Edge computing that utilizes in-network CPUs to achieve high capacity and interruption tolerance with fewer edge servers

Koki MURAMATSU
Graduate School of Science and Technology
Keio University
Kanagawa, Japan
koki.muramatsu@yamanaka.ics.keio.ac.jp

Yoshihiko UEMATSU
Graduate School of Science and Technology
Keio University
Kanagawa, Japan
yoshihiko.uematsu@yamanaka.ics.keio.ac.jp

Satoru OKAMOTO
Graduate School of Science and Technology
Keio University
Kanagawa, Japan
okamoto@ieee.org

Naoaki YAMANAKA
Graduate School of Science and Technology
Keio University
Kanagawa, Japan
yamanaka@keio.jp

***Abstract*— We have proposed AMec (Access-Metro edge computing), an edge computing system that utilizes surplus computing resources in access metro networks. Although the use of CPUs on network devices was expected to reduce the number of edge servers, there were concerns about frequent interruption and pod's relocation due to executing their main job such as information gathering and routing reconfiguration, unlike MEC （Multi-access Edge Computing), which only uses dedicated edge servers. To avoid the pod's relocation, in this paper, we propose an allocation method that calculates the interruption occurrence rate when each pod of the application is placed in each node by using the usage duration characteristics of the application and allocates the pods in a way that the value is below a threshold. Under conditions where a certain level of interruption can be acceptable, it is confirmed that AMec with the proposed method can reduce the number of edge servers by 62.5 % ~ 66.7 % compared to MEC.**

***Keywords*— MEC, Edge computing, Kubernetes, White-box Network device**

I. INTRODUCTION

Today, demand for edge computing is increasing and the number of applications that require low latency or large resources on user devices is also increasing [1]. Besides, the traffic to clouds is more and more being enlarged. In addition, today's network devices are equipped with multicore CPUs (Central Processing Units) and they are being White-boxed [2]. So, products that can execute user applications on their surplus resources originally for their main job such as information gathering and routing reconfiguration are becoming popular [3][4].

Based on the above, we have proposed AMec (Access-metro Edge Computing), a concept of edge computing that uses surplus in-network computing resources such as CPUs on network devices [5][6] (Fig. 1). It can be one of the solutions to reduce CAPEX (Capital Expenditure) to install edge servers.

Although the use of CPUs on network devices was expected to reduce the number of edge servers, there are concerns that a node becomes unavailable while a pod is running due to unavailable time for executing their main job (referred to here as "interruption"), unlike MEC which uses dedicated edge servers.

We have proposed the pod relocation method for when a node becomes unavailable in [6] and their working has been confirmed by experiments in [7], but relocations caused by interruptions should be avoided as much as possible, because the relocation of pods reduces the efficiency of resource utilization, increases the workload on the AMec controller, and requires migration if it is state-full pods.

To avoid the pod's relocation, in this paper, we propose an allocation method that calculates the interruption occurrence rate when each pod of the application is placed in each node by using the usage duration characteristics of the application and allocates the pods in a way that the value is below a threshold. Then, we prove that AMec with the proposed method can reduce the number of edge servers in simulation compared to MEC, keeping a low interruption rate.

Our contribution is to show the following two points by simulation. We proved that our idea of AMec, which is edge computing using surplus computing resources such as network equipment, can significantly reduce the number of edge servers compared to MEC, which only uses dedicated edge servers. We also show that the proposed allocation algorithm can reduce the occurrence rate of "interruption", a problem inherent to AMec.

This work is partly supported by the R&D of innovative optical network technologies for supporting new social infrastructure project (JMPI00316) funded by the Ministry of Internal Affairs and Communications Japan and JGN (TB-A22001).

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

This paper is organized as follows: In section 2, we outline the research goals and architecture of AMec. Section 3 presents the system sequence of AMec. Three important situations in AMec are described: when a worker joins a cluster, when accepting processing from a user, and when workers become unavailable. Section 4 describes the proposed allocation algorithm, details of the simulation, and the result which indicates that AMec can reduce the number of edge servers compared to MEC.

