Dynamic Task Offload System Adapting to the State of Network Resources in Mobile Edge Computing

Hayata Satake, Yuki Kobayashi, Ryotaro Tani, Hiroshi Shigeno
Graduate School of Science and Technology
Keio University
3-14-1 Hiyoshi Kohoku-ku
Yokohama-shi Kanagawa Japan
{satake, kobayashi, tani, shigeno}@mos.ics.keio.ac.jp

***Abstract*—In disaster situation, collecting information about victims and disaster area quickly is required. However, it is expected that the communication environment is not fully prepared because communication facilities may be damaged. Mobile Edge Computing (MEC) is attracting attention, in which servers are distributed to the vicinity of end users. Thus, dynamic task offloading adapting to changes of network resources is useful in unstable network. This paper proposes a dynamic task offload system based on the state of network. The proposed offload system consists of two mechanisms. First mechanism is Software Defined Mobile Edge Computing (SDMEC) architecture. Second mechanism is dynamic task placement based on the state of network resources. We formulated an optimization problem to minimize the total response time while application execution. In the proposed offload system, multiple tasks are placed dynamically depending on the load of servers and network resources to minimize response time of application. We implemented the prototype of the proposed offload system and evaluated its performance.**

***Index Terms*—Mobile Edge Computing, SDN, OpenFlow, Offloading;**

I. Introduction

In disaster situation, collecting information about victims and disaster area quickly is required. Analyzing these information leads to efficient allocation of necessary resources such as rescue teams. It is expected that the communication environment is not fully prepared because communication facilities may be damaged. Furthermore, analyzing images and videos on battery-powered mobile devices increase power consumption. Therefore, mobile devices are not suitable for executing a high-load application independently. Cloud computing enables to reduce the load of mobile devices by aggregating and processing tasks from end users. However, cloud computing intensively processes high-load applications on the cloud server, so that the load of network and delay of application processing can increase.

Hence, Mobile Edge Computing (MEC) attracts attention for solving the problems of cloud computing. In MEC, servers are distributed to the vicinity of end users. Therefore, delay while processing applications may be reduced by shortening the distance between servers and users. In MEC, dividing application into tasks enables application processing to utilize multiple edge servers and cloud servers. Management of edge networks is difficult because servers are widely distributed. Therefore, SDN (Software Defined Networking) that realizes centralized network control is introduced to MEC. Control plane and data plane are separated, and centralized network control is performed by SDN controller. As a result, flexible network control is possible. OpenFlow is one of techniques which provides SDN. OpenFlow is a standard communication interface that enables programmable network control by separating control plane and forwarding plane.

This paper proposes a dynamic task offload system based on the state of network and the load of servers. The proposed offload system consists of two mechanisms. First mechanism is implementation of Software Defined Mobile Edge Computing (SDMEC). Second mechanism is dynamic task placement considering network resources. We formulate an optimization problem to minimize the total response time while application execution. We implement a prototype of the proposed offload system and evaluate its performance. In experiment, we use face recognition application and confirm that routing and task execution are performed on the network and server for minimizing response time.

The rest of this paper is organized as follows. In Section 2, related work is presented. In Section 3, the proposed dynamic task offload system is described. In Section 4, the implementation of the prototype system of the proposed offload system is explained. Experimental evaluations are also discussed in Section 4. Finally, concluding remarks are mentioned in Section 5.

II. Related Work

In this section, we explain Mobile Edge Computing (MEC) and Software Defined Mobile Edge Computing (SDMEC), and mention its challenges as related work.

A. Overview of Mobile Edge Computing (MEC)

Nowadays, mobile devices such as smartphones and laptops are widespread. Executing application on mobile devices is limited due to their poor battery resources. Cloud computing can be a solution for this problem [1], which offloads application on mobile devices to cloud servers and reduces the load of mobile devices. However, with the widespread use of mobile devices, the cloud server processes all data from mobile devices intensively, a large amount of data from end users squeezes network resources between mobile devices and cloud servers. As a result, delay while executing applications increases which requires real-time performance. Hence, Mobile Edge Computing (MEC) [2] attracts attention for the purpose of improving delay problem of cloud computing [3]. The feature of MEC is that servers which have the same role as the cloud server are distributed to the vicinity of end users. Therefore, it is possible to reduce delay by shortening the distance between end users and edge servers. In addition, offloading part of the applications to edge servers enables to reduce the load of network resources between mobile devices and cloud servers.

