server usage of application a be the ratio of the number of centralized executed instruction over the number of application a instruction. The server usage value is computed based on the amount of cloud layer's server usage for execution application. It means the number of instructions which are deployed and executed on servers. It also means the cost which a user has to charge when using utilities of an application. Therefore, our proposed method is minimizing the server usage parameter in order to reduce overall cost of an application. The *server usage* criteria are calculated as the ratio of the number of centralized executed instruction over the number of application instruction. It means how much you have to pay when an application executes. Given $s_{Central}$ is a set of services which are deployed on cloud servers, server usage SU_a of application a is computing as

$$SU(a) = \frac{\sum_{s_i \in s_{Central}} Dema(Comp, s_i)}{\sum_{s_j \in S} Dema(Comp, s_j)} \quad (6)$$

When making decision on choice of an execution plan, we try to reduce server usage of an application as Eq. (5)- the last subobjective function in network usage factor.

Given d_i is the device on which service s_i will be placed and executed. Fog node f_j is chosen to become d_i if fog node f_j is satisfied on the following constraints.

4.3. Constraints

4.3.1. Basic Constraints. Firstly, given $Dema(RAM, s_i)$ is the RAM demand of service s_i, it shows the necessary minimum amount of memory for service s_i deployment. Given device d_i is the device which is chosen for deploying service s_i, d_i has to satisfy the RAM constraint as

$$Dema(RAM, s_i) \le Capa(RAM, d_i) \quad (7)$$

Additionally, $Dema(Storage, s_i)$ is the storage demand of service s_i, measured on megabyte. It shows the necessary minimum amount of storage for service s_i's input, output, and instructor(s). Hence, if device d_i is the device which is chosen for deploying service s_i, then d_i has to satisfy the Storage constraint as

$$Dema(Storage, s_i) \le Capa(Storage, d_i) \quad (8)$$

The type value of service s_i is $Dema(Type, s_i)$, which defines that service s_i is only deployed on devices with the same type value $Capa(Type, f_i)$. For example, it is suited to deploy a service on the device which sends this execution request or on nearing devices if the mission of this service is only raw data preprocessing. On the other side, a service which requires the complex processing should be deployed on cloud layer's servers. The constraint about service type is regulated in

$$Dema(Type, s_i) = Capa(Type, f_i) \quad (9)$$

4.3.2. Response Time. The *response time* $Time(Resp, s_i, f_j)$ is the estimated response time of deploying service s_i in fog node f_j. The response time of a service is the overall time using for data processing, e.g., waiting time, input transferring time, service placement time, service execution time, and returned result transferring. Given service s_k in a service in $Prev(s_i)$, the set of previous services of s_i, we denote the time, when service s_k's output is transferred from devices d_k to fog node f_j as $Time(Tran, s_k, d_k, f_j)$. $Time(Exec, s_i, f_j)$ is the time for executing s_i on fog node f_j. If service s_i has more than two services in the previous services list, the period which is required for transferring input data from previous services' deployed devices to device f_i is the maximized value of each previous service. The response time $Time(Resp, s_i, f_j)$ of deploying service s_i on fog node f_j is estimated as

$$Time(Resp, s_i, f_j) = \max_{s_k \in Prev(s_i)} (Time(Tran, s_k, d_k, f_j) + Time(Exec, s_i, f_j)) \quad (10)$$

For example, given a set of three services s_1, s_2, s_3 with two relationships (s_1,s_3), (s_2,s_3), *Option*1 shows us that we try to calculate for making decision on deploying service s_3. We estimate the transfer time from device d_1 and d_2 to consider fog node f_j is $Time(Tran, s_1, d_1, f_j) = 3.5$s and $Time(Tran, s_2, d_2, f_j) = 3.3$s, respectively. If we choose device f_j for deploying service s_i, then the response time of service s_i is equal to 5.5s. If there are more than two suitable fog nodes (f_j and f_k), we choose the fog node which has the less estimated response time as Eq. (11) for deploying this service.

$$Time(Resp, s_i) = \min_{f_j \in F} (Time(Resp, s_i, f_j)) \quad (11)$$

The *transfer time* $Time(Tran, s_i, f_j, f_k)$ is the period when service s_i is transferred from fog node f_j to fog node f_k as described in (12). For example, given s_{i-1} is the only service in set $Prev(s_i)$. d_{i-1}, f_i are devices on which services s_{i-1}, s_i are deployed, respectively. Fog device d_{i-1}

stores returned result (12 megabyte data) of service s_{i-1} after it is executed. The distance between device d_{i-1} and f_i is 10 meters, and its average velocity is 30 megabyte-meter-per-second. Consequently, the *trans time*, when service s_{i-1} transfers its returned result from device d_{i-1} to f_i, is 12 x 10 / 30 = 4 seconds.