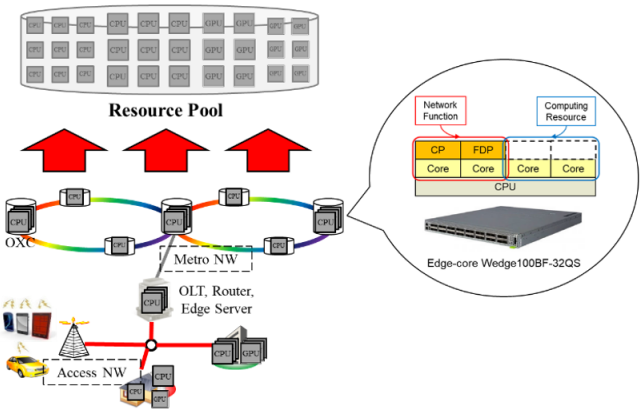

Fig. 1 Image of resource pool consisting of xPUs on heterogeneous devices

II. AMEC (ACCESS-METRO EDGE COMPUTING)

A. Research Goals

Ordinary MEC (Multi-access Edge Computing) needs to install dedicated servers for MEC at edge sites. However, as targets of computing resources in AMec, we assume CPUs on network devices such as OLTs (Optical Line Terminations), routers, switches, ROADMs (Reconfigurable Optical Add/Drop Multiplexers), etc. owned by telecommunication carriers. Then in the future, we are also considering utilizing underutilized home game consoles and IoT devices.

By using those heterogeneous devices, AMec plans to achieve the common MEC benefits such as reduction of latency, traffic, and processing load on user equipment while reducing CAPEX for edge server installations.

- Executing third-party applications offloaded by user devices / distributed from clouds
- Managing computing devices that join and leave the resource pool
- Providing optimum edge resources according to the situation of available resources, users, and applications

B. System Architecture

As described above, AMec makes the resource pool with surplus computing resources. The AMec Overall Architecture is shown in Fig. 2.

AMec Controller works with the master of each cluster to manage multiple clusters, providing functionality that is difficult to achieve with the master alone (i.e. complicated allocation algorism). The front end is the point of contact for users.

When a user asks the frontend to offload some processing, the frontend takes instructions from AMec Controller and offloads them to optimum clusters and resources. Basically, the user interacts only with the front end and is unaware of where the requested processing will be offloaded.

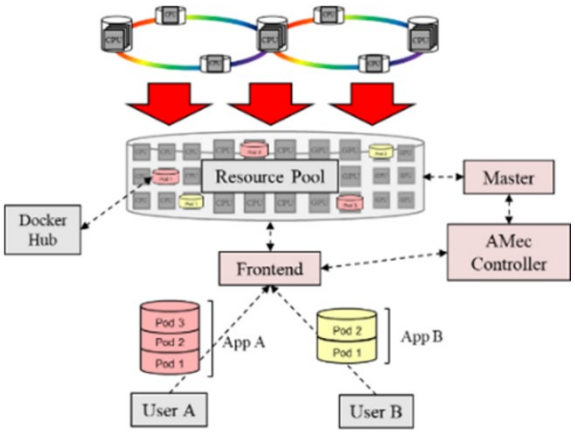

Fig. 2 AMec Overall Architecture

Due to container-based system configuration, third-party containers can be pulled from docker-hub and run on AMec. We have already confirmed application pods can be run on a white-box switch [7].

Because of utilizing surplus computing resources, there are times when resources on network devices are reclaimed for their original tasks, such as information gathering and routing reconfiguration, and then are not available for AMec tasks. Therefore, computing devices join and leave the resource pool and it is a key feature of AMec.

Fig. 3 shows the internal structure of the resource pool.

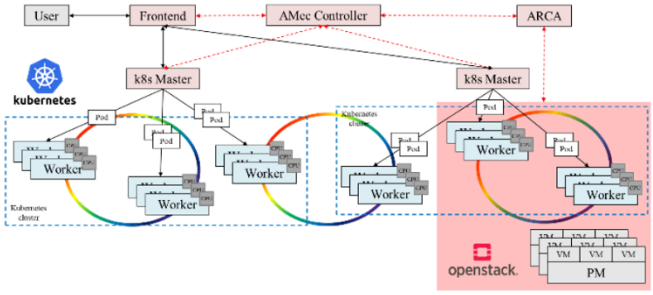

Fig. 3 Architecture of Resource Pool

The clusters are built using Kubernetes. In the future, one cluster will be built for each prefectural unit, consisting of network devices and edge servers in carrier buildings. The AMec controller collaborates with Kubernetes masters and assigns processing requests from users to the optimum node in the clusters according to the situation of available resources, users, and applications.

As connections to external services, VMs (Virtual Machines) on physical machines managed by OpenStack can also join AMec clusters, and this was used in collaboration with ARCA [7][8].

In past research, we have conducted an experiment where a pod on AMec at Keio Yagami received live streaming from

a smartphone at NICT Koganei via JGN [7][9]. Specifically, we have made an original web server docker image and uploaded it to Docker-Hub. AMec pulled the image and ran a web server to stream the video received from the smartphone. We have proved that AMec can run multiple third-party applications and can collaborate with applications in external networks.