B. Software Defined Mobile Edge Computing (SDMEC)

MEC enables to reduce the load of mobile devices, in addition to solving the delay problems of cloud computing. A challenge of MEC is that edge servers are placed distributedly, thus managing such complicated network [4] [5]. To overcome this challenge, Software Defined Mobile Edge Computing (SDMEC) is proposed to manage distributed networks by introducing SDN which enables flexible network control [6]. Monetary cost increases due to managing an enormous number of mobile devices. Therefore, SDN is software based management, it is possible to reduce the monetary cost compared with hardware based management [7]. Considering mobility of users, handover of heterogeneous networks is possible by network monitoring by SDN [8]. By monitoring the load of link among servers in edge network, SDN controller enables routing based on the latency. In addition, by monitoring the resources of each server, allocating resources to each edge server [9] is possible.

C. Offloading of image processing in MEC

In disaster situation, collecting information about victims and disaster area quickly is required. Analyzing these information leads to efficient allocation of necessary resources such as rescue teams. Especially, analyzing images of victims and disaster area is effective way in disaster situation. Ashish et al. [10] propose a fog computing-based data offload mechanism for real-time analysis of disaster information. Although cloud based data analysis related to disaster situation involves a significant delay, offloading to fog nodes makes the process more effective. Trinh et al. [11] propose offloading image processing mechanism in mobile edge computing assuming disaster situation. This mechanism offloads face recognition application focusing on calculation of the load of edge servers and cloud servers, and power consumption of mobile devices. Meanwhile, Minh et al. [12] propose an architecture of mobile edge computing considering fault tolerance in unstable network such as disaster situation. When the network links among edge servers and client are disconnected, application processing is offloaded to nearby mobile devices. In disaster situation, it is necessary to analyze visual data gathered from disaster area quickly and in real time. However, efficient application execution is required because usable resources are limited.

In related work, offload mechanisms consider the power consumption of the mobile devices and computational load of edge servers and cloud servers. However, considering unstable network situation such as disaster situation, task offload considering network resources in addition to computational resources is more beneficial. Therefore, our proposed offload system dynamically performs task placement while executing application considering the state of network, in addition to computation resources of each server.

III. Proposed Dynamic Task Offload System

This section proposes dynamic task offload system adapting to the state of network resources in MEC environment. The purpose of the proposed offload system is to realize dynamic offloading of image processing applications in unstable network situation such as disaster situation. To achieve this purpose, the proposed offload system places multiple tasks dynamically and executes route selection considering network resource and the load of servers.

A. Overview of the Proposed Task Offload System

Figure 1 shows an overview of proposed offload system. The proposed offload system has following two features to enable dynamic task offloading considering the state of network resources.

First mechanism is implementing framework of Software Defined Mobile Edge Computing (SDMEC). For edge-cloud network management, the proposed offload system introduces OpenFlow to MEC environment. By introducing OpenFlow to MEC, the data plane of edge-cloud network converts to OpenFlow switch and the OpenFlow controller manages data plane of edge-cloud network. The proposed offload system consists of edge servers, a cloud server and OpenFlow network. A mobile device (we call it Client) sends requests to the controller node of MEC. We introduce MEC controller as controller node which includes OpenFlow controller and the role of Offloading management.

Second mechanism is dynamic task offloading considering the state of network resources. MEC controller solves an optimization problem based on the processing at each edge server and a cloud server, and determines the task allocation and the route of application processing. In the proposed

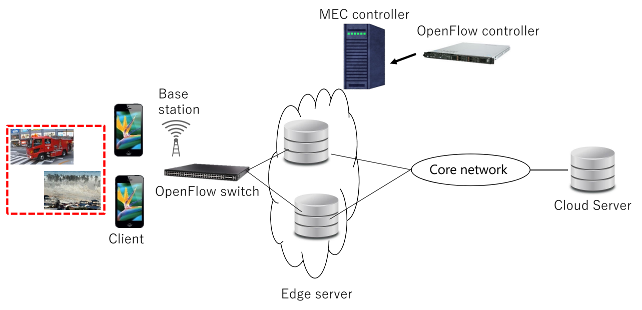

Fig. 1. Overview of the Proposed Offload System.

offload system, MEC controller places tasks dynamically and controls routing considering the state of network resources and task execution time so as to minimize total response time.