$$Time\left(Tran, s_i, f_j, f_k\right) = \frac{Dema\left(Storage, s_i\right) * Dist\left(f_j, n_k\right)}{Velo\left(f_j, f_k\right)} \tag{12}$$

As we have introduced in Section 3.2, the orchestration node on each fog colony plays an important role to manage other fog nodes in it owning fog colony. Therefore, the distance of two fog nodes f_i, f_j is calculated based on a path through their owned orchestration nodes F_i, F_j in Figure 3. The distance value and the average velocity value of two fog nodes f_i and f_j arbitrarily are calculated as (13) and (14). Because the distance between two fog colonies is a positive number, we apply Dijkstra's algorithm [34] to find the shorted path between every two fog colonies.

$$Dist\left(f_i, f_j\right) = Dist\left(f_i, F_i\right) + Dist\left(F_i, F_j\right) + Dist\left(F_j, f_j\right) \tag{13}$$

$$Velo\left(f_i, f_j\right) = \frac{Dist\left(f_i, f_j\right)}{Dist\left(f_i, F_i\right)/Velo\left(f_i, F_i\right) + Dist\left(F_i, F_j\right)/Velo\left(F_i, F_j\right) + Dist\left(F_j, f_j\right)/Velo\left(F_j, f_j\right)} \tag{14}$$

The *execution time* $Time(Exec, s_i, f_j)$ is period when service s_i will be executed on fog node f_j, as defined in (15). For example, the execution time when service s_i (having computation demand: 300 mega-instruction) is deployed on fog node f_j (using computation capacity: 30 mega-instruction-per-second) takes 10 seconds. When estimating execution time of a service, we have one special case. Because computation capacity of cloud servers is unlimited, we assume that the execution time value of a service when executing on cloud servers is a tiny value, namely, about 1 millisecond.

$$Time\left(Exec, s_i, f_j\right) = \frac{Dema\left(Comp, s_i\right)}{Capa\left(Comp, f_j\right)} \tag{15}$$

Making decision on which device d_i will execute a service s_i is different among provisioning resource solutions, because of various criteria. Our proposal focuses on minimizing the total response time of the whole application. There are so many candidate devices which have enough capacity on deployment of service s_i. We try to deploy service s_i on device d_i which has the minimal service response time. This is regulated in (16).

For example, in Figure 4, given a set of three services s_1, s_2, s_3 with two relationships (s_1,s_3), (s_2,s_3), at current time $t = 0$, we make some calculations in order to estimate the response time of service s_3. We find that there are two candidates *option 1*, *option 2*. Although the transfer time to device d_i: $\max(Time(Tran, s_1, d_1, f_i), Time(Tran, s_2, d_2, f_i))$ in *option 1* is less than in *option 2*, we choose *option 2* for deployment service s_3 into device f_k. We know that there is a trade-off between *transfer time* and *execution time*. In *option 1*, the transfer time to device f_i lasts less than in *option 2*, but the computation capacity on *option 2* is stronger than *option 2*. As we see in Figure 4, the overall response time of service s_3 in *option 1* (=5.5s) lasts greater time than *option 2* (=4.8s). Therefore, we make decision on deployment service s_3 following *option 2* because of its minimal response time value.

$$\begin{aligned}\min\left(Time\left(Resp, s_i, f_i\right)\right) &= \min\left(Time\left(Trans, Prev\left(s_i\right), f_i\right)\right. \\ &\quad \left. + Time\left(Exec, s_i, f_i\right)\right)\end{aligned} \tag{16}$$

$$= \min\left(\max_{s_k \in Prev(s_i)}\left(Time\left(Trans, s_k, d_k, f_i\right)\right) + Time\left(Exec, s_i, f_i\right)\right) \tag{17}$$

4.4. Service Deployment Policy

4.4.1. Cloudy Service Deployment Strategy. The cloud service policy is a service deployment policy in which service deployment follows the traditional cloud paradigm. Every service has to send preprocessing data to the cloud centralized server(s) for execution through the internet. Thus, cloud server processes request and then transfer postprocessing results from the executing cloud server to the target devices. In this case, we assume that devices (exception of cloud servers) do not have enough capacity on computation, storage, and RAM to execute a service. The cloud service policy is a base-line for evaluating our proposed deployment method in terms of the response time, network usage, and server usage.

