An Energy-Efficient Service Scheduling Algorithm in Federated Edge Cloud

Yeonwoo Jeong, Khan Esrat Maria and Sungyong Park
Department of Computer Science and Engineering
Sogang University, Seoul, Korea
{akssus12, esratmaria, parksy}@sogang.ac.kr

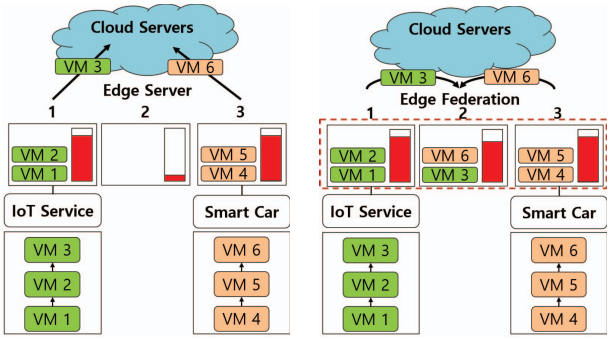
***Abstract*—Federated edge cloud (FEC) is an edge cloud environment where multiple edge servers in a single administrative domain collaborate together to provide real-time services. This environment reduces the possibility of violating the quality of service (QoS) requirements of target services by locating delay-sensitive services at nearby edge servers instead of deploying them on the cloud. However, as the number of edge servers increases, the amount of energy consumed by servers and network switches also increases. This creates another challenge for how to schedule delay-sensitive services over FEC, while minimizing the total energy consumption and reducing the QoS violation of a service at the same time. This paper proposes an energy-efficient service scheduling algorithm in FEC. The proposed algorithm is based on an observation that as the number of edge servers along the service path is reduced, the total energy consumption can be minimized. Traditional approaches place services using their maximum traffic requirements to ensure QoS without considering the actual traffic change. In contrast, the proposed algorithm schedules them with actual traffic requirements to increase the number of services co-located in a single server. This maximizes the consolidation of services in a single server and thus minimizes the energy consumption. Moreover, when edge servers are overloaded, the proposed algorithm reconfigures the service path such that service migration overhead and energy consumption are minimized while guaranteeing the QoS requirements of services. The simulation results show that the proposed algorithm improves energy efficiency by up to 21% and lowers the service violation rate by up to 80% against existing approaches.**

***Keywords*—edge computing, energy-efficient, service scheduling, federated edge**

I. Introduction

With the development of 5G mobile communication and Internet of Things (IoT), an edge cloud environment has emerged to smoothly provide latency-sensitive services such as augmented reality (AR) and autonomous vehicles. Instead of managing data only on a physically separated cloud server, edge cloud [1] performs key functions such as data collection, analysis, and processing on an edge server close to users. In other words, services that are sensitive to latency are first deployed on the edge server, and services that are not sensitive to latency or services that cannot be deployed on an edge server due to the limitation of computing resources are placed in the cloud. As a result, when a latency-sensitive service is deployed in the cloud, network latency increases, resulting in a QoS violation of the service.

* Corresponding author: Sungyong Park



(a) Edge Cloud (b) Federated Edge Cloud

Figure 1. Edge Cloud and Federated Edge Cloud

Meanwhile, as the number of edge servers managed in one administrative domain increases, a federated edge cloud (FEC) environment that provides services by the collaboration of multiple edge servers has appeared [2] as shown in Fig. 1. In this environment, services that are not deployed on one edge server can be deployed on nearby edge servers to ensure the service QoS. However, as the number of physical resources such as distributed edge servers and switches increases, energy consumption also increases [3]. Considering that the energy consumption and QoS depends on which server a service is deployed, it is necessary to devise an efficient service scheduling strategy that satisfies the QoS of a service while reducing energy consumption in FEC.

There have been few research activities conducted to address this issue in FEC. On the other hand, a number of research efforts for scheduling services to reduce energy consumption have been made either in multi clouds [4]–[8] or edge clouds [9], [10]. However, most of the previous studies have been 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 results in low resource utilization and unnecessary energy consumption. Furthermore, service migration scenario is not taken into account because services are scheduled based on their maximum traffic requirements and, therefore, cannot violate their QoS requirements by traffic fluctuations. This assumption is not suitable for our target environment since the computing capacity of each edge server in FEC is limited.

