Cluster Computing
https://doi.org/10.1007/s10586-021-03338-9

Towards energy-efficient service scheduling in federated edge clouds

Yeonwoo Jeong[1] · Esrat Maria[1] · Sungyong Park[1]

Received: 28 November 2020 / Revised: 4 June 2021 / Accepted: 9 June 2021

Abstract

This paper proposes an energy-efficient service scheduling mechanism in federated edge cloud (FEC) called ESFEC, which consists of a placement algorithm and three types of reconfiguration algorithms. Unlike traditional approaches, ESFEC places delay-sensitive services on the edge servers in nearby edge domains instead of clouds. In addition, ESFEC schedules services with actual traffic requirements rather than maximum traffic requirements to ensure QoS. This increases the number of services co-located in a single server and thereby reduces the total energy consumed by the services. ESFEC reduces the service migration overhead using a reinforcement learning (RL)-based reconfiguration algorithm, ESFEC-RL, that can dynamically adapt to a changing environment. Additionally, ESFEC includes two different heuristic algorithms, ESFEC-EF (energy first) and ESFEC-MF (migration first), which are more suitable for real-scale scenarios. The simulation results show that ESFEC improves energy efficiency by up to 28% and lowers the service violation rate by up to 66% compared to a traditional approach used in the edge cloud environment.

Keywords Energy-efficient · Federated edge cloud · Service scheduling · Reinforcement learning

1 Introduction

A federated edge cloud (FEC) [2] is an edge cloud environment [3] where multiple edge servers in a single administrative domain collaborate together to provide real-time services. This environment reduces the possibility of violating the quality of service (QoS) requirements of target services by locating delay-sensitive services at nearby edge servers instead of deploying them on the clouds. However, as the number of edge servers increases in FEC, the amount of energy consumed by servers and network switches also increases [4]. Considering that the energy consumption and QoS depends on which server a service is deployed to, it is necessary to devise an efficient service scheduling strategy that satisfies service QoS while reducing energy consumption in FEC.

There have been a large number of research activities for scheduling services to reduce energy consumption in multi-cloud or edge clouds. However, most of them have focused on scheduling services using maximum traffic requirements regardless of their actual traffic usage. Although these approaches can ensure service QoS, they prevent services from being co-located even when the traffic volume is quite low. This leads to low resource utilization and unnecessary energy consumption. It is reported that the average CPU utilization of a server cluster is only about 50–60% [5]. Furthermore, service migration scenarios are not taken into account because services are scheduled based on their maximum traffic requirements. This assumption is not suitable for our target environment because the computing capacity of each edge server in the FEC is quite limited.

This paper proposes an energy-efficient service scheduling mechanism in the FEC called ESFEC that minimizes energy consumption on the service path, while ensuring QoS at the same time. In order to prevent physical resources from being wasted, this paper considers placing

A preliminary version of this article [1] was presented at the 2020 IEEE 1st International Workshops on Autonomic Computing and Self-Organizing Systems (ACSOS), Washington DC, USA, August, 2020.

✉ Sungyong Park
parksy@sogang.ac.kr

Yeonwoo Jeong
akssus12@sogang.ac.kr

Esrat Maria
esratmaria@sogang.ac.kr

[1] Department of Computer Science and Engineering, Sogang University, 35, Baekbeom-ro, Mapo-gu, Seoul, Republic of Korea

services with their actual traffic requirements along the service path and gradually reconfigures the path as the amount of traffic increases. This approach can save energy consumption when traffic is low compared to service scheduling with maximum traffic requirements. However, when traffic is high, physical resources are quickly overloaded and the number of service migration can increase. This also affects energy consumption [6]. Thus, we devised a reconfiguration mechanism to minimize migration overhead, which also has an effect on overall energy consumption.

ESFEC initially places delay-sensitive services in edge servers based on their minimum CPU utilization requirements when service requests arrive. By periodically monitoring traffic fluctuation and volume in each edge server, ESFEC reconfigures the service path when the CPU utilization of any edge server exceeds the predefined threshold value. All decisions are made by simultaneously considering the minimization of total energy consumption, the reduction of QoS violation, and the minimization of migration overhead.

ESFEC provides a reinforcement learning (RL)-based reconfiguration algorithm called ESFEC-RL that enables the idea of an agent to learn to survive in an unknown environment with no prior knowledge. Whenever a host is over-utilized, ESFEC-RL immediately reacts to the situation by migrating a potential virtual machine (VM) to an optimal host. The optimal host is chosen by the learning agent which not only reduces energy consumption but also efficiently deals with migration overhead. In a federated environment, the latency along the path and the number of hops may vary. The learning agent chooses an optimal host that requires minimal migration energy along the path and immediately chooses the destination host because the longer a host stays over-utilized the greater the energy consumption.

ESFEC also provides two heuristic-based reconfiguration algorithms called ESFEC-EF (energy first) and ESFEC-MF (migration first). ESFEC-EF focuses more on an energy-aware approach by placing services in a nearby edge server to minimize excess energy consumption along the path, whereas ESFEC-MF is focused on the minimization of the number of migrations. Although the heuristic algorithms are more suitable for real-scale scenarios, a learning-based algorithm is proven to be much more efficient in the long term since it always has an opportunity to learn in an environment that can change at any time.

We conducted a simulation using a cloud simulator called `CloudSimSDN`. The simulation results show that ESFEC outperforms the traditional approach used in the edge cloud environment in terms of energy consumption and service violation rate by up to 28% and 66%, respectively.

In summary, this paper makes the following specific contributions:

- The mechanism proposed in this paper is the first attempt to address energy efficiency issues in the FEC.
- To increase the number of co-located services, ESFEC places services based on their actual traffic requirements instead of their maximum requirements. This is in contrast to traditional approaches used in multi-cloud and edge clouds
- ESFEC proposes an RL-based reconfiguration algorithm to reduce the service migration overhead. In addition, it also provides two heuristic algorithms that are more suitable for large-scale scenarios.

2 Related work

Although few research efforts have been made to minimize energy consumption in FEC, various research activities for scheduling services in terms of energy efficiency and QoS guarantee are conducted in multi-cloud and edge cloud environments. This section provides a brief summary of each activity and discusses the limitations of previous works.

Kim et al. [7] proposed a dynamic virtual network function (VNF) placement and reconfiguration algorithm to minimize energy consumption while ensuring service QoS. Abdessamia and Tian [8] proposed an energy-efficient virtual machine placement based on the binary gravitation search algorithm to assure cloud consolidation and blueuce power wastage. Tarahomi et al. [9] presented an efficient power-aware VM allocation in cloud data center. They utilized a micro-genetic algorithm to dynamically consolidate cloud servers using live migration. Sun et al. [10] proposed an energy-efficient and traffic-aware service function chaining (SFC) orchestration in multi-domain networks. They proposed an online service scheduling mechanism to minimize energy consumption in servers and network links. Shang et al. [11] proposed a network congestion-aware service placement and load balancing mechanism. This paper suggests that the algorithm can minimize operation cost and network congestion time generated by nodes and links.