4.4.2. Foggy Service Deployment Strategy. Service deployment is a process of matching the list of n services to list of n executed fog nodes based on many criteria. We use recursive approach to simplify this problem solving. Because it is simple to deploy a service on list of m executing fog nodes, we find out that it is easier to match the list of $n - 1$ services than matching list of n services. Therefore, our idea is trying

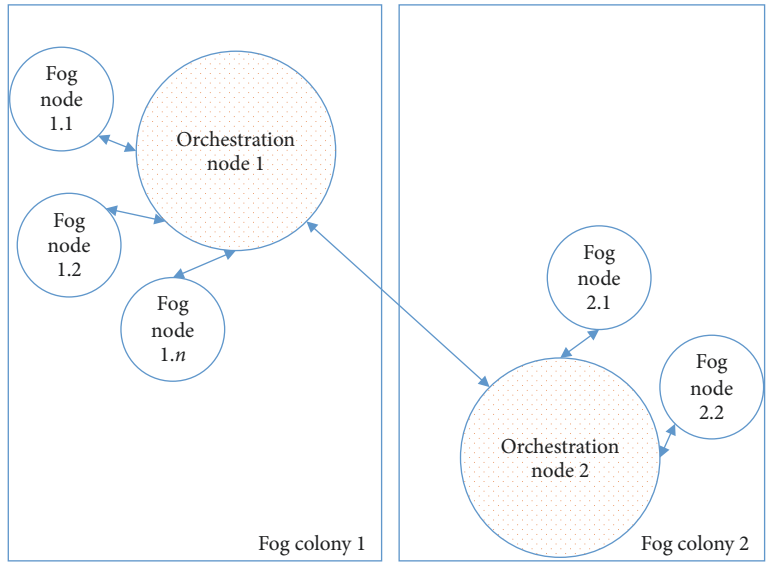

Figure 3: The distance between two fog nodes.

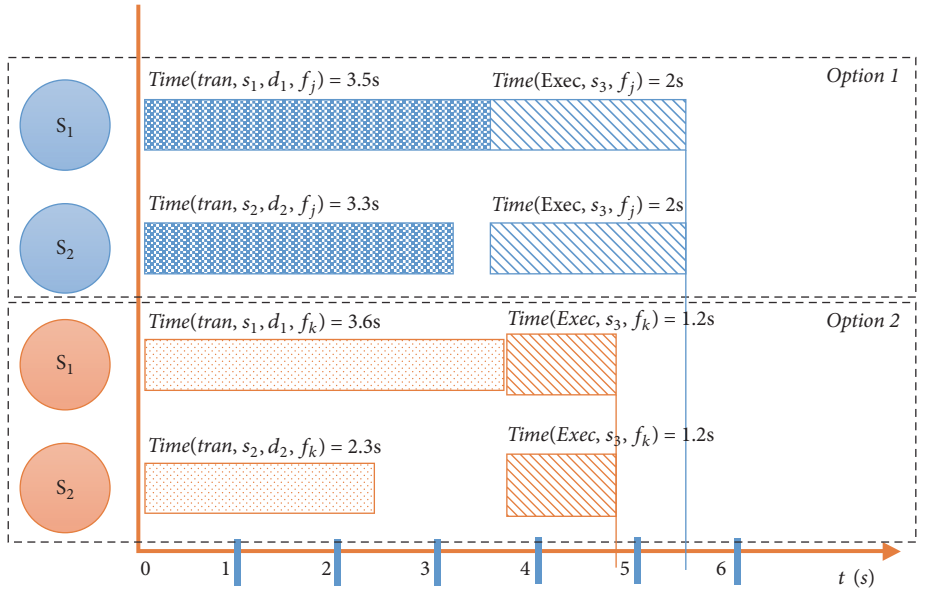

Figure 4: A sample of service deployment.

to deploy a service on a fitted device firstly; then, we resolve this problem with smaller scope.

Our method is described in Algorithm 1. When deploying a service on a device, we consider about types of criteria, including hard type and soft one. Hard criteria are constraints on *RAM*, *Storage*, and *Type* demand of a service which is described in (7), (8), and (9). They are conditions which have to be satisfied certainly. In line 5 of Algorithm 1, we use method *CanDeploy*(*EP*, *S*, *FN*, *t*) for finding the list of current devices which have strong enough capacity for listing of available services at time *t*. *Available-Services*, *Available-Nodes* in lines 6, 7 are lists of items with contextual data when at time *t*, deployed services, and executing devices of *Execution Plans EP*. Line 8 estimates deployment time for finding the best deployment solution of a service, based on the objective function in (1). Then, the system has to update fog nodes' states and services' states; for example, if service *s*- has 3 MB on RAM- placed into fog node *n*, the current RAM capacity of fog node *n* will decrease 3 MB. If there does NOT exist any available services at time *t*, it will try to estimate in order to deploy the service at next period *Next*(*t*). *Next*(*t*) is calculated by Algorithm 2.