After execution of each task of face recognition at each edge server or a cloud server, the result is returned to the client. Each task of face recognition is pre-processing, feature extraction and counting of the number of people in given images. In the proposed offload system, these tasks are allocated to each server based on the state of network and the load of servers, and total response time will be reduced.

B. Framework of Software Defined Mobile Edge Computing (SDMEC)

In MEC environment, edge servers are distributed to the vicinity of end users. Thus, managing such complicated distributed edge network, while maintaining connectivity and providing services are difficult. However, OpenFlow enables flexible management of such complicated network. Figure 2 shows a framework of introducing OpenFlow to edge-cloud network.

OpenFlow enables flexible routing control by separating the control plane and the data plane of the network. In the proposed offload system, data plane of edge-cloud network converts to OpenFlow switch and the OpenFlow controller manages data plane of edge-cloud network. We introduce MEC controller as controller node which includes OpenFlow controller and the role of Offloading management. MEC controller collects the state of network and information of each server and updates these parameters at regular intervals. Information of each server assumes the CPU usage and memory usage of servers and MEC controller monitors the load of servers by monitoring these information. The state of network is estimated from the amount of traffic per unit time which is measured by OpenFlow switch. Specifically, OpenFlow switch measures traffic flow to each port. By transmitting the Stats-Request message at regular intervals from MEC controller to OpenFlow switch, it is possible to collect the statistical information stored in the flow table on the OpenFlow switch. MEC controller calculates residual bandwidth by obtaining the difference between the set value of links and the traffic flow to each port. Thus, MEC controller monitors the state of edge-cloud network via OpenFlow switch. As a result, MEC controller can monitor the state of edge-cloud network in real time. The proposed offload system enables routing control related to task processing of applications.

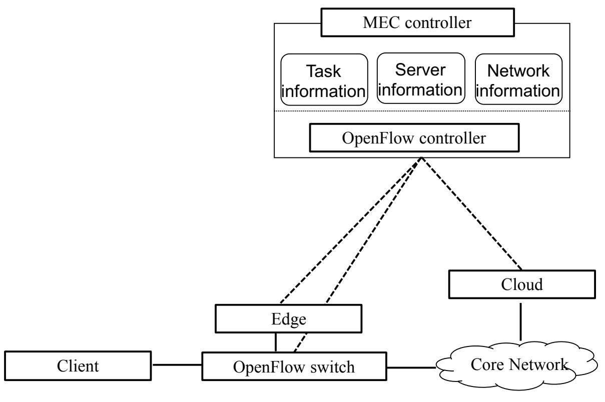

Fig. 2. Framework of Software Defined Mobile Edge Computing (SDMEC).

C. Dynamic Task Offloading

In the proposed offload system, MEC controller places multiple tasks considering the load of servers and network resources. In the proposed offload system, application is divided to multiple tasks. In order to place these multiple tasks, we formulate an optimization problem to minimize the total response time while application execution. MEC controller solves optimization problem and determines task placement, and routing of allocating tasks. We introduce MIP (Mixed Integer Programming) problem with parameters such as bandwidth and task execution time. The objective function and constraint conditions used in the optimization problem are as shown in the following equations, and Table I shows the parameters.