Ascigil et al. [12] proposed resource allocation algorithms to place service requests from users and reconfigure service resources in order to maximize the QoS experienced by users. This paper focuses on a uncoordinated service placement strategy whenever service requests arrive. Son and Buyya [13] suggested a latency-aware VNF provisioning scheme in distributed edge clouds. This paper places latency-sensitive services between edge and cloud to guarantee QoS. Keshavarznejad et al. [14] presented a task offloading mechanism in fog environments that blueuces the total power consumption as well as the delay in executing tasks. The authors solved the multi-objective optimization problem using the non-dominated sorting genetic algorithm and the Bees algorithm.

At present, reinforcement learning has proven to be a promising approach for the selection of an optimal policy. It has outperformed the traditional threshold-based approach which is not always very convenient. Duggan et al. [15] proposed an agent that learns the optimal time to schedule VM migration. An agent that can choose an optimal VM for migration from an overloaded host has been designed by Duggan et al. [16]. Peng et al. [17] proposed an online resource scheduling framework in cloud systems based on the Deep Q-Network (DQN) algorithm. The authors solved two optimization problems of energy consumption and service quality by adjusting the proportion of different reward optimization objectives. Alfakih et al. [18] proposed a resource management framework in mobile edge computing (MEC) environments using an online learning algorithm called state–action–reward–state–action (SARSA). They made an optimal offloading decision for minimizing energy consumption and computing delay based on the SARSA.

3 System models and problem definition

In this section, we discuss the system model for the proposed approach and define the problem to be solved in this paper.

3.1 System model

Figure 1 shows an example of the FEC environment that we are targeting in this paper. The FEC environment consists of multiple edge domains where each domain includes multiple edge servers with limited computing capacity. The edge servers within an edge domain are connected by an edge switch. Each edge switch is in turn connected to aggregate switches and core switches to route traffic to the cloud servers. We assume that the cloud servers are equipped with unlimited computing capacity. Both cloud servers and edge servers in an FEC environment are virtualized and multiple VMs can be created over these servers.

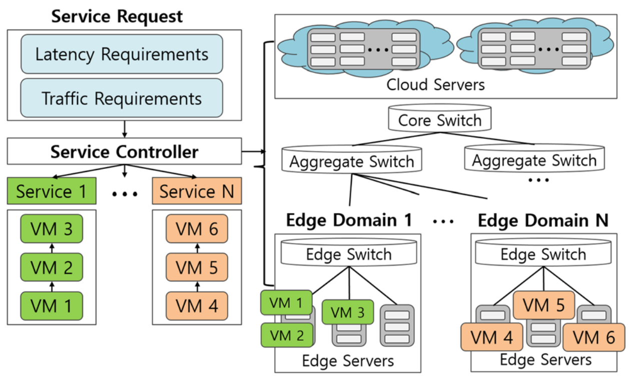

Fig. 1 System model

A service in an FEC environment consists of a series of service functions where each service function contains the descriptions of its minimum and maximum traffic requirements. When a service is requested, its service functions are placed in appropriate VMs to create a service path. We call this *service placement*. This paper assumes that only one service function can reside in one VM. The VM running a service function can be migrated to other edge servers or cloud servers to minimize the total energy consumption while ensuring service QoS. We call this *service reconfiguration*.

3.2 Problem definition

We assume that physical resources including edge servers and network links can be represented as a graph $G=(V,E)$. For example, V represents either edge servers or cloud servers where services can be placed, while E represents virtual network links between the servers.

If a service S is composed of K service functions SF, a service path SP can be expressed as $SP = SF_1 \rightarrow SF_2 \rightarrow \cdots \rightarrow SF_K$. Assume that the number of edge servers used for running all SP is N and the number of network links used to route the traffic from all SP is M. Then, the energy consumption of physical resources $E_{SP}^{physical}$ can be calculated by adding the sum of energy consumption from N edge servers and M links as shown in Eq. 1.

$$E_{SP}^{physical} = \sum_{i=1}^{N} E_i^{server} + \sum_{j=1}^{M} E_j^{link}. \tag{1}$$

After placing a service on the service path, the service can be migrated to nearby edge servers according to service traffic fluctuations. If the total number of migrations occurred in all SP is P, the total energy consumed for service migration $E_{SP}^{migration}$ is sum of energy consumed for each migration as shown in Eq. 2.

$$E_{SP}^{migration} = \sum_{i=1}^{P} E_{SP_i}^{migration}. \tag{2}$$

Then, the total energy consumption E_{SP}^{total} is the sum of $E_{SP}^{physical}$ and $E_{SP}^{migration}$ as shown in Eq. 3.

$$E_{SP}^{total} = E_{SP}^{physical} + E_{SP}^{migration}. \quad (3)$$

The main objective of the proposed approach is to minimize the total energy consumption E_{SP}^{total} such that the latency along the ith service path L_{SP_i} does not exceed the target service latency $L_{SP_i}^{target}$. If L_{SP_i} exceeds $L_{SP_i}^{target}$ in a service placement and service reconfiguration, *service QoS violation* is said to happen.

3.3 Energy model

The energy consumption in physical resources consists of the energy consumption of a server and a link between the servers. If a server or a switch is powered off, the energy consumption of this resource is considered to be 0. This paper assumes that the energy consumed by a server is the sum of static energy E_{server}^{static} (also called *idle* energy) and dynamic energy. It is reported in [19] that the dynamic energy of a server increases linearly with the CPU utilization used.

Therefore, the energy consumption of the ith server E_i^{server} used in this paper is defined as Eq. 4.

$$E_i^{server} = E_{server_i}^{static} + (E_{server_i}^{max} - E_{server_i}^{static}) \times \frac{CPU_i^{used}}{CPU_i^{total}}. \quad (4)$$

Similarly, a network switch consumes static energy E_{switch}^{static} regardless of the inflow of traffic, and dynamic energy depending on the number of active switch ports used for processing service traffic. In addition, the dynamic energy of a network switch is proportional to the number of active switch ports [20].

Assume that E_{switch}^{port} is defined as the energy consumption of each port, and num_{port} is the number of active switch ports. Then, the energy consumed by the ith link E_i^{link} is defined as Eq. 5.

$$E_i^{link} = E_{switch_i}^{static} + E_{switch_i}^{port} \times num_{port}. \quad (5)$$

The overall energy is also affected by the migration energy. This paper follows the migration energy model proposed by Liu et al. [21]. We assume that the energy for VM migration is affected by the duration of VM migration $L_{SP_i}^{migration}$ (defined in Eq. 10) and two regression parameters α, β which can be obtained in [21]. Then, the energy consumed by VM migration $E_{SP_i}^{migration}$ can be defined as Eq. 6, where VM_{size}^j represents the size of a VM j in the ith service path and BW_{link}^{avail} represents the available bandwidth of a link used for the migration.

$$E_{SP_i}^{migration} = \alpha \times \frac{VM_{size}^j}{BW_{link}^{avail}} + \beta. \quad (6)$$

3.4 Latency model

The service latency on the ith service path L_{SP_i} is the sum of the VM processing time in physical servers $L_{SP_i}^{server}$, the VM transmission time between physical servers $L_{SP_i}^{trans}$, and the VM migration time $L_{SP_i}^{migration}$ when service reconfiguration is invoked as shown in Eq. 7.