We explain in more detail line 8 in Algorithm 1. If we choose the best candidate service deployment by basing only on policy minimizing the response time of an application, the following services of a cloudy deployed service will be deployed almost in cloud servers. Therefore, we add the *network usage* and the *server usage* criteria to make service

```
1: procedure DEPLOY(EP, S, FN, t)                    ▷ return EP
2:     if EP is deployed completely then
3:         return EP
4:     else
5:         if Can-Deploy(EP, S, FN, t) == true then
6:             for s in Available-Services do
7:                 for fn in Available-Nodes(s, FN) do
8:                     Finding the best candidate( f0, s0)
9:             Deploy( f0, s0, t)
10:            if Has-Deployment == true then
11:                return Deploy(EP, S, FN, t)
12:            else
13:                return Deploy(EP, S, FN, Next(t))
14:        else
15:            return Deploy(EP, S, FN, Next(t))
```

ALGORITHM 1: Recursive service deployment algorithm.

```
1: procedure NEXT(EP, t)                             ▷ Return t
2:     result = t
3:     for ep in EP do
4:         if At-Executing(ep) ≥ t then
5:             if At-Executing(ep) ≤ result then
6:                 result = At-Executing(ep)
7:         else
8:             if At-Executed(ep) ≥ t then
9:                 if At-Executed(ep) ≤ result then
10:                    result = At-Executed(ep)
11:    return result
```

ALGORITHM 2: Get a next time value.

deployment more effective and flexible. The *network usage* criteria are calculated on the amount of data transferred on networks in specified period.

We find out it is necessary to update the *Available-Services* and the *Available-Nodes* at two moments. They are the time when a service is being executed on a device, and the time when a service has just executed on a device. Therefore, next time of time t is the moment which not only is greater than t, but also has minimal difference with t. Algorithm 2 describes procedure *Next* in detail.

Given n, m is the number of fog nodes, and the number of services, we use *Big-O* notation to describe the performance or complexity of our algorithm. Because our system gets the current states of fog nodes, and services; for each service executing, our system costs n times for comparing with all current fog nodes, and one time for assignment the most suited fog node to the executing one. In the best case (all services have the capacity for executing in parallel); the complexity of our algorithm is $O(m*n)$. In the worst case (all services have to be executed in sequence), the complexity of our algorithm is $O(m*(1+m)/2*n)$. In average case (all services are regulated to executed follows tree structure with k-child nodes), the complexity of our algorithm is $O(\log_k(m)*k*(m*n))$.

5. Experimental Results

5.1. Application. We consider on an application type which defines service execution priority already, because this type covers all service combinations of any application. In service strings, a service uses previous services' returned results as input data. In other words, a service sends processed data to its next services as input data. An application executed with different execution plans will return various results significantly. Different orderings of the same service combination have different meaning. There is a strict rule which requires some particular services having to be executed on cloud servers only. In practice, some services can be executed on fog devices and cloud servers, e.g., computing simple operations and raw data preprocessing. Other services can be executed only in cloud servers because this service type requires strong resources on computing, storage, e.g., data collecting from various sources, mining historical stored data. Therefore, we add the *type* attribute for services, devices and a strict rule defining that a service is executed on a device if they have the same *type* attribute value.

We introduce an application which finds the nearest direction between two locations with awareness of real-time traffic information. The nearest direction between two

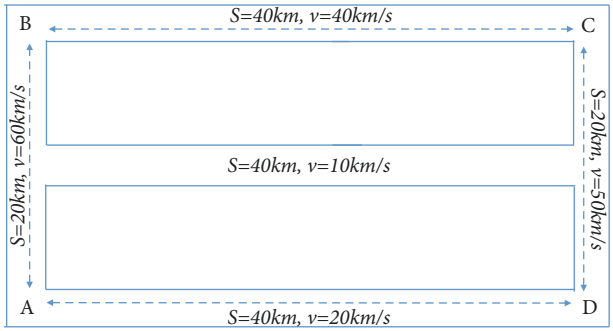

Figure 5: A sample of fog computing application.

locations is the way which we drive from the source point to the target point with the less time. The closest in terms of physical distance direction between two locations is NOT the nearest direction between them, because of traffic problems, for example, traffic jam, accidents, and one way road. This model includes a set of public live traffic cameras in the bottom layer which play the role for capturing the traffic flows, density of vehicles on roads. The second important part is the system of mediate linked network devices which support communication among devices. Mediate linked network devices have limited capabilities on computing, storage. The final important part is cloud stage, which is a set of linked servers, known as unlimited computing, storage devices.

In this application, we divide it into two execution parts:

(1) Find available directions between two locations on the cloud stage.