978-1-7281-8414-2/20/$31.00 ©2020 IEEE
DOI 10.1109/ACSOS-C51401.2020.00028

This paper proposes an energy-efficient service scheduling algorithm in FEC that minimizes energy consumption on the service path while ensuring QoS at the same time. The proposed algorithm initially places delay-sensitive services in edge servers based on their minimum CPU utilization requirements when service requests arrive. This allows us to increase the number of co-located services and thus reduce energy consumption. By periodically monitoring the traffic fluctuation and volume in each edge server, our algorithm reconfigures the service path when the CPU utilization of any edge server exceeds beyond the predefined target threshold value. All decisions are made by simultaneously considering the minimization of total energy consumption, the reduction of QoS violation possibility, and the minimization of service migration overhead. Since this problem is a variation of a multi-constrained optimization problem known as NP-complete [11], we propose a heuristic algorithm to solve such a scenario. The simulation has been conducted on a cloud simulator called `CloudsimSDN` [12]. The simulation results show that the proposed algorithm minimizes energy consumption by up to 21% compared to existing approaches and also reduces service violation rate by up to 80%.

The organization of this paper is as follows. Section II provides the descriptions of related works for minimizing energy consumption in multi clouds and edge clouds. Section III defines the problem and explains the details of our proposed algorithm. Section IV evaluates the performance of the proposed algorithm and shows its comparison with other existing works. Finally, Section V concludes this paper.

II. Related Works and Motivation

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 as well as the motivations for the proposed research.

Multi Clouds Kim et al. [4] proposes a dynamic virtual network function (VNF) placement and reconfiguration algorithm to minimize the energy consumption while ensuring service QoS. Nonde et al. [5] suggests an energy efficient virtual network embedding approach for cloud computing networks to consolidate resources in the network and data centers. Son et al. [6] proposes a dynamic overbooking algorithm that allocates host and network resources dynamically based on the resource utilization. Sun et al. [7] proposes an energy-efficient and traffic-aware service function chaining (SFC) orchestration in multi-domain networks. They propose an online service scheduling mechanism to minimize energy consumption in servers and network links. Shang et al. [8] proposes 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.

Edge Clouds Ascigil et al. [9] proposes 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 et al. [10] suggests a latency-aware VNF provisioning scheme in distributed edge clouds. This paper places latency-sensitive services between edge and cloud to guarantee QoS.

Motivation The average CPU utilization of a server cluster where services are placed is only about 50-60% [13]. This means that if services are allocated based on their maximum traffic requirements, a large amount of computing resources are wasted, resulting in unnecessary energy consumption. Most of the aforementioned prior works use maximum traffic requirements as a decision criterion in where to place services. Thus, it is highly likely that edge servers can be severely underutilized if service traffic is low.

It should also be noted that our target environment (i.e., FEC) is an federated environment that enables to share computing resources with nearby edge servers. In traditional edge cloud environment, delay-sensitive services are often placed on the cloud due to the capacity limitation of a edge server. However, FEC allows us to utilize the computing resources shared by other edge servers, which further reduces the possibility of violating the QoS requirements of a service.

Based on these observations, this paper proposes a service scheduling algorithm in FEC. The proposed algorithm places services with their minimum traffic requirements along the service path and gradually reconfigures the path as the amount of traffic increases. Since the service migration overhead is one of the biggest concerns, the service placement and reconfiguration algorithm schedules services to minimize the service migration.

III. Energy-Efficient Service Scheduling Algorithm

In this section, we discuss the system model for the proposed approach and define the problem followed by the detailed algorithm.

A. System Model

Fig. 2 shows an example of an FEC environment we are targeting in this paper. We assume that there exist multiple edge domains where multiple edge servers with limited computing capacity are located and share computing resources among them. The edge servers in each edge domain are connected by an edge switch. Each edge switch is connected by aggregate switches and core switches to route traffic to the cloud servers. The cloud servers are assumed to have unlimited computing capacity. Both cloud servers and edge servers are virtualized and multiple virtual machines (VMs) can be created over these servers.