III. System Sequence

In this section, we present three system sequences of AMec. These sequences were proposed in [6] and their working has been confirmed by experiments in [7].

A. *When workers join clusters*

As described above, computing devices join and leave the resource pool because AMec does not use dedicated computing resources. Fig.4 shows the AMec system sequence when a worker joins a cluster.

The section highlighted in red shows that the worker is marked as unavailable, being preoccupied with its primary function. After a while, the worker becomes available and asks to join the AMec cluster. This request is fielded by the AMec controller and it returns information to join the clusters to the worker.

We have implemented a mechanism to automatically join the AMec cluster by executing a script file for AMec participation after the white-box switch detects its own availability [7].

Specifically, when a white-box switch notifies the AMec controller that it is ready to join AMec, the AMec controller returns the IP of the master node in the cluster where the white-box switch should join, and the token required to join. These exchanges are done by HTTPS.

Then, the white-box switch joins the cluster using the information received from the AMec controller.

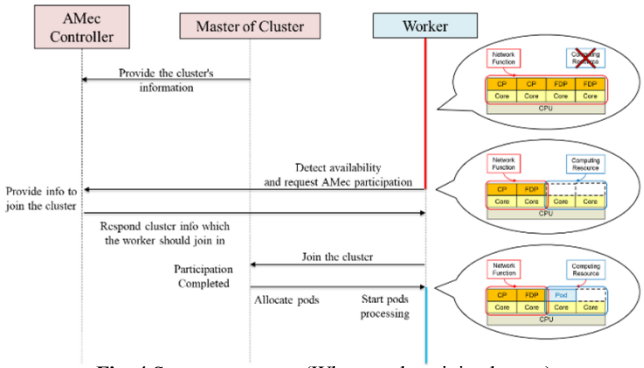

Fig. 4 System sequence (When workers join clusters)

B. *When accepting processing from a user*

Fig.5 shows the system sequence when accepting and allocating processing from a user.

Basically, users communicate only with front ends in AMec. A front end receives an offloading request from a user and the AMec controller judges which node the pods in the apps should be processed in. Then, the user offloads their processing to the cluster via the front end.

To judge which nodes the pods in the apps should be processed in, the AMec controller gets information of each node in the clusters from its master. The allocation algorithm used in this step is described in section 4.

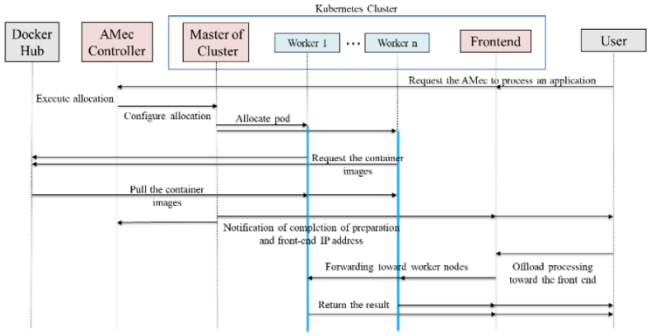

Fig. 5 System sequence (When accepting processing from a user)

C. *When workers become unavailable*

Fig.6 shows the system sequence when workers become unavailable, entering unavailable time due to their main job.

The most important point is we aim for proactive dealing to prevent service disconnections from occurring. If the worker that is processing a pod is to be unavailable, the worker notifies the AMec controller in advance of entering unavailable time.

Then, the AMec controller makes a master of cluster duplicate pod on another available node. Specifically, the master of the cluster renews the deployment file for the application.

Experiments have shown that it takes about 7 seconds for the AMec controller to deploy a pod on a new node after receiving the notification.

Although this is how we deal with the situation when a node becomes unavailable, it should be avoided as much as possible to have a node become unavailable while a pod is running. This is because the relocation of pods reduces the efficiency of resource utilization, increases the workload on the AMec controller, and requires migration if it is state-full pods. Therefore, to avoid having a node which pods running on become unavailable is one of the major research issues in AMec.