Minimize

$$\sum_{i \in N} (T^i_{M^i} - t^i_0) \tag{1}$$

Subject to

$$T^i_{M^i} = \sum_{j=0}^{M^i} T^i_j, j \in N^i, i \in N \tag{2}$$

$$T^i_j = \sum_{i \in N^i} (t^i_j + x^i_j * e^i_j), j \in N^i, i \in N \tag{3}$$

$$x^i_j = \{x^i_0, x^i_1, \cdots, x^i_{M^i}\} \tag{4}$$

$$W^i_{j,j+1} = \omega^i_{j,j+1} + \frac{y^i_{j,j+1} * d^i_{j,j+1}}{b} \tag{5}$$

$$\omega^i_{j,j+1} \geq T^i_j, j \in N^i, i \in N \tag{6}$$

$$t^i_{j+1} \geq W^i_{j,j+1}, j \in N^i, i \in N \tag{7}$$

$$y^i_{j,j+1} \geq x^i_j - x^i_{j+1}, j \in N^i, i \in N \tag{8}$$

$$y^i_{j,j+1} \geq x^i_{j+1} - x^i_j, j \in N^i, i \in N \tag{9}$$

$$y^i_{j,j+1} \leq x^i_{j+1} + x^i_j, j \in N^i, i \in N \tag{10}$$

Function (1) is an objective function which minimizes total response time of application i. Equation (2) to (10) represent constraint conditions. Equation (2) shows end time of task^i_M which is sum of task$_j$ of application i. Equations (3) and (4) show execution time of task$_j$ of application i. Execution time of task$_j$ is calculated by the sum of start time of task$_j$ and execution time of task$_j$. x^i_j takes a value of 0 or 1 to decide whether to process the task on the same server or offload task

TABLE I
PARAMETER OF OPTIMIZATION PROBLEM.

parameter	content
N	set of application i
N^i	set of task, N^i = {0,1, $\cdots$,M^i }
e^i_j	execution time of task$_i$
$d^i_{j,j+1}$	size of data transferring from task$_j$ to task$_{j+1}$
$\omega^i_{j,j+1}$	start time of data transmission from task$_j$ to task$_{j+1}$ of application i
$W^i_{j,j+1}$	end time of data transmission from task$_j$to task$_j + 1$
x^i_j	decision variables for task$_j$ of application i, $x^i_j = \{0, 1\}$
$y^i_{j,j+1}$	binary integer for task$_j$ and task$_{j+1}$ of application i are executed on the same server
b	bandwidth of links
t^i_j	start time of task$_j$ of application i
T^i_j	end time of task$_j$ of application i

to other server. If x^i_j is 1, task is offloaded to other server, and 0 otherwise. Equation (5) shows data transferring time from task$_j$ to task$_{j+1}$. End time of data transmission from task$_j$ to task$_{j+1}$ is the sum of start time of data transmission and value which is obtained by dividing the amount of data by bandwidth of links. y^i_j is binary integer determined by the value of x^i_j, and it takes 1, if task is offloaded to other computational resource, 0 otherwise. Equation (6) shows start time of data transmission from task$_j$ to task$_{j+1}$ after execution of task$_j$ is completed. Equation (7) shows start time of task$_{j+1}$ processing after the transfer from task$_j$ to task$_{j+1}$. Equations (8) to (10) indicate whether or not a plurality of tasks are processed on the same server. By solving the defined optimization problem at regular intervals, it is possible to place multiple tasks. OpenFlow switch measures the bandwidth of each link and it is substituted and updated at regular intervals.

The procedure of executing application is described below, and Figure 3 shows its flow.

Step1 Client sends request to MEC controller.

Step2 MEC controller transmits routing information to each server based on the solution obtained by the optimization problem, and places tasks to each server.

Step3 Client sends an image to server$_i$.

Step4 After server$_i$ receives the image, if server$_i$ decides to execute task$_j$ based on information received in Step2, then executes task$_j$ and process moves to Step5. Otherwise, sever$_i$ sends image to server$_{i+k}$ and process moves to Step6.

Step5 If server$_i$ continues executing task$_{j+1}$, executes task$_{j+1}$ continuously, otherwise sends processed data to server$_{i+k}$ then process moves to Step6.

Step6 Based on information received in Step2, server$_{i+k}$ executes task$_j$, if *j=j+1*, and process moves to Step5. Otherwise, the image or processing data is transmitted to server$_{i+k}$, and process moves to Step4.

Step7 Step4 to Step6 are repeated, and when the number of tasks to be processed reaches the maximum number *M*, execution result is sent back to client.

In the proposed offload system, dynamic task placement and route selection are enabled by solving the objective function defined by Function (1) at regular intervals.