When a service request with a series of service functions and its latency requirement arrives at the service controller, the controller creates a corresponding VM for each service

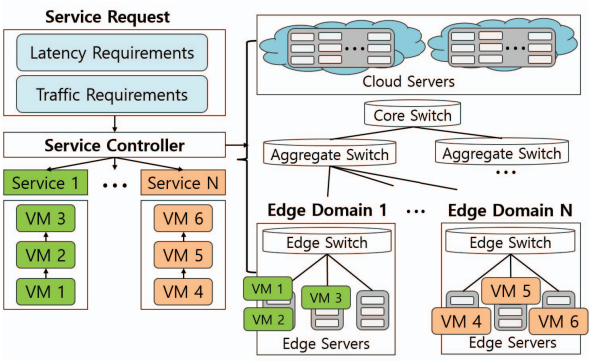


Figure 2. System Model

function and place them on appropriate edge servers. Each service function contains the descriptions of its minimum and maximum traffic requirements. This paper assumes that only one service function can reside at one VM.

B. 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 $SP = SF_1 \rightarrow SF_2 \rightarrow ... \rightarrow SF_K$. 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.

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

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 migration $E_{SP}^{migration}$ is sum of energy consumed for each migration as shown in Eq. 2.

$$E_{SP}^{migration} = \sum_{i=1}^{P} E_{SP_i}^{migration} \quad (2)$$

Then, the total energy consumption E_{SP}^{total} is the sum of $E_{SP}^{physical}$ and $E_{SP}^{migration}$ as shown in Eq. 3. The main objective of the proposed approach is to minimize the total energy consumption E_{SP}^{total} such that L_{SP_i} the latency along the i-th service path L_{SP_i} does not exceed the target service latency $L_{SP_i}^{target}$, where L_{SP_i} includes both the latency along the path and the latency incurred by service migration.

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

C. The Proposed Approach

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. The dynamic energy increases linearly with the CPU utilization used [14]. Therefore, the energy consumption of the i-th server E_i^{server} used in this paper is defined as Eq. 4.

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

The other energy consumption arises from network links. The energy consumed by links depends on the number of switch ports used for processing traffic [15]. Assume that E_{switch}^{static} and E_{switch}^{port} are defined as the energy consumption when the switch is idle and the energy consumption of each port, respectively. Assume also that num_{port} is the number of ports used. Then, the energy consumed by the i-th 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. [16]. Therefore, 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 [16]. 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 i-th service path and BW_{link}^{avail} represents the available bandwidth of a link used for the migration.

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

Latency Model The service latency on the i-th service path L_{SP_i} is the sum of the VM processing time in the servers $L_{SP_i}^{server}$, the VM transmission time between servers $L_{SP_i}^{trans}$, and the VM migration time $L_{SP_i}^{migration}$ as shown in Eq. 7.

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

$L_{SP_i}^{server}$ depends on server utilization. Thus, $L_{SP_i}^{server}$ is calculated by the sum of latency values from all edge servers along the i-th service path as shown in Eq. 8. The latency in the j-th server can be obtained by multiplying the idle processing time $T_{processing}^{idle}$ and the actual CPU utilization of the j-th server.

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

$L_{SP_i}^{trans}$ is also defined by the sum of latency values from all switches along the i-th service path as shown in Eq. 9. The latency in the j-th switch is the average packet size

generated in the i-th service path $Pkt^{size}_{SP_i}$ divided by the available bandwidth of a link for transmitting a packet.

$$L^{trans}_{SP_i} = \sum_{j=1}^{M} \frac{Pkt^{size}_{SP_i}}{BW^{avail}_{link}} \quad (9)$$

Besides, $L^{migration}_{SP_i}$ is calculated by the sum of the size of VM j in the i-th service path divided by the available bandwidth of a link for the migration BW^{avail}_{link} as shown in Eq. 10.

$$L^{migration}_{SP_i} = \sum_{i=1}^{M} \frac{VM^{j}_{size}}{BW^{avail}_{link}} \quad (10)$$

Algorithm The proposed algorithm consists of two sub-algorithms: *service placement* and *service path reconfiguration* as shown in Algorithm 1.

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 $SP = SF_1 \rightarrow SF_2 \rightarrow ... \rightarrow SF_K$, 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. Although this mechanism may cause service migration when the service traffic is relatively high, the service path reconfiguration algorithm is designed to minimize the service migration overhead.