(2) From each determined junction node, we analyze local traffic flow, density of vehicles on roads based on systems of public live traffic cameras in order to make decision on driving, for example, go straight, turn left, and turn right.

For example, Figure 5 shows a sample of finding route between two points in map. If route AD is shorter (in terms of physical distance) than route ABCD, but route AD has more crowded density of vehicles than ABCDB right now, the velocity of a vehicle on route AD may be NOT as fast as on route ABCD. Consequently, the driver on route AD takes a longer time than that on route ABCD. This is the reason why our application has to recheck the current traffic information in order to decide which way to drive dynamically. If we drive from A to D directly, it will take us 2 hours. If we drive from A to D through B and C, it will take us x hours. This is the total time we have to cost 26/15 hour. It is total of three components, 1/3 hour on route AB, 1 hour on route BC, and 2/5 hour on route CD.

5.2. Experimental Setup. We have conducted six experiments with two variables, namely, types of service strings (application types) and number of services. A list of application types includes sequence, parallel, and heterogeneous types, as we describe in Section 3.1. For each application type, we have conducted experiments for two cases: an application

Table 2: The configuration of services and of fog devices.

	RAM (MB)	CPU (MI)	Storage (MB)	Type
Fog cell	80..160	2..10	20..50	1..5
Orchestration	200..300	5..20	40..100	1..5
Service	50..70	40..60	10..20	1..5

Table 3: The relationship between two fog devices.

	Distance (meter)	Velocity (MBmpMS)
Cloud - Fog	2000..3000	5..10
Fog - Fog	70..80	1..5
Fog - Node	1..10	1..2

with 20 services, and 50 services, described in Table 4. Each service requires CPU, RAM, and storage resources of which values are randomized from ranges 2..10 mega-instruction-per-second (MIPS), 50..70 megabyte (MB), and 10..20 megabyte (MB) accordingly, described in Table 2. In heterogeneous applications, if a randomized number for each pair of services (from 0 to 100) is less than or equal to 70, we assume that there exists a relationship on execution order of them. Each service has a *type* attribute which is a randomized number from 1 to 5. This value indicates which devices are able to execute the considering service.

Figure 6 describes the structure of fog computing system which is the same in all six experiments. This is a tree architecture whose each node presents for a fog colony. Root of tree is a special fog colony; it includes unlimited-capacity cloud servers. Each node in tree has two child nodes. The height of tree equals four. Each fog colony includes one orchestration control node and nine other fog cells. The values of capacity on fog cells' CPU, RAM, and storage resources are randomized from ranges 2..10 mega-instruction (MI), 80..160 MB, and 20..50 MB accordingly. The respective resources of a fog orchestration control node also are randomized from ranges 5..40 MI, 200..300 MB, and 40..100 MB. Each fog cell has a *type* attribute which is a randomized number from 1 to 5 as described above.

The communication link delays between two fog nodes in a fog colony and between two fog colonies have configuration with parameters described in Table 3. In reality, the communication link delays depend on the physical distance between resources, type of connection, e.g., wireless and fiber.

For each of six configurations, we deploy services following cloudy and foggy strategies which are described in Section 4.4. Firstly, we put all services into cloud, called cloudy scenarios. Then, we put services into fog landscape partially based on their deployment constraints called foggy scenarios. The differences of six configurations are the number of fog nodes, the number of services. We assume that capacity of fog nodes and demand of services do not change so much among six configurations. We find that the more the number of fog nodes, the more powerful the fog landscape's computation capacity. Therefore, services are

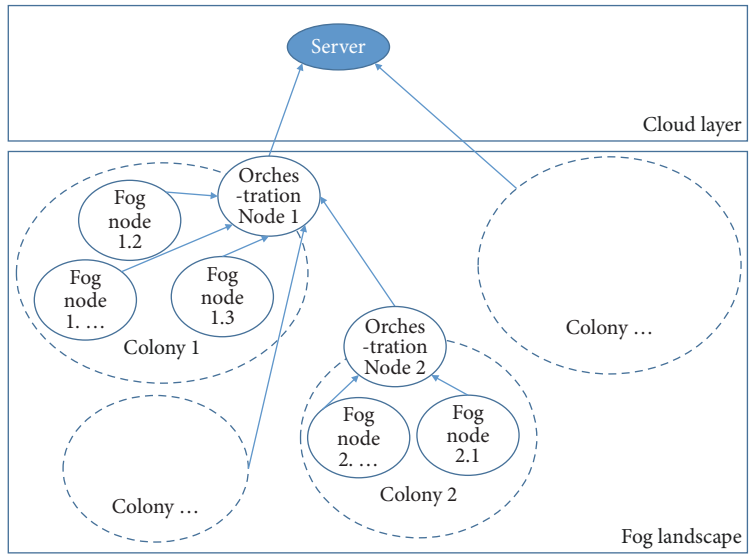

Figure 6: A sample of fog landscape structure.