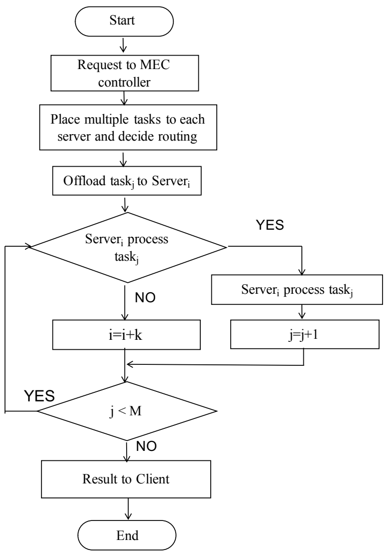
Start
Request to MEC controller
Place multiple tasks to each server and decide routing
Offload task$_j$ to Server$_i$
Server$_i$ process task$_j$
YES
NO
Server$_i$ process task$_j$
i=i+k
j=j+1
YES
j < M
NO
Result to Client
End

Fig. 3. Flow of Multiple Task Processing.

IV. IMPLEMENTATION AND EVALUATION

This section describes a prototype implementation of the proposed offload system and evaluation of experiment.

A. Implementation Environment

In order to confirm the operation of the proposed offload system, we implemented a prototype system. Table II shows specifications of a prototype system of the proposed offload system. MEC controller and OpenFlow switch consist of one machine. Application tested in experiment was face recognition application. Face recognition application was implemented by OpenCV [13]. There are multiple tasks for face recognition, pre-processing, feature extraction and counting the number of people. In addition, Haar-like feature [14] was used for facial feature quantity extraction. Table III describes detailed values of experiment.

TABLE II
SPECIFICATIONS OF PROTOTYPE SYSTEM.

Protocol	
OpenFlow	OpenFlow 1.3 [15]
Software	
OpenFlow controller	Ryu ver.4.29 [16]
OpenFlow switch	Open vSwitch ver. 2.8.4 [17]
OpenCV	OpenCV ver.3.3.0 [13]
Hardware	
MEC controller	Ubuntu16.04 Intel(R) Core(TM) i7-4790@3.60GHz, 16GB
Edge1	Ubuntu16.04 Intel(R) Core(TM) i7-3770@3.40GHz, 8GB
Edge2	Ubuntu16.04 Intel(R) Pentium (R) CPU G3420@3.20GHz, 8GB
Cloud	Ubuntu16.04 Intel(R) Core(TM) i5-4590@3.30GHz, 8GB

TABLE III
DETAILED VALUES OF EXPERIMENT.

Parameter of Experiment	
Image transmission interval	3 [s]
Bandwidth detection interval	20 [s]
Solving interval of optimization problem	20 [s]
Size of image	181 [Kbyte]
Delay of links	
Edge1 - Cloud	50 [msec]
Edge2 - Cloud	50 [msec]
Client - Edge1	5 [msec]
Client - Edge2	5 [msec]
Edge1 - Edge2	10 [msec]
CPU usage	
Edge1	80 [%]
Edge2	60 [%]
Cloud	0 [%]

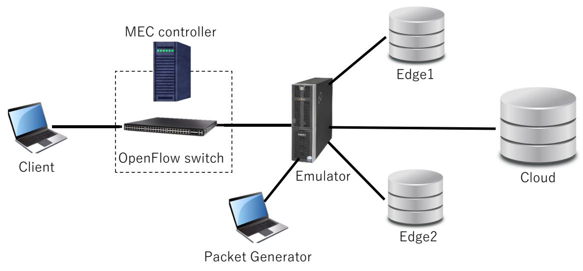

Fig. 4. Experimental Environment.

In experiment, we evaluated response time. OpenFlow switch detects the bandwidth of each link. MEC controller calculates residual bandwidth from the bandwidth status obtained from OpenFlow switch and solves optimization problem at regular intervals based on task execution time of servers and residual bandwidth of each link. After that, task placement and route control are performed for each edge server and a cloud server. Client sends image to edge server at regular intervals. UDP is used for transmission of image, and the image is transmitted dividedly. In order to make difference of the bandwidth of each link, background traffic is transmitted by iperf [18]. For background traffic, 70 Mbps traffic is transmitted every 20 seconds between Edge 1 or Edge 2 and client. Thus, bandwidth of links changes every 20 seconds. For background traffic, the packet size was set to 1470 Bytes and generated as CBR (Constant Bit Rate) traffic by UDP. CPU usage was changed in the experiment to reproduce the case that computational power of each server changes. Thus, task execution time is variable depending on CPU usage. In order to change CPU usage, we used *CPUlimit* command [19]. In case that the load of each server is none, execution time of each server is assumed to be almost same. Execution time of each task is based on the average of preliminary 100 times execution. Response time is defined by time from client transmits the image to result returns to client. Figure 4 shows experimental environment. Packet generator sends background traffic between client and edge servers. Thus, bandwidth of links is changed at regular intervals. The packet generator changes the amount of traffic between Edge1 and Client, between Edge2 and Client.