On the other hand, the service path reconfiguration algorithm is triggered by a service monitor. When the service path reconfiguration algorithm is invoked, a list of overloaded edge servers $HOST_{over}$ is delivered 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 $HOST_{over}$. 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 Eq. 6 and Eq. 10. Finally, among the edge servers excluding the servers in $HOST_{over}$, we create a list of candidate edge servers $HOST_{dest}$ to determine a destination edge server for service migration. From the edge servers in $HOST_{dest}$, this algorithm searches for a $HOST$, where the energy consumption after service migration is minimized and the latency L_{SP} is below the target latency L^{target}_{SP}.

Algorithm 1 Service Placement and Reconfiguration

```
SF_i^max, SF_i^min : Max/min traffic requirements of SF_i
N : Number of edge servers
SP_i : i-th service path that includes HOST_i
HOST_N : List of edge servers
HOST_over : List of overloaded edge servers
HOST_dest : List of candidate edge servers for migration
HOST_i^util : Server utilization of HOST_i
MaxUtil : Maximum traffic threshold

procedure SERVICE PLACEMENT
    while SP_{S_U} exists in the service request queue do
        SF_i ← SP_{S_U}
        Sort HOST_N by utilization in ascending order
        i ← 1
        while i ≤ N do
            if SF_i^min + HOST_i^util ≤ MaxUtil then
                Place SF_i on HOST_i
            else
                i ← i + 1
            end if
        end while
    end while
end procedure

procedure SERVICE PATH RECONFIGURATION
    Sort HOST_over by utilization in descending order
    while HOST_over not empty do
        HOST_i ← the 1st HOST in HOST_over
        VM_j ← a VM with the the smallest VM_size^i
        HOST_dest = {HOST_N} - {HOST_over}
        while HOST_dest not empty do
            HOST_k ← HOST with minimum energy
            after migration
            if L_{SP_k} > L_{SP_k}^target then
                HOST_dest = {HOST_dest} - {HOST_k}
            else
                Migrate VM_j to HOST_k
            end if
        end while
        HOST_over = {HOST_over} - {HOST_i}
    end while
end procedure
```

Implementation Architecture Fig. 3 depicts the overall architecture and operational flow of our proposed algorithm. The proposed algorithm consists of three components: *service placement manager*, *migration manager* and *service monitor*.

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 the proposed algorithm 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 the minimum traffic requirements. The migration manager is invoked when the service monitor detects that any edge server is currently overloaded. For this, the service monitor periodically checks the traffic fluctuations of each edge server in every 30 seconds and updates the corresponding status into a database. If the service traffic exceeds the predefined threshold value (we use 70% in this paper), the service monitor triggers the migration manager to

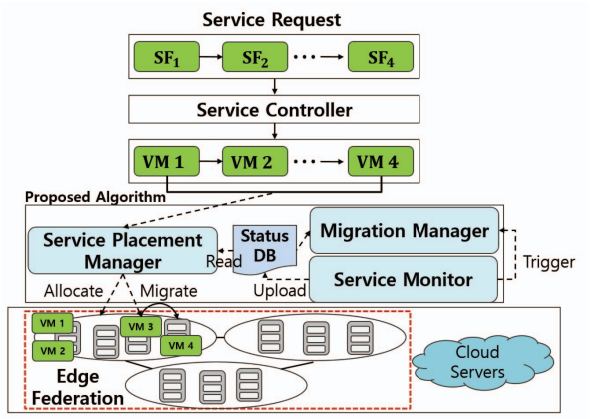


Figure 3. System Overview

migrate a service VM to proper nearby edge servers to avoid the QoS violation of the service.

IV. Evaluation

To show the effectiveness of the proposed algorithm, we have implemented the algorithm over a cloud simulator called `CloudsimSDN` [12]. We also compared its performance with that of a traditional service scheduling algorithm proposed for edge computing environments. For simplicity, we call the proposed algorithm as the actual resource-aware service scheduling algorithm in FEC (*RASSFEC*) and the comparison target algorithm as the non-resource-aware service scheduling algorithm in edge cloud (*NRASSEC*).