$$L_{SP_i} = L_{SP_i}^{server} + L_{SP_i}^{trans} + L_{SP_i}^{migration}. \quad (7)$$

$L_{SP_i}^{server}$ is calculated by the processing time of each service function multiplied by the actual CPU utilization of servers along the service path as shown in Eq. 8. We assume that the processing time of each service function $T_{processing}^{ideal}$ is obtained when only one vCPU is mapped to one pCPU in a server. Note that $T_{processing}^{ideal}$ can be increased as the number of vCPUs handled by one pCPU is increased.

$$L_{SP_i}^{server} = \sum_{j=1}^{N} \left(T_{processing}^{ideal} \times \frac{CPU_j^{used}}{CPU_j^{total}} \right). \quad (8)$$

$L_{SP_i}^{trans}$ is also defined by the sum of latency values from all switches along the ith service path as shown in Eq. 9. The latency in the jth switch is the average packet size generated in the ith service path $Pkt_{SP_i}^{size}$ divided by the available bandwidth of a link for transmitting a packet.

$$L_{SP_i}^{trans} = \sum_{j=1}^{M} \frac{Pkt_{SP_i}^{size}}{BW_{link}^{avail}}. \quad (9)$$

Besides, the VM migration time $L_{SP_i}^{migration}$ is affected by the size of a migrating VM because a target VM image is transferred to a destination server through network switches. Therefore, $L_{SP_i}^{migration}$ is calculated by the sum of the size of VM j in the ith service path divided by the available bandwidth of a link for the migration BW_{link}^{avail} as shown in Eq. 10.

$$L_{SP_i}^{migration} = \sum_{i=1}^{M} \frac{VM_{size}^j}{BW_{link}^{avail}}. \quad (10)$$

4 ESFEC: energy-efficient service scheduling in federated edge cloud

ESFEC consists of two sub-algorithms: *service placement* and *service reconfiguration*. This section explains the overall architecture of ESFEC and presents its algorithms in detail.

4.1 System overview

Figure 2 depicts the overall system architecture and operational flow of ESFEC, which consists of three main components: *service placement manager*, *service monitor*, *migration manager*.

The service placement manager is initiated when a new service request arrives at the service controller. That is, when a user initiates a service request to the service controller, the controller invokes the service placement manager to decide where to place the service in an edge domain. Since ESFEC considers actual resource utilization when placing services, the service placement manager initially allocates VMs running a series of service functions to appropriate edge servers based on actual traffic requirements.

The service monitor periodically checks the CPU utilization of VMs in each edge server every 30 s and updates the corresponding status in a database. If the aggregated CPU utilization of VMs in an edge server exceeds the predefined threshold value (we use 70% in this paper), the service monitor triggers the migration manager to reconfigure the service.

ESFEC provides three reconfiguration algorithms: one learning-based algorithm (ESFEC-RL) and two heuristic algorithms (ESFEC-EF and ESFEC-MF). Users can specify their preference at run time. In particular, when ESFEC-RL is chosen, it refers to the Q-table generated by the service agent. This allows us to find an optimal destination host that minimizes the total energy consumption and ensures service QoS during service reconfiguration.

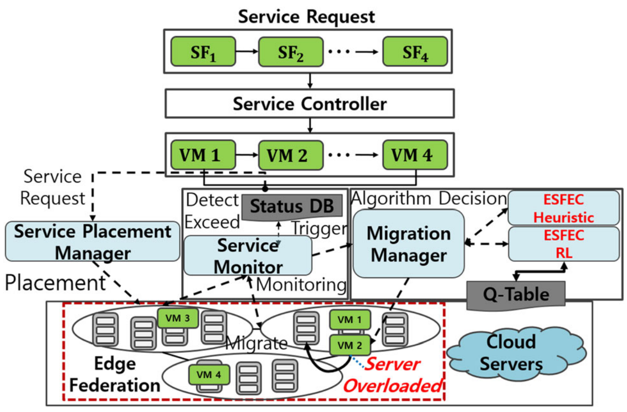

Fig. 2 Overview of ESFEC

4.2 Service placement

In the service placement algorithm, the main idea is to maximize the level of VM consolidation for low energy consumption. For this, the service placement algorithm tries to find an edge server *HOST* after placing a service *S* with multiple service functions $SP = SF_1 \rightarrow SF_2 \rightarrow \cdots \rightarrow SF_K$, where the number of *SF* is *K*. Therefore, we assume that each service request has both minimum and maximum traffic requirements as well as its latency requirement L_{SP} when a new service request arrives at the service controller. In order to minimize the used physical resources, the service controller places services from the low utilized edge servers so that the edge servers are not overloaded. Algorithm 1 shows the service placement algorithm in ESFEC.

Algorithm 1 Service Placement

```
1: $SF_i^{max}$, $SF_i^{min}$ : Max/Min traffic requirements of $SF_i$
2: $N$ : Number of edge servers
3: $HOST_N$ : List of edge servers
4: $HOST_i^{util}$ : Server utilization of $HOST_i$
5: $MaxUtil$ : Maximum traffic threshold
6:
7: procedure SERVICE PLACEMENT
8:     while $SP_{S_U}$ exists in the service request queue do
9:         $SF_i \leftarrow SP_{S_U}$
10:        Sort $HOST_N$ by utilization in ascending order
11:        $i \leftarrow 1$
12:        while $i \leq N$ do
13:            if $SF_i^{min} + HOST_i^{util} \leq MaxUtil$ then
14:                Place $SF_i$ on $HOST_i$
15:            else
16:                $i \leftarrow i + 1$
17:            end if
18:        end while
19:    end while
20: end procedure
```

4.3 RL-based service reconfiguration

ESFEC-RL is a *Q-learning*-based service reconfiguration algorithm that introduces the concept of an agent. The agent takes actions in the provided environment to maximize energy efficiency and guarantee service QoS. The agent learns an optimal behavior with repeated trial and error interactions in an environment that is completely unknown to the agent. Initially, the agent starts by taking random actions. Actions are chosen based on the policy π that the agent is supposed to follow. The agent receives a reward depending on the action selected. The agent can only learn interactively with the help of this reward function.

The behavioral value function is calculated using Eq. 11, where s' represents the next state after current state s under action a and a' represents all possible actions. γ is the discount factor that determines the importance of future returns relative to current returns. α is the learning rate. The larger the value of α, the smaller the influence of historical training results. We use 0.08 for γ and 0.05 for α in this paper. r is the reward value which will be explained later in Sect. 4.3.3.

$$Q(s,a) = Q(s,a) + \alpha \times (r + \gamma MinQ(s',a') - Q(s,a)). \tag{11}$$

The entire learning process of ESFEC-RL depends on the following three major components: *state*, *action*, and *reward function*. In what follows, we describe them in detail.

4.3.1 State space

The state space in ESFEC-RL is defined as a set $S = \{s_i \in S \mid 1 \leq i \leq k\}$, where s_i represents the ID i of an overloaded host $HOST_i$ and k is the total number of physical hosts in the FEC. ESFEC-RL also maintains a status table for all physical hosts that stores the CPU utilization and the size of each VM running on each physical host. Therefore, if $HOST_i$ is overloaded, a VM with the smallest size is chosen as a migrating VM and the resulting CPU utilization in the ith place of the status table is updated reflecting the migration.