Table 4: The configuration of six scenarios.

	Level / Sub colonies	Sub fog nodes	Number / Service type
Config 1	4 / 2	10	20 / Sequence
Config 2	4 / 2	10	20 / Parallel
Config 3	4 / 2	10	20 / Heterogeneous
Config 4	4 / 2	10	50 / Sequence
Config 5	4 / 2	10	50 / Parallel
Config 6	4 / 2	10	50 / Heterogeneous

placed and are executed on fog landscape usefully parallelly. On the other hand, the number of services is too large for fog landscape to execute completely parallelly. So, the system costs so much time to find the suited nodes for executing services. In that case, the system places and executes services in sequence. The differences of six configurations point out the advantages between foggy and cloudy service deployment strategies.

Our objective function is combined from three components $Time(Resp, a)$-the response time, $NU(a)$-the network usage, and $SU(a)$-server usage with three weighted values α, β, and δ. Because of the different domains of three components in our objective function, we assume that α =1, $\beta = \delta = 0$ in this work. This means that the response time of a service is the only important factor on finding the most suitable solution to deploy a service on a device. Therefore, $NU(a)$-the network usage and $SU(a)$-server usage will become criterion used for comparing different sides among our six configurations. However, in general case we have to consider other components. For example, if your application often disturbs near devices in order to try service deployment, then the availability of near effected devices goes down. Thus, our proposed method also considers other criteria in service deployment.

5.3. Results and Discussion. The differences of six configurations are the combinations of three types of application (sequence, parallel, and heterogeneous) with two sizes of application (20, and 50 services). We set up our systems that the capacity of fog nodes and the demand of services do not change so much (details in Table 4) among six configurations. We decide to choose that six configurations because we want to prove that the effectiveness will increase significantly if the number of services increases in the similar fog landscape. Regarding the response time of configuration 2, 5, we find that in the same fog landscape if our simulated services are able to be executed in utilized parallel, our proposed service deployment algorithms work significantly effective. On the other hand, the number of services is too great for our fog landscape to execute available services in parallel. Therefore, our fog landscape has to transfer the overloaded services into cloud servers to execute. That is the reason why the benefit on response time between foggy and cloudy service deployment strategy is not significantly effective in configuration 6.

5.3.1. Response Time. Figure 7 shows us the response time of six scenarios. Because services in parallel application (configuration 2 and 5) do not need to wait for previous completed services, the system tries to deploy the number of services on

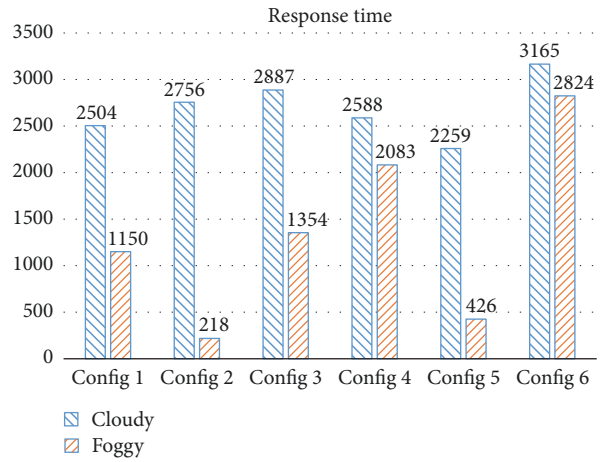

Figure 7: The response time of six experimental configurations.

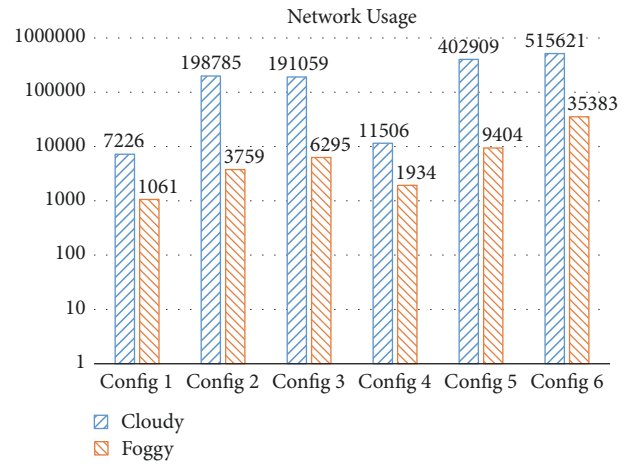

Figure 8: The network congestion of six experimental configurations.

fog landscape as good as possible. In a parallel application, the response time of the foggy strategy is less than that of the cloudy strategy significantly. In configuration 1 and 3, the 20-service application has the response time of foggy strategy equal to a half of the cloudy strategy approximately. In other complex scenarios, the response time of an overall application is approximately equal to communication to cloud servers. When a service is deployed in cloud servers, list of next services will be deployed in cloud servers too because of the minimum of transmission time and execution time. Therefore, the response time of the foggy strategy in configuration 4 and 6 is less than that on the cloudy strategy significantly.