TABLE IV
TASK PLACEMENT PATTERN INDEX.

Pattern Index	Edge1	Edge2	Cloud
1 (offload)	P	FC	
2 (offload)		P	FC
3 (offload)	P		FC
4 (no offload)	P, FC		
5 (no offload)		P, FC	
6 (no offload)			P, FC

Table IV shows pattern index of task placement to classify task placements of the results. Pattern index P shows the task of pre-process, and pattern index FC shows the tasks of feature extraction and counting the number of people in the image.

B. Evaluation : Response Time of Application

Figure 5 shows the transition of the response time within the experimental time. In this experiment, the state of network was varied at every 20 seconds, because of varying background traffic between Edge1 to Client and Edge2 to Client at every 20 seconds alternately. MEC controller solved optimization problem at every 20 seconds during the experiment and placed multiple tasks and changed routing of task allocation. In the proposed offload system, pre-processing and feature amount extraction were processed on Edge1 and Cloud respectively until 20 seconds. In addition, up to 40 seconds afterwards, pre-processing was processed on Edge2 and feature extraction was processed on Cloud. After that, the pattern index 3 and 2 were repeated. This transition means that task placement was changed depending on the variation of background traffic between Edge1 to Client and Edge2 to Client. This results shows that the proposed offload system enabled task placement depending on the state of network.

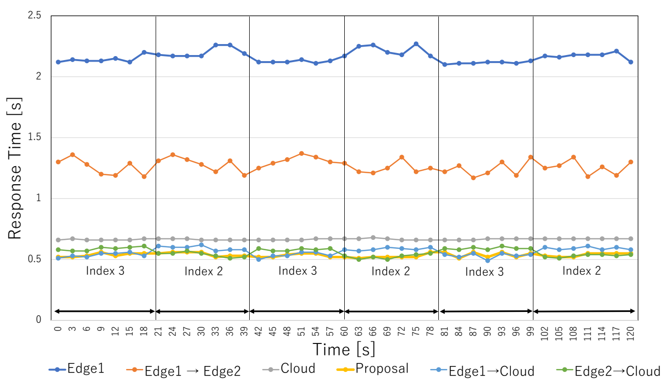

Fig. 5. Transition of Response Time.

We placed tasks to specific servers intentionally to compare task placement of the proposed offload system with other task placement patterns. Figure 6 shows the average response time of the proposed offload system and other task placement patterns. Response time consists of the sum of data transmission time and application execution time. This result shows that the proposed offload system enabled to place multiple tasks to minimize response time compared with other task placement patterns.

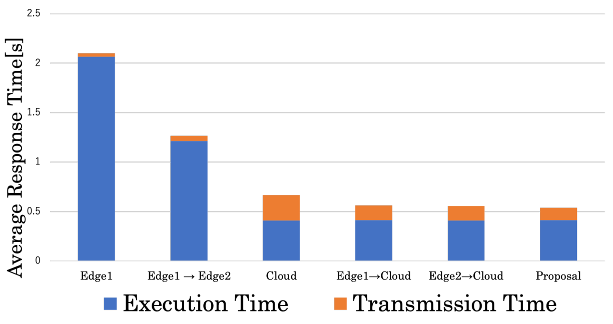

Fig. 6. Comparison of the Proposed Offload System and Other Task Placement.

V. Conclusion

In this paper, we proposed dynamic task offload system. The purpose of the proposed offload system was dynamic task placement considering the state of network and the load of servers. The proposed offload system consists of following two mechanisms. First mechanism is introduction of SDMEC. For edge-cloud network management, we introduced OpenFlow to MEC environment. By introducing OpenFlow, the data plane of edge-cloud network converts to OpenFlow switch and the OpenFlow controller manages control plane of edge-cloud network. Second mechanism was dynamic task placement considering network resources. We formulated an optimization problem to minimize the total response time while application execution. MEC controller solves optimization problem based on the load of each server and the state of network, and updates it at regular intervals. Thus, MEC controller enables dynamic task placement to each server.