4.3.2 Action space

Agents need to take a set of actions to modify their state in the environment. We define a set of actions that the agent can perform. There are two ways for selecting an action from a set of possible actions in every state.

- *Exploration or random action selection* in the earliest stages of our algorithm, an agent deals with an unknown environment. Thus, the action taken by the agent is random and optimal actions are not chosen yet.
- *Exploitation* an experienced agent takes an action based on the learning policy (π). The learning policy defines the reward which is discussed later in this paper.

In this paper, we use an ϵ-*greedy* policy [22] to make sure that an agent performs the best in our environment.

In ESFEC-RL, an action space A is defined as a set $A = \{a_i \in S \mid 1 \leq i \leq k\}$, where a_i is the ID i of a destination host $HOST_i$ for migration and k is the total number of physical hosts in the FEC. For example, an action a_3 means that $HOST_3$ is selected as a destination host and we migrate a VM to $HOST_3$. The CPU utilization after the migration is also updated in the status table. In this case, the maximum number of actions to be chosen by an agent is equivalent to the total number of physical hosts in the FEC.

With the increased number of physical hosts and the number of iterations, the dimension of state–action space grows exponentially. This can make a negative impact on getting an optimal result and may also affect the convergence speed. To cope with this, ESFEC-RL uses a candidate host list for deciding an action using only the hosts that meet the QoS requirements. Since the size of candidate host list is smaller than the total number of physical hosts, this approach allows us to greatly reduce the dimension complexity and simplify the energy calculation.

4.3.3 Reward function

A Q-table is maintained so that a learning agent can take a look at a Q-value with each action taken by the agent and work toward reaching the optimal action. For this, after every action taken by the learning agent, a reward value is calculated and the corresponding Q-value obtained by the behavior value function in Eq. 11 is updated in the Q-table. While updating the Q-values in the Q-table, ESFEC-RL learns in a direction in which energy on the service path is minimized in consideration of migration overhead.

Depending on which host the agent migrates a VM to, the total energy consumption along the service path E_{SP}^{total} becomes higher or lower than that in current state. To decide whether the agent's action is beneficial, we define a reward function r_t at time t as shown in Eq. 12, where we compare the total energy consumption along the service path at time t with that at $t+1$. If the reward value is negative, the action is considered to be good because the total energy consumption is reduced. Since our goal is to minimize the total energy consumption along the service path, the smaller the reward value the better the action taken by the agent.

$$r_t = \frac{E_{SP}^{total_{t+1}} - E_{SP}^{total_t}}{E_{SP}^{total_t}}. \tag{12}$$

4.3.4 Learning agent

To make an intelligent decision during service reconfiguration when overloaded hosts are detected, a learning agent has to be trained to minimize the total energy consumption while considering a migration overhead. The learning agent first perceives the current state to include overloaded host servers in the provided environment. Then it filters out the hosts that do not meet service QoS requirements and creates a candidate host list to determine a destination host for the migration. This filtering process not only helps us reduce the overall state–action space dimension but also confirms that all hosts on the candidate host list meet the QoS requirement.

Our learning agent chooses the optimal destination host from the candidate host list by referring to the Q-table [23]. However, since there is not enough information in the

Q-table in the initial learning stage, the learning agent randomly chooses the target destination host. The agent calculates a reward value based on this migration action and updates a new Q-value in the Q-table. After training the learning agent for appropriate episodes, the agent is able to make a better decision in terms of choosing an optimal destination host with Q-table. An optimal decision made by an agent is rewarded with a smaller value when the total energy consumption is minimized. Algorithm 2 describes the pseudocode of training a learning agent in detail.

Algorithm 2 RL-based Learning Agent

```
    Input Overloaded Host Servers
    Output Updated Q-table
 1: $\alpha$ : Learning rate where $\alpha \in [0,1]$, $\alpha = 0.05$
 2: $\gamma$ : Discount factor where $\gamma \in [0,1]$, $\gamma = 0.08$
 3: $s_t \leftarrow$ Overloaded host server at time t
 4: $a_t \leftarrow$ Destination host server for migration at time t
 5: $r_t$ : Reward value taken by $a_t$
 6: $Host_{dest}$ : List of candidate edge servers
 7: $Host_{status}$ : Host status table
 8: $Num_{episodes}$ : The number of episodes

 9: procedure RL-BASED LEARNING AGENT
10:     while $Num_{episodes}$ is terminated do
11:         Create $Host_{dest}$
12:         $s_t \leftarrow$ Overloaded host servers
13:         Take action $a_t$ with the smallest Q-value
14:         in Q-table
15:         Update $Host_{status}$ after migration
16:         $r_t = \frac{E_{SP}^{total_{t+1}} - E_{SP}^{total_t}}{E_{SP}^{total_t}}$
17:         $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma MinQ(s_{t+1}, a_{t+1})$
18:         $- Q(s_t, a_t))$
19:         Update Q-table
20:     end while
21: end procedure
```

An example scenario of training a learning agent with Q-table is shown with Figs. 3 and 4. Assume that the CPU utilization of VM_3 in $HOST_1$ increases from 10 to 20%, which causes the aggregated CPU utilization of $HOST_1$ to exceed the maximum CPU utilization per server (70% in this paper). Then, service reconfiguration is needed to prevent service QoS violation.

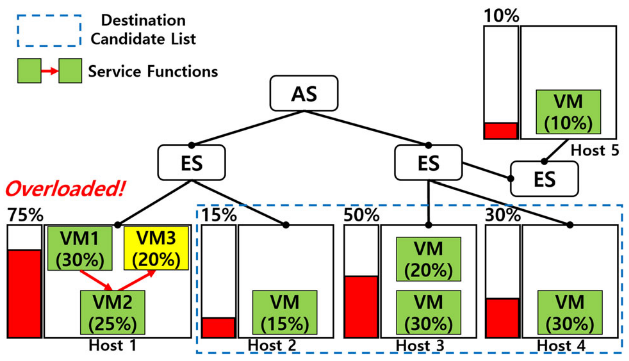

Fig. 3 Scenario of service reconfiguration in ESFEC-RL

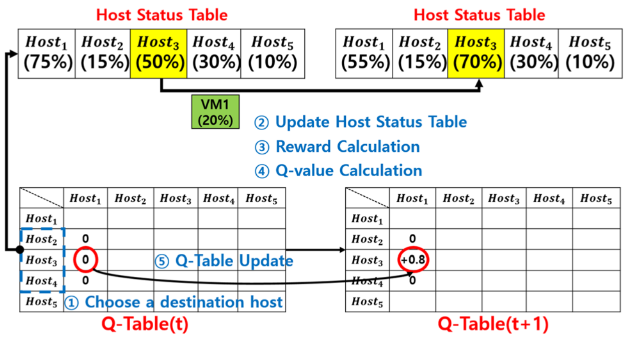
Host Status Table
Host1 (75%) Host2 (15%) Host3 (50%) Host4 (30%) Host5 (10%)
Host Status Table
Host1 (55%) Host2 (15%) Host3 (70%) Host4 (30%) Host5 (10%)
VM1 (20%)
② Update Host Status Table
③ Reward Calculation
④ Q-value Calculation
⑤ Q-Table Update
① Choose a destination host
Q-Table(t)
Q-Table(t+1)
0
0
0
+0.8