In fact, the response time rate between foggy scenarios and cloudy scenarios has significant effects in parallel cases with a great number of fog cells, because every service can be deployed if constraints are satisfied. If fog landscape has a great number of powerful fog cells, then system tries to find out a fog cell fits for service deployment. It means that services try to be deployed spreading on fog landscape. The response time rate between the foggy scenarios and the cloudy scenarios decreases gradually following from Parallel ⟶ Heterogeneous ⟶ Sequence cases, and a decrease of the number fog cells.

5.3.2. Network Congestion. In this work, we consider the communication between fog devices as a physical connection. When a device sends a request to or receives returned results from other devices so much, the link of this device's connection is always on busy. This is the situation when it is difficult or slow for your devices to connect to other devices. We have conducted an experiment of *network congestion* metric for comparing with six above scenarios. The value of network congestion attribute is computed by the weight of transfer data multiple with transfer time. Figure 8 shows us clearly that cloudy scenarios have a great amount of data transferring over devices' limited connections. Therefore, sending data too much is not a balanced solution for service deployment. We should utilize resources of the local and the global sides for reducing latency cost as much as possible.

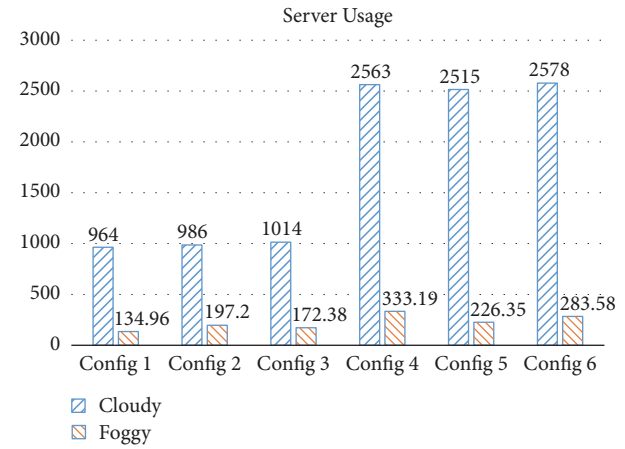

Figure 9: The server usage of six experimental configurations.

5.3.3. Server Usage. The final considered parameter is server usage. The service usage is computed based on the amount of cloud layer's server usage for execution application. It means the cost which a user has to charge when using utilities of an application. Therefore, our proposed method is minimizing the server usage parameter in order to reduce overall cost of an application. Figure 9 shows us that six deployment plans of foggy strategy cost less than that on the cloudy strategy, because of our above assumption about service deployment policy in Section 4.4.

6. Conclusion

In this work, we propose a novel method for service deployment on fog landscape. Our strategy is to combine three considered components (application's response time, network congestion, and server usage) into one objective function. The best service deployment, we proposed, is an execution plan which has trade-off among minimizing application's response time, minimizing network congestion, and minimizing cloud layer's server usage.

In this work, we consider security risks on data-in-transit among devices, data-in-process on vulnerable executing devices, and privacy of data which is divided into executed parts in other devices. The existed security issue raising one of our future research directions is data protecting. This will prove that fog computing not only is an effective solution in latency applications but also is a secured one.

In future work, we will consider device's energy consumption because of mobile devices with limited life battery. Utilization of computing sharing among near mobile devices made the application's response time faster, but it disturbs other users by losing devices' life battery fast. We have conducted more experimental configuration on well-known fog computing simulation environments, e.g., iFogSim [24].

One of our future research directions is finding out β, δ value. Firstly, we have conducted experiments with a great number of situations. Secondly, we classify those situations' configurations into approximated classes. For each class, we can deduce or update the value of beta, delta by averaged methods when our system runs with the significant number of deployment plans. Finally, we can apply accordingly β, δ into finding optimized deployment plans in other related configurations.

Data Availability

The simulated evaluation data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research is funded by Ho Chi Minh City University of Technology, VNU-HCM, under grant number BK-SDH-2019-1880316.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the 1st ACM Mobile Cloud Computing Workshop, MCC 2012*, pp. 13–15, ACM, New York, NY, USA, 2012.