A. Experimental Environment

Topology For the simulation, we assume an FEC environment with 8 edge domains where there are 24 edge servers in each domain. Each edge server is equipped with one 16-core CPU and 32GB RAM. The ratio of pCPU to vCPU is 1 (i.e., no sharing) and thus 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 4 aggregate switches with a 10 Gbps link. Finally, each aggregate switch is connected to the 2 core switches with a 128 Gbps link to reach the 2 cloud servers. We assume that the cloud servers have unlimited computing capacity.

Energy Parameters For the energy parameters for a server and a switch such as peak power consumption and idle power consumption, we use the parameters suggested in [17].

Workloads We assume that two real-time application services such as *face recognition* and *online text translator* run at the same time with a ratio of 60% to 40% for the evaluation. Each application service consists of 3 different service functions. The specification of application services and service functions are summarized in Table I and Table II. The service traffic used for each application service is generated by using Weibull distribution [18] and the average packet size used for the latency model is referenced from [19]. With the service traffic discussed above, we generate two different types of service traffic such as low service traffic and high service traffic. The low service traffic is generated with a speed of 50 packets per second (PPS), while the high service traffic is generated with a speed of 200 PPS.

Table I
Specification of Application Services [20]

Application	Service Functions	Latency	Ratio
Face recognition	SuperHub - TPLink - Face recongition API	2.0 sec	60%
Text translator	Livebox - Netgear - Text translator API	1.5 sec	40%

Table II
Specification of Service Functions [20]

Type	CPU (min/max)	VM Size	Idle Proc Time
SuperHub	2 / 4	512 MB	186 ms
Livebox	1 / 2	256 MB	225 ms
TPLink	2 / 4	128 MB	214 ms
Netgear	2 / 4	384 MB	196 ms
Face recognition API	0.5 / 1	1 GB	122 ms
Text translator API	0.5 / 1	1 GB	122 ms

B. Comparison of Energy Efficiency

Fig. 4 shows the energy efficiency per service of *RASSFEC* normalized with respect to *NRASSEC*. As shown in Fig. 4, when the service traffic is low, *RASSFEC* outperforms *NRASSEC* by about 21%. Whereas, when the service traffic is high, the performance improvement is shortened to 10%.

This can be explained by the following reasons. When the service traffic is low, the possibility of violating the maximum traffic threshold in each service function is rare. Therefore, the *RASSFEC*'s placement policy that increases the level of service consolidation with minimum traffic requirements allows more VMs to be co-located in a single edge server. This results in high energy efficiency. In contrast, as the service traffic increases, it is more likely that the sum of traffic generated by all VMs running in a single edge server exceeds the maximum traffic threshold. As a result, more energy consumption is expected due to service migration. However, *RASSFEC* reconfigures services to minimize the service migration overhead in case of traffic fluctuation. This can minimize energy waste along the service paths.

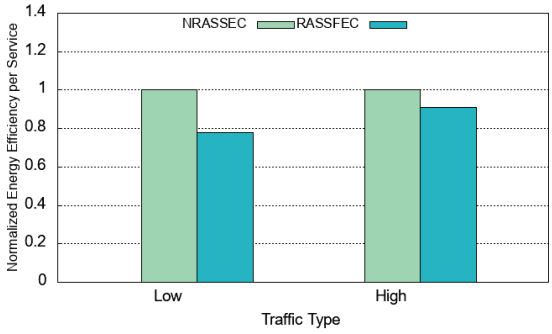


Figure 4. Comparison of Energy Efficiency per Service

C. Comparison of Service Violation Rate

Fig. 5 shows the normalized service violation rate of *RASSFEC* with respect to *NRASEC*. As shown in Fig. 5, *RASSFEC* generates almost similar service violations to *NRASEC* in low service traffic, while *RASSFEC* performs extremely better than *NRASEC* in high service traffic (i.e., about 80%).

This is because both algorithms prefer to placing services in an edge domain when the target edge domain has sufficient computing capacity. That is, when the service traffic is low, both algorithms place most services on edge servers instead of cloud servers, which results in low service violation rate. However, as the service traffic increases, *RASSFEC* places the services that cannot be deployed on edge servers to nearby edge servers, while *NRASEC* starts to move them to cloud servers. This creates more service violations since the transmission latency from an edge domain to cloud servers is relatively high. Although more service migration is expected in *RASSFEC* when the service traffic is high, the migrating VMs using *RASSFEC*'s migration policy are likely to be located at nearby edge servers instead of cloud servers. It is worthy to note that the transmission latency between two edge servers is shorter than the latency between edge servers and cloud servers.