Fig. 4 Example of Q-table construction

As shown in Fig. 3, before starting the service migration, the learning agent makes a candidate host list by calculating whether the service latency along the service path exceeds the target latency after VM_3 is migrated to the destination hosts. Note that VM_3 is a VM with the smallest file size in $HOST_1$, which reduces the migration energy. If $HOST_2$, $HOST_3$, and $HOST_4$ are chosen as candidate hosts, the learning agent randomly chooses a destination host to migrate VM_3 since there is no prior knowledge. Suppose that the learning agent chooses $HOST_3$ as a destination host to migrate VM_3. After the migration, the learning agent updates the host status table (i.e., 75% $\rightarrow$ 55% of $HOST_1$, 50% $\rightarrow$ 70% of $HOST_3$) and calculates a reward value based on the action taken by migrating VM_3 to $HOST_3$. Finally, the Q-value calculated by the reward value is updated in the Q-table. After repeatedly performing the learning episodes, the agent is supposed to select an optimal destination host by referring to the Q-table.

4.4 Heuristic-based service reconfiguration

Although ESFEC-RL is likely to go toward the optimal goal, the execution time to converge to the minimum energy consumption can take longer. For this reason, the learning-based approach may not be suitable for an environment with a large number of states and actions. Therefore, ESFEC provides two heuristic-based reconfiguration algorithms: ESFEC-EF and ESFEC-MF. While ESFEC-EF focuses on minimizing the total energy consumption in reconstructing the service path, ESFEC-MF is targeting at minimizing the number of migrations.

Algorithm 3 Heuristic-based Service Reconfiguration

```
1:  $N$ : Number of edge servers
2:  $QOS^{num}_{violation}$ : Number of QoS violation
3:  $HOST_N$ : List of edge servers
4:  $HOST_{over}$ : List of overloaded edge servers
5:  $HOST_{dest}$ : List of candidate edge servers for migration
6:  $VM_j$ : VM with the smallest $VM^j_{size}$ in $HOST_{over}$
7:  $Energy_{path}$ : Total energy consumption along the path
8:  $Energy_{min}$ : Min energy consumption along the path

9:  procedure Heuristic-based Reconfiguration
10:     Sort $HOST_{over}$ by utilization in descending order
11:     while $HOST_{over}$ not empty do
12:         while $HOST_{dest}$ not empty do
13:             if ESFEC-EF is selected then
14:                 Call Energy-First($HOST_{dest}$, $VM_j$)
15:             else if ESFEC-MF is selected then
16:                 Call Migration-First($HOST_{dest}$, $VM_j$)
17:             end if
18:         end while
19:         $HOST_{over} = \{HOST_{over}\}$ - $\{HOST_i\}$
20:     end while
21: end procedure

22: procedure Energy-First($HOST_{dest}$, $VM_j$)
23:     $HOST_k \leftarrow$ Select 1st $HOST$ in $HOST_{dest}$
24:     Calculate energy consumption before the migration
25:     $Energy_{path} = E^{physical}_{SP} + E^{migration}_{SP}$
26:     if $L_{SP_k} > L^{target}_{SP_k}$ then
27:         $QOS^{num}_{violation} = QOS^{num}_{violation} + 1$
28:         $HOST_{dest} = \{HOST_{dest}\}$ - $\{HOST_k\}$
29:     else
30:         if $Energy_{path} < Energy_{min}$ then
31:             $Energy_{min} = Energy_{path}$
32:             Migrate $VM_j$ to $HOST_k$
33:         end if
34:     end if
35: end procedure

36: procedure Migration-First($HOST_{dest}$, $VM_j$)
37:     Sort $HOST^{utilization}_{dest}$ in ascending order
38:     $HOST_k \leftarrow$ Select $HOST$ sequentially
39:     if $L_{SP_k} > L^{target}_{SP_k}$ then
40:         $QOS^{num}_{violation} = QOS^{num}_{violation} + 1$
41:         $HOST_{dest} = \{HOST_{dest}\}$ - $\{HOST_k\}$
42:     else
43:         Migrate $VM_j$ to $HOST_k$
44:     end if
45: end procedure
```

The heuristic-based reconfiguration algorithm is described in Algorithm 3. When a service reconfiguration is required, a list of overloaded edge servers $HOST_{over}$ is sent to this algorithm by the *service monitor*. Then, this algorithm locates a VM with the smallest size VM_{size} from the *HOST* with the maximum server utilization in $HOST_{over}$. This is because a *HOST* with the maximum server utilization must be the most overloaded *HOST* and a VM with the smallest size has the minimum migration overhead, as shown in Eqs. 6 and 10. Finally, among the edge servers excluding the servers in $HOST_{over}$, this algorithm creates a list of candidate edge servers $HOST_{dest}$ to determine an appropriate destination edge server for service migration.

If ESFEC-EF is selected as a reconfiguration policy, a destination edge server is chosen so that the total energy consumption along the service path is minimized. For this, this algorithm searches for a *HOST* from the edge servers in $HOST_{dest}$ where the energy consumption after the service migration is minimized, while the service latency L_{SP} is below the target latency L^{target}_{SP}. In contrast, ESFEC-MF reconfigures a service path to minimize the possibility of service migration. That is, ESFEC-MF selects a VM with the smallest size VM_{size} and reallocates it to an edge server with the largest leftover CPU utilization. This reduces the possibility of service migration in next monitoring interval, while satisfying the latency L_{SP} requirement.

When ESFEC-EF and ESFEC-MF search for a destination edge server to migrate a VM, they initially try to find an edge server that belongs to the same edge domain. However, if there is no capacity available to migrate a VM in current edge domain, they move to the rest of edge domains that are connected by network switches. In this case, the latency requirement should also be ensured.

5 Evaluation

To show the effectiveness of ESFEC, we implemented the proposed algorithms over a cloud simulator called `CloudSimSDN` [24]. This section compares their performance with that of a traditional service scheduling algorithm proposed for the edge computing environment.

5.1 Experimental environment

5.1.1 Topology

For the simulation, we assume a FEC environment with 8 edge domains, where there are 100 edge servers in each domain (total 800 edge servers). Each edge server is equipped with one 16-core CPU and 32 GB RAM. The ratio of pCPU to vCPU is 1 (i.e., no sharing), which means that the maximum number of VMs per each edge server is 16. The edge servers in each domain are interconnected by a edge switch with a 1 Gbps link. Each edge switch is in turn connected to the four aggregate switches with a 10 Gbps link. Finally, each aggregate switch is connected to the two core switches with a 128 Gbps link to reach the two cloud servers. We assume that the cloud servers have unlimited computing capacity.

5.1.2 Energy parameters

For the energy parameters for a server and a switch such as peak power consumption and idle power consumption, we used the parameters suggested in `CloudSimSDN`.