[2] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[3] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "The nist definition of fog computing," Special Publication (NIST SP)-800-191, 2017.

[4] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," Special Publication (NIST SP)-500-325, 2018.

[5] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2018.

[6] R. K. Naha, S. Garg, D. Georgakopoulos et al., "Fog computing: survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[7] M. A. Nadeem and M. A. Saeed, "Fog computing: an emerging paradigm," in *Proceedings of the 6th International Conference on Innovative Computing Technology, INTECH 2016*, pp. 83–86, IEEE, Dublin, Ireland, August 2016.

[8] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support IoT applications," *IET Networks*, vol. 5, no. 2, pp. 23–29, 2016.

[9] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "Mist," *Journal of Network and Computer Applications*, vol. 82, no. C, pp. 152–165, 2017.

[10] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: principles, architectures, and applications," in *Internet of Things Principles and Paradigms*, pp. 61–75, Morgan Kaufmann, Oxford, UK, 1st edition, 2016.

[11] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare-a review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[12] A. A. Mutlag, M. K. A. Ghani, N. Arunkumar, M. A. Mohammed, and O. Mohd, "Enabling technologies for fog computing in healthcare iot systems," *Future Generation Computer Systems*, vol. 90, pp. 62–78, 2019.

[13] A. Kumari, S. Tanwar, S. Tyagi, and N. Kumar, "Fog computing for Healthcare 4.0 environment: opportunities and challenges," *Computers and Electrical Engineering*, vol. 72, pp. 1–13, 2018.

[14] K. Kai, W. Cong, and L. Tao, "Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues," *Journal of China Universities of Posts and Telecommunications*, vol. 23, no. 2, pp. 56–96, 2016.

[15] J. Liu, J. Li, L. Zhang et al., "Secure intelligent traffic light control using fog computing," *Future Generation Computer Systems*, vol. 78, pp. 817–824, 2018.

[16] T. Shuminoski, S. Kitanov, and T. Janevski, "Advanced QoS provisioning and mobile fog computing for 5G," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 5109394, 13 pages, 2018.

[17] P. S. Khodashenas, R. Cristina, S. S. Muhammad, A. Betzler, and J. F. Riera, "The role of edge computing in future 5G mobile networks: concept and challenges," in *Cloud and Fog Computing in 5G Mobile Networks: Emerging Advances and Applications*, pp. 349–370, IET, Stevenage, UK, 2017.

[18] N. Chen, Y. Chen, X. Ye, H. Ling, S. Song, and C. Huang, "Smart city surveillance in fog computing," in *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, Chapter 9, pp. 203–226, Springer International Publishing, Cham, Switzerland, 2017.

[19] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1728–1739, 2016.

[20] M. Mukherjee, R. Matam, L. Shu et al., "Security and privacy in fog computing: challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[21] A. Alrawais, A. Alhothaily, C. Hu, X. Xing, and X. Cheng, "An attribute-based encryption scheme to secure fog communications," *IEEE Access*, vol. 5, pp. 9131–9138, 2017.

[22] T. D. Dang and D. Hoang, "A data protection model for fog computing," in *Proceedings of the 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017*, pp. 32–38, IEEE, Valencia, Spain, May 2017.

[23] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in Fog computing: a survey," *Future Generation Computer Systems*, vol. 88, pp. 16–27, 2018.

[24] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[25] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: a simulator for virtual machine migration in fog computing," in *Proceedings of the 10th International Conference on Utility and Cloud Computing, UCC 17 Companion*, pp. 47–52, ACM, New York, NY, USA, 2017.

[26] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for IoT services in the fog," in *Proceedings of the 9th IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2016*, pp. 32–39, IEEE, Macau, China, November 2016.

[27] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, 2017.

[28] N. Mostafa, I. A. Ridhawi, and M. Aloqaily, "Fog resource selection using historical executions," in *Proceedings of the 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018*, pp. 272–276, IEEE, Barcelona, Spain, April 2018.

[29] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 283–294, 2018.

[30] J. Zhang, X. Hu, Z. Ning et al., "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[31] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proceedings of the IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, IEEE, Atlanta, GA, USA, May 2017.

[32] P. Casas and R. Schatz, "Quality of experience in cloud services: survey and measurements," *Computer Networks*, vol. 68, pp. 149–165, 2014.

[33] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: a platform for internet of things and analytics," in *Big data and internet of things: a roadmap for smart environments*, vol. 546 of *Studies in Computational Intelligence*, pp. 169–186, Springer, Cham, Switzerland, 2014.

[34] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein, "Dijkstra's algorithm," in *Introduction to Algorithms*, ch. Section 24.3, pp. 595–601, The MIT Press, Cambridge, Mass, USA, 2nd edition, 2001.