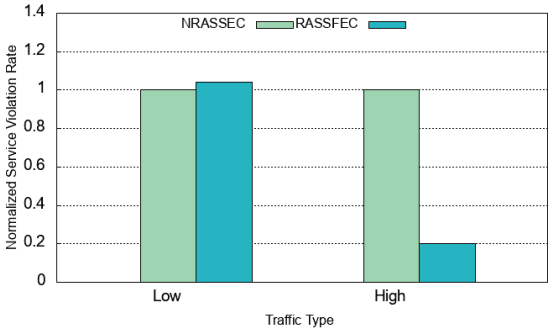


Figure 5. Comparison of Service Violation Rate

V. Conclusion

In this paper, we have proposed an energy efficient service placement and reconfiguration algorithms for FEC environments. The proposed algorithm minimizes energy consumption by consolidating as many VMs as possible in a single edge server and reduces the possibility of QoS violation by locating services in nearby edge servers instead of moving them to the cloud. Through simulations, we have also showed that the proposed algorithm is effective for reducing energy consumption as well as QoS violation rate. However, as we increase the service traffic, the performance gap is shortened due to the service migration overhead. This necessitates the need for more efficient reconfiguration algorithm to minimize the service migration overhead in future works.

VI. Acknowledgement

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] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] X. Cao, G. Tang, D. Guo, Y. Li, and W. Zhang, "Edge Federation: Towards an Integrated Service Provisioning Model," *arXiv preprint arXiv:1902.09055*, 2019.

[3] L. Ganesh, H. Weatherspoon, T. Marian, and K. Birman, "Integrated approach to data center power management," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1086–1096, 2013.

[4] S. Kim, S. Park, K. Youngjae, S. Kim, and K. Lee, "Vnf-eq: dynamic placement of virtual network functions for energy efficiency and qos guarantee in nfv," *Cluster Computing*, vol. 20, 09 2017.

[5] L. Nonde, T. E. H. El-Gorashi, and J. M. H. Elmirghani, "Energy efficient virtual network embedding for cloud networks," *Journal of Lightwave Technology*, vol. 33, no. 9, pp. 1828–1849, 2015.

[6] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 76–89, 2017.

[7] G. Sun, Y. Li, H. Yu, A. V. Vasilakos, X. Du, and M. Guizani, "Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks," *Future Generation Computer Systems*, vol. 91, pp. 347 – 360, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1831848X

[8] X. Shang, Z. Liu, and Y. Yang, "Network congestion-aware online service function chain placement and load balancing," in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP 2019. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3337821.3337850

[9] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 41–48.

[10] J. Son and R. Buyya, "Latency-aware virtualized network function provisioning for distributed edge clouds," *Journal of Systems and Software*, vol. 152, pp. 24 – 31, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219300391

[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '74. New York, NY, USA: Association for Computing Machinery, 1974, p. 47–63. [Online]. Available: https://doi.org/10.1145/800119.803884

[12] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "Cloudsimsdn: Modeling and simulation of software-defined cloud data centers," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 475–484.

[13] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012, p. 7.

[14] Q. Chen, P. Grosso, K. v. d. Veldt, C. d. Laat, R. Hofman, and H. Bal, "Profiling energy consumption of vms for green cloud computing," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011, pp. 768–775.

[15] X. Wang, X. Wang, K. Zheng, Y. Yao, and Q. Cao, "Correlation-aware traffic consolidation for power optimization of data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 992–1006, 2016.

[16] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," vol. 16, 06 2011, pp. 171–182.

[17] IBM, "IBM System x3550 M3 Power Specification," https://lenovopress.com/tips0804, 2019.

[18] M. A. Arfeen, K. Pawlikowski, D. McNickle, and A. Willig, "The role of the weibull distribution in internet traffic modeling," in *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, 2013, pp. 1–8.

[19] I. Antoniou, V. Ivanov, V. Ivanov, and P. Zrelov, "On the log-normal distribution of network traffic," *Physica D*, vol. 167, pp. 72–85, 03 2002.

[20] R. Cziva and D. P. Pezaros, "Container network functions: Bringing nfv to the network edge," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 24–31, 2017.