5.1.3 Workloads and comparison target

We assume that two real-time application services with different traffic characteristics, *face recognition* and *online text translator*, run at the same time with a ratio of 60% to 40% for the simulation. While the face recognition service is CPU intensive, the online text translator service is I/O intensive. Each application service consists of 3 different service functions and a total of 3000 service requests is generated for the simulation (i.e., 1800 face recognition services and 1200 online text translator services). We referred to [26] to decide the parameters used for latency model such as service packet size and latency between network switches. The detailed specifications of application services and service functions are summarized in Tables 1 and 2.

In order to emulate the traffic ingested into each application service, we used a real traffic log from PlanetLab [27]. This real dataset has trace logs of CPU utilization for two months (i.e., from March to April in 2011) from more than a thousand VMs in 500 distributed physical servers around the world. In this dataset, the CPU utilization of each VM in 1 day was measured in every 5 min and recorded into a separate file. Therefore, each file includes the 288 lines of a VM's CPU utilization data for 24 h. As a result, the same number of unique files as the number of VMs running on a certain day is generated in the same day.

As we discussed in Sect. 4, ESFEC places services with their minimum CPU utilization rather than their maximum utilization. To simulate this environment with different traffic characteristics, we collected the PlanetLab's trace log files on April 3, 2011 that include the CPU utilization data of 1463 VMs (i.e., 1463 files). By analyzing those files, we classified them as *low*, *medium* and *high* traffic types. Figure 5 shows the CPU utilization of three VMs randomly chosen and averaged from each traffic type. Also, to determine the minimum and maximum utilization of each application service, we chose 20 files each from 3 traffic types and used the averages of their minimum and maximum CPU utilization for the simulation. The detailed

Table 2 Specification of service functions [25]

Function type	# of CPU cores	VM size
SuperHub	2	512 MB
Livebox	1	256 MB
TPLink	2	128 MB
Netgear	1	384 MB
Face Recognition API	1	1 GB
Text Translator API	1	1 GB

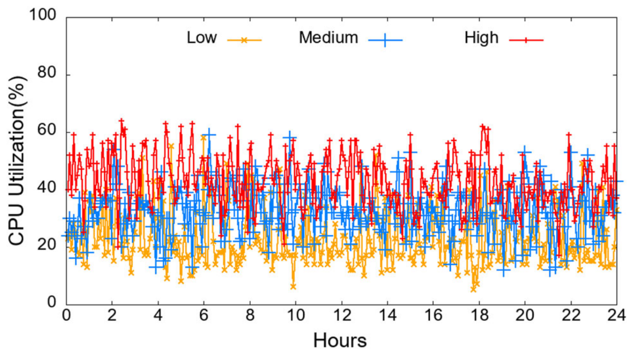

Fig. 5 CPU utilization distribution per traffic type

CPU utilization requirements of each application service are shown in Table 3. We assume that each application service has the same CPU utilization per traffic type in the simulation.

For comparison, we also implemented one heuristic service scheduling algorithm called EC-MAX, which is used in the traditional edge cloud environment. EC-MAX places service functions in a service path with their maximum traffic requirements on the edge servers and reconfigures the service path without considering migration overhead. In contrast to ESFEC, this algorithm relocates services to cloud servers if sufficient computing capacity cannot be provided in the edge servers. In what follows, we compare the performance of ESFEC and its related algorithms with that of EC-MAX in terms of energy consumption and QoS violation rate.

Table 1 Specification of application services [25]

Type	Functions	Latency (s)	Traffic
Face Recognition (FR)	SuperHub → TPLink → FR API	2.0	CPU intensive
Text Translator (TT)	Livebox → Netgear → TT API	1.5	I/O intensive

Table 3 CPU utilization requirements

Application	Traffic	Minimum (%)	Maximum (%)
Face Recognition	Low	30	48
	Medium	36	54
	High	40	60
Text Translator	Low	15	32
	Medium	20	38
	High	24	44

5.2 Convergence analysis of ESFEC-RL

In this section, we analyze ESFEC-RL and check whether this algorithm converges to the optimal solution. Figure 6 shows the convergence patterns of ESFEC-RL for 60 iterations over the three different traffic types. As shown in Fig. 6, ESFEC-RL starts to converge around the 50th iteration regardless of traffic type. This indicates that there is no significant reduction in energy consumption after that point.

On the other hand, in a low traffic type shown in Fig. 6a, we can observe that the learning curve is quite smooth up to the 20th iteration. However, when the traffic is getting heavier (i.e., medium traffic), the learning curve starts to fluctuate during that period. In particular, in the high traffic type shown in Fig. 6c, ESFEC-RL sharply fluctuates without converging to its optimal solution and finally starts to converge around from the 25th iteration.

This can be explained by the following observations. When the traffic is low, the traffic volume of most destination hosts in the candidate host list is also likely to be very low. Thus, after a migration is finished, the difference of energy consumption by a random decision and an optimal decision is minimal, which creates a smooth learning curve. Meanwhile, when the traffic volume gets larger, the difference of resulting energy consumption by both decisions widen because the probability of having much traffic on destination hosts is increased due to the increased number of VM migrations and the agent's random decision. It should be noted that as ESFEC-RL has more iterations, it finally converges to the optimal solution around at the 50th iteration.

5.3 Comparison of energy consumption

Figure 7 shows the energy consumption of heuristic-based ESFEC algorithms and EC-MAX normalized with respect to that of ESFEC-RL using three different traffic types. Overall, ESFEC-RL shows the lowest energy consumption, while EC-MAX has the highest energy consumption over all traffic types. Since ESFEC places services based on their minimum CPU utilization, the number of co-located VMs in a single edge server increases. This leads to the reduction of energy consumption.

On the other hand, as the traffic gets heavier, the performance gap in energy consumption is reduced. For example, in the low traffic condition, EC-MAX consumed 28%, 10% and 12% more energy than ESFEC-RL, ESFEC-EF and ESFEC-MF, respectively. Whereas, the gap is slightly reduced to 16%, 10% and 8% in the high traffic condition.

This is because low traffic causes less service migration and the migration energy does not make a significant impact on the overall energy consumption. However, when the traffic goes higher, frequent service migrations among overloaded edge servers result in high migration energy along the service path. It is worthy to note that the reconfiguration algorithms in ESFEC reduce migration overhead by choosing a migrating VM with the smallest size. Moreover, choosing a host with the largest leftover capacity also reduces the probability of the migrating VM to migrate again in the next monitoring interval. Although the number of migrations in ESFEC is expected to be more than that of EC-MAX, the mechanisms mentioned above can reduce the energy consumption even in the high traffic condition.