We implemented a prototype of the proposed offload system and evaluated its performance. In experiment, we tested face recognition application and confirmed that tasks were executed with the minimum response time while client transmits image at regular intervals. In experiment, MEC controller solved optimization problems at regular intervals based on task execution time of each server and the state of network, and confirmed that MEC controller placed multiple tasks dynamically. In addition, we confirmed that the proposed offload system minimized response time. Therefore, evaluation results showed that the proposed offload system placed multiple tasks dynamically based on the load of servers and the state of network. In addition, we confirmed that the proposed offload system processed multiple tasks to minimize response time of application execution.

Acknowledgement

Part of this work was carried out under the Cooperative Research Project Program of the Research Institute of Electrical Communication, Tohoku University.

References

[1] S. M. A. Ataallah, S. M. Nassar, and E. E. Hemayed, “Fault tolerance in cloud computing - survey,” in *Proceedings of 2015 11th International Computer Engineering Conference (ICENCO)*, Dec 2015, pp. 241–245.

[2] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, “Challenges and opportunities in edge computing,” in *Proceedings of 2016 IEEE International Conference on Smart Cloud (SmartCloud)*, Nov 2016, pp. 20–26.

[3] P. Li, Y. Luo, K. Wang, and K. Yang, “Energy minimization and offloading number maximization in wireless mobile edge computing,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec 2018, pp. 1–6.

[4] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “Edge computing enabling the internet of things,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec Proceedings of 2015, pp. 603–608.

[5] T. Kan, Y. Chiang, and H. Wei, “Qos-aware mobile edge computing system: Multi-server multi-user scenario,” in *2018 IEEE Globecom Workshops (GC Wkshps)*, Dec 2018, pp. 1–6.

[6] A. C. Baktir, A. Ozgovde, and C. Ersoy, “How can edge computing benefit from software-defined networking: A survey, use cases, and future directions,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, Fourthquarter 2017.

[7] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, “How will nfv/sdn transform service provider opex?” *IEEE Network*, vol. 29, no. 3, pp. 60–67, May 2015.

[8] S. Gringeri, N. Bitar, and T. J. Xia, “Extending software defined network principles to include optical transport,” *IEEE Communications Magazine*, vol. 51, no. 3, pp. 32–40, March 2013.

[9] S. W. Prakash and P. Deepalakshmi, “Server-based dynamic load balancing,” in *2017 International Conference on Networks Advances in Computational Technologies (NetACT)*, July Proceedings of 2017, pp. 25–28.

[10] A. Rauniyar, P. Engelstad, B. Feng, and D. V. Thanh, “Crowdsourcing-based disaster management using fog computing in internet of things paradigm,” in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, Nov Proceedings of 2016, pp. 490–494.

[11] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan, “Energy-aware mobile edge computing for low-latency visual data processing,” in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, Aug 2017, pp. 128–133.

[12] M. Le, Z. Song, Y. Kwon, and E. Tilevich, “Reliable and efficient mobile edge computing in highly dynamic and volatile environments,” in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May Proceedings of 2017, pp. 113–120.

[13] M. Agrawal, K. Konolige, and M. Blas, “Censure: Center surround extremas for realtime feature detection and matching,” vol. 5305, 10 2008, pp. 102–115.

[14] T. Mita, T. Kaneko, and O. Hori, “Joint haar-like features for face detection,” in *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, vol. 2, Oct Proceedings of 2005, pp. 1619–1626 Vol. 2.

[15] Open Networking Foundation, “OpenFlow - Open Networking Foundation,” [Accessed Jan. 12, 2019]. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow

[16] Ryu SDN Framework Community, “Ryu SDN Framework,” [Accessed Jan. 12, 2019]. [Online]. Available: http://osrg.github.io/ryu/

[17] Linux Foundation, “Open vSwitch,” [Accessed Jan. 12, 2019]. [Online]. Available: http://openvswitch.org/

[18] V. Gueant, “iPerf - The TCP, UDP and SCTP network bandwidth measurement tool,” [Accessed Jan. 12, 2019]. [Online]. Available: https://iperf.fr/

[19] Angelo Marletta, “cpulimit,” [Accessed Jan. 12, 2019]. [Online]. Available: https://github.com/opsengine/cpulimit