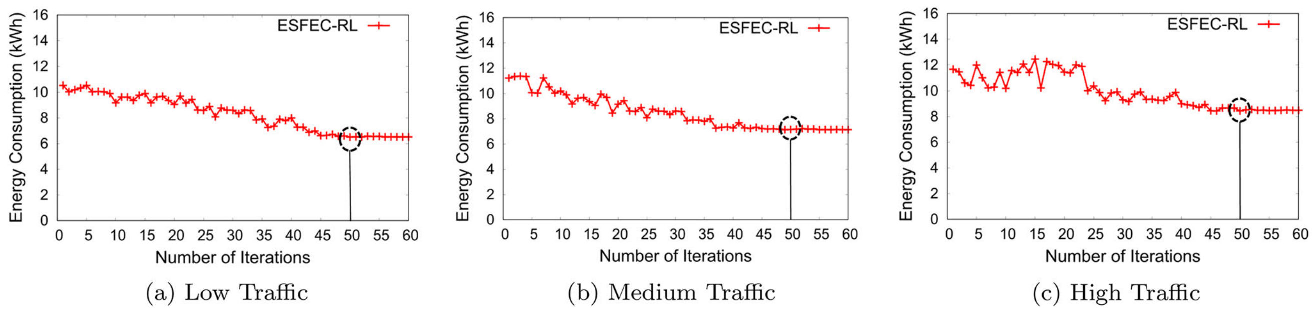

Fig. 6 Convergence patterns in ESFEC-RL

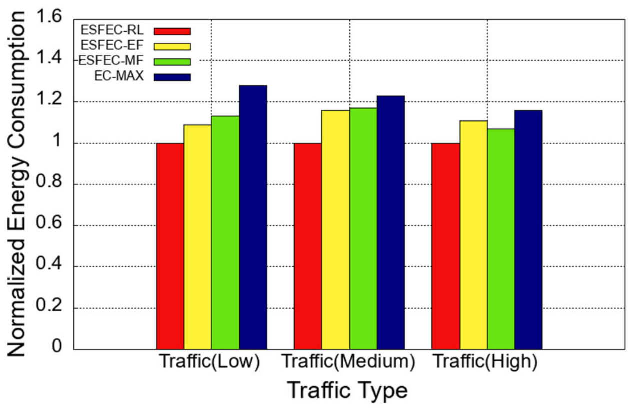

Fig. 7 Comparison of energy consumption per service

5.4 Comparison of QoS violation rate

Figure 8 compares the QoS violation rate of ESFEC-RL with those of heuristic-based ESFEC algorithms and EC-MAX. As shown in Fig. 8, all algorithms developed in ESFEC have lower violation rates than that of EC-MAX. While the QoS violation rates of three ESFEC algorithms are almost consistent regardless of traffic type, the QoS violation rate of EC-MAX keeps increasing as we increase the traffic volume. For example, in the high traffic condition, EC-MAX shows a violation rate that is almost 66% higher than ESFEC-RL.

Note that all ESFEC algorithms continuously find nearby edge servers when they fail to place services on one edge server. In contrast, EC-MAX starts to place services on cloud servers if sufficient capacity is not available in the edge server. This causes a high QoS violation rate because the service traffic traverses through multiple network switches, which results in high network latency.

5.5 Migration overhead analysis

We showed in Sect. 5.3 that all ESFEC algorithms consume less energy than EC-MAX even when the number of service migrations increases in a high traffic condition. To understand why this happens, this section analyzes the number of service migrations in different traffic conditions and its relationship with migration energy consumption.

Figures 9 and 10 show the number of service migrations and migration energy consumption of ESFEC-EF, ESFEC-MF and EC-MAX normalized with respect to those of ESFEC-RL, respectively. As shown in Fig. 9, the number of migrations in EC-MAX is less than those of ESFEC algorithms in all traffic types. Strangely, the migration energy in EC-MAX is relatively larger than those of ESFEC algorithms, as shown in Fig. 10. For example, ESFEC-RL generates 8% and 12% more migrations than EC-MAX in medium and high traffic conditions, respectively. However, EC-MAX consumes 33% more energy than ESFEC-RL in the medium traffic type, and its migration energy consumption increases up to 50% more than ESFEC-RL in a high traffic type.

The main reason for this is as follows. Note that EC-MAX does not take into account migration overhead when service reconfiguration is conducted, which incurs more migration energy. However, the service reconfiguration algorithms in ESFEC reconstruct a service path while minimizing migration overhead. In fact, we can observe in Fig. 10 that the migration energy in all ESFEC algorithms is smaller than that of EC-MAX in the high traffic condition in spite of the increased number of migrations. Another reason is that ESFEC reduces the number of network switches along the service path because it migrates VMs from an overloaded edge server to nearby edge servers connected by edge switches. In contrast, EC-MAX migrates VMs to cloud servers if not enough capacity is provided in an edge server, which increases the number of

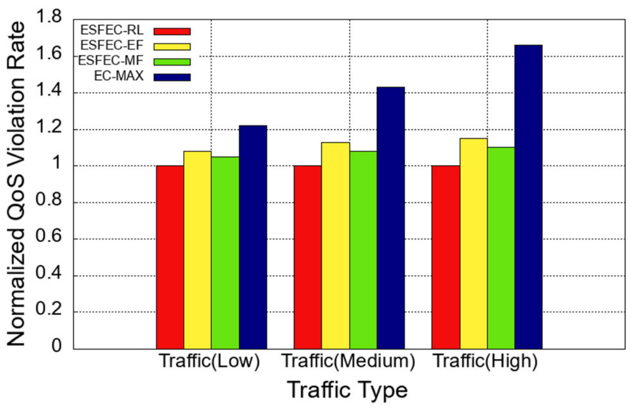

Fig. 8 Comparison of QoS violation rate per service

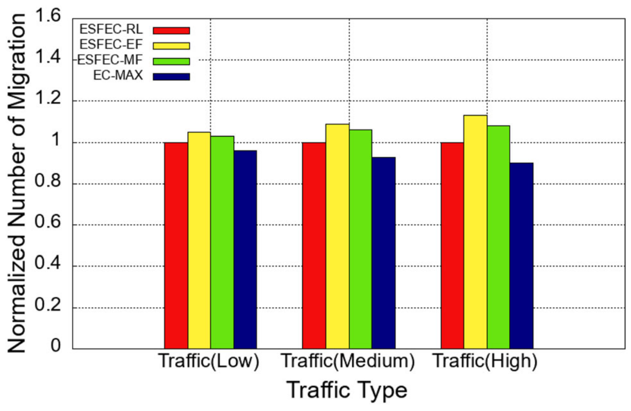

Fig. 9 Number of migrations

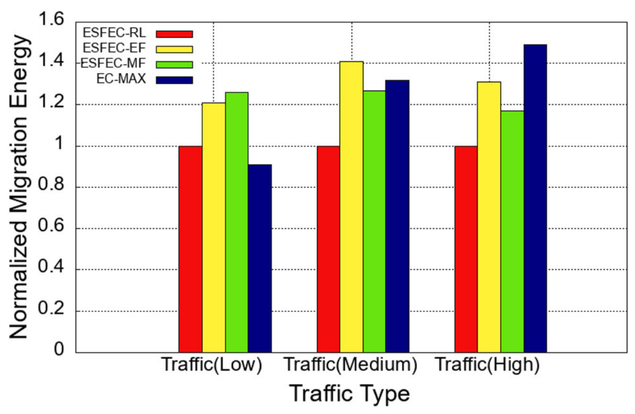

Fig. 10 Migration energy

network switches (i.e., edge/aggregation/core switches) along the service path.

It is also shown in Figs. 9 and 10 that ESFEC-EF incurs slightly more service migrations and consumes more migration energy than ESFEC-MF, especially in medium to high traffic conditions. This is because ESFEC-EF focuses more on minimizing the energy consumption along the service path rather than minimizing the number of migrations when a migration is initiated. However, the energy consumption and QoS violation rate of both algorithms are very similar to each other in all traffic conditions, as shown in Figs. 7 and 8. This indicates that both algorithms have similar energy efficiency regardless of migration overhead.

6 Conclusion

In this paper, we have proposed an energy efficient service scheduling algorithm in a FEC environment called ESFEC, which consists of a service placement algorithm and three service reconfiguration algorithms: ESFEC-RL, ESFEC-EF, and ESFEC-MF.

The main idea behind the ESFEC's design is to place services on edge servers in nearby edge domains and use the minimum traffic requirements instead of their maximum requirements. This approach reduces the QoS violation rate of a given service and increases the level of VM consolidation in a singe edge server, which reduces energy consumption along the service path. Moreover, the service reconfiguration algorithms in ESFEC are designed so that they reduce migration overhead by selecting a VM with the smallest size as a migrating VM and a host with the largest leftover CPU utilization as a destination host for migration. Through simulations, we have shown that the proposed algorithms are effective for minimizing energy consumption as well as QoS violation rate. The simulation results show that ESFEC improves energy efficiency by up to 28% and reduces the service violation rate by up to 66% more than the existing service scheduling mechanism used in edge clouds.

Acknowledgements This research was supported by Next-Generation Information Computing Development Program through National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT 2017M3C4A7080245.

References

1. Jeong, Y., Maria, K.E., Park, S.: An energy-efficient service scheduling algorithm in federated edge cloud. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), pp. 48–53 (2020)
2. Cao, X., Tang, G., Guo, D., Li, Y., Zhang, W.: Edge Federation: Towards an Integrated Service Provisioning Model. arXiv preprint (2019). arXiv:1902.09055
3. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
4. Ganesh, L., Weatherspoon, H., Marian, T., Birman, K.: Integrated approach to data center power management. IEEE Trans. Comput. **62**(6), 1086–1096 (2013)
5. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing, p. 7 (2012)
6. Eramo, V., Ammar, M., Lavacca, F.G.: Migration energy aware reconfigurations of virtual network function instances in NFV architectures. IEEE Access **5**, 4927–4938 (2017)
7. Kim, S., Park, S., Youngjae, K., Kim, S., Lee, K.: VNF-EQ: dynamic placement of virtual network functions for energy efficiency and QoS guarantee in NFV. Clust. Comput. **20**, 09 (2017)
8. Abdessamia, F., Tian, Y.-C.: Energy-efficiency virtual machine placement based on binary gravitational search algorithm. Clust. Comput. **23**, 09 (2020)
9. Tarahomi, M., Izadi, M., Ghobaei-Arani, M.: An efficient power-aware VM allocation mechanism in cloud data centers: a micro genetic-based approach. Clust. Comput. **24**, 06 (2021)
10. Sun, G., Li, Y., Yu, H., Vasilakos, A.V., Du, X., Guizani, M.: Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. Future Gener. Comput. Syst. **91**, 347–360 (2019)
11. Shang, X., Liu, Z., Yang, Y.: Network congestion-aware online service function chain placement and load balancing. In: Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019. Association for Computing Machinery, New York (2019)
12. Ascigil, O., Phan, T.K., Tasiopoulos, A.G., Sourlas, V., Psaras, I., Pavlou, G.: On uncoordinated service placement in edge-clouds. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 41–48 (2017)
13. Son, J., Buyya, R.: Latency-aware virtualized network function provisioning for distributed edge clouds. J. Syst. Softw. **152**, 24–31 (2019)
14. Keshavarznejad, M., Rezvani, M., Adabi, S.: Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. Clust. Comput. (2021). https://doi.org/10.1007/s10586-020-03230-y
15. Duggan, M., Duggan, J., Howley, E., Barrett, E.: A network aware approach for the scheduling of virtual machine migration during peak loads. Clust. Comput. **20**, 1–12 (2017)

16. Duggan, M., Flesk, K., Duggan, J., Howley, E., Barrett, E.: A reinforcement learning approach for dynamic selection of virtual machines in cloud data centres. In: The Sixth International Conference on Innovative Computing Technology (2016)
17. Peng, Z., Lin, J., Cui, D., Li, Q., He, J.: A multi-objective trade-off framework for cloud resource scheduling based on the deep Q-network algorithm. Clust. Comput. **23**, 12 (2020)
18. Alfakih, T., Hassan, M.M., Gumaei, A., Savaglio, C., Fortino, G.: Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access **8**, 54074–54084 (2020)
19. Chen, Q., Grosso, P., van der Veldt, K., de Laat, C., Hofman, R., Bal, H.: Profiling energy consumption of VMs for green cloud computing. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, pp. 768–775 (2011)
20. Wang, X., Wang, X., Zheng, K., Yao, Y., Cao, Q.: Correlation-aware traffic consolidation for power optimization of data center networks. IEEE Trans. Parallel Distrib. Syst. **27**(4), 992–1006 (2016)
21. Liu, H., Xu, C.-Z., Jin, H., Liao, X.: Performance and energy modeling for live migration of virtual machines. Clust. Comput. **16**, 171–182 (2011)
22. Wunder, M., Littman, M., Babes, M.: Classes of multiagent Q-learning dynamics with ϵ-greedy exploration. In: 27th International Conference on Machine Learning (2010)
23. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992)
24. Son, J., Dastjerdi, A.V., Calheiros, R.N., Ji, X., Yoon, Y., Buyya, R.: CloudSimSDN: modeling and simulation of software-defined cloud data centers. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 475–484 (2015)
25. Cziva, R., Pezaros, D.P.: Container network functions: bringing NFV to the network edge. IEEE Commun. Mag. **55**(6), 24–31 (2017)
26. Antoniou, I., Ivanov, V., Ivanov, V., Zrelov, P.: On the log-normal distribution of network traffic. Physica D **167**, 72–85 (2002)
27. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. Pract. Exp. **24**, 1397–1420 (2012)

Yeonwoo Jeong Yeonwoo Jeong is currently pursuing Ph.D. Degree in Computer Science and Engineering from Sogang University, Seoul, Republic of Korea. He received his B.S. Degree in Computer Software from Kwangwoon University and M.S. Degree in Computer Science and Engineering in Sogang University. His research interests include cloud computing and resource management.

Esrat Maria Esrat Maria is a Master Degree Student in Sogang University, Seoul, Republic of Korea. She received her B.S. Degree in Computer Science from Ahsanullah University of Science and Technology. Her research interests include cloud computing and resource management.

Sungyong Park Sungyong Park is a Professor in the Department of Computer Science and Engineering at Sogang University, Seoul, Korea. He received his B.S. Degree in Computer Science from Sogang University, and both the M.S. and Ph.D. Degrees in Computer Science from Syracuse University. From 1987 to 1992, he worked for LG Electronics, Korea, as a Research Engineer. From 1998 to 1999, he was a Research Scientist at Telcordia Technologies (formerly Bellcore), where he developed network management software for optical switches. His research interests include cloud computing and systems, virtualization technologies, high performance I/O and storage systems, and embedded system software.