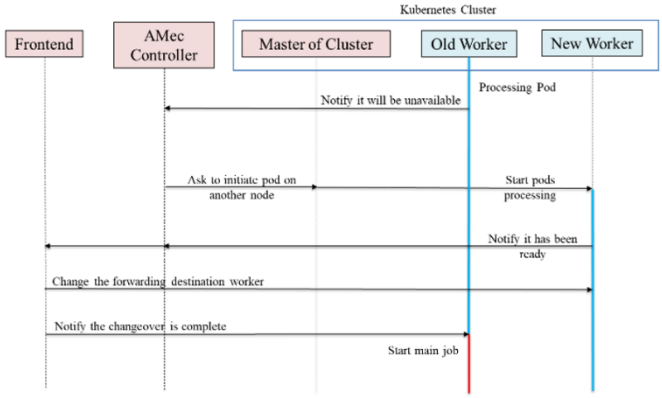

Fig. 6 System sequence (When workers become unavailable)

IV. Simulation and Results

A. Simulation conditions

We have performed simulations in order to show that AMec can reduce the number of edge servers.

First, three types of applications consisting of multiple Pods are generated (Table 1), and then they are assigned to nodes in the cluster consisting of network devices and edge servers in a carrier network of each prefectural unit (Table 2).

Application B is an application that has a long execution time but can be executed on a low-performance CPU, Application C is an application that has a short execution time but requires execution on a high-performance CPU, and Application A is an application in between B and C but consists of one more Pod than them.

To calculate the number of each device in Table 2, we assumed that a carrier network in each prefecture consists of 100 buildings [10]. Assuming that each building serves 5000 users and each OLT can accommodate 500 users, there will be one router, one OXC, and about 10 OLTs in each building.

The start of the first unavailable time of OLT, Router, and OXC varies from node to node, and after that, the unavailable time occurs for 1 minute every 30 minutes.

TABLE I. Application configuration

	Average Duration	Composition Pods	Performance Requirements	Occurrences per minute
A	10 min	A-1	1	20
		A-2	2	
		A-3	3	
B	15 min	B-1	1	20
		B-2	2	
C	5 min	C-1	2	20
		C-2	3	

The following are the requirements for nodes that pods belonging to an application can assigned to.

- The node must be AMec Ready (The node is not in Unavailable time).
- The node has free capacity
- The node's performance value must be greater than or equal to the Pod's performance requirements.

TABLE II. Node configuration

	OLT	Router	OXC	Edge Server
Unavailable time period	30 min			-
Unavailable time duration	1 min			-
Performance	1, 2	2, 3	1	3
Quantity in the cluster	500, 500	50, 50	100	Variable
Capacity (pod) of each node	1			10

Nodes that satisfy the above conditions are referred to here as "suitable nodes". In MEC, nodes consist only of edge servers, so we only need to pay attention to the capacity of nodes. However, since AMec uses in-network CPUs, the other two conditions must also be considered when creating the list of "suitable nodes".

In AMec, our concern is which nodes we place the pods from the list of "suitable nodes" in order to avoid having the nodes become unavailable while the pods running on them.

We compared the following three, changing the number of edge servers.

The first is MEC, which runs solely on edge servers. Since the edge servers are always available as worker nodes for AMec, there is no interruption due to unavailable time.

Second, AMec with a random selection algorithm. It is an algorithm that randomly selects one node from "suitable nodes", which meet the performance requirements of the pod and are available for allocation.

The last one is AMec with the proposed allocation method. The proposed allocation method is as follows.

d_{App} means the duration that the application works on AMec, and r_{Node} means the remaining time of the node until the next unavailable time. Interruption is defined as follows.

$$d_{App} > r_{Node} \tag{1}$$

AMec has past data of each apps as cumulative frequency distribution of application working duration

$$F(t) = P(d_{App} \leq t) \tag{2}$$

In this simulation, the application usage time d_{App} is given by a normal distribution with mean μ and variance σ^2

$$d_{App} \sim N(\mu, \sigma^2) \tag{3}$$

By using them, interruption occurrence rate is presented as follows

$$P_{interruption} = 1 - F(r_{Node}) = 1 - P(d_{App} \leq r_{Node}) \tag{4}$$

Calculate $P_{interruption}$ for all nodes in "suitable nodes" to each application, pick up nodes whose $P_{interruption}$ is below a threshold, and randomly select from the minimum-performance nodes among them. In this case, we set the threshold at 5%. EC2 (Elastic Compute Cloud) at AWS offers 100% service credit if the monthly utilization is less than 95% [11]. The 5% set in this paper was learned from this value.

B. Result

The results are shown in Fig. 7.

First, MEC requires 150 ~ 160 edge servers to accommodate the generated Apps (to achieve 100 % allocation success rate).

On the other hand, AMec (Random) already boasts 100 % allocation rate with only 60 edge servers, but it is not practical due to the high interruption rate.

AMec with the proposed method (with a threshold of 5 %) can achieve an allocation rate of 100 % with 50 ~ 60 edge servers, and the interruption rate is 4 ~ 6 % regardless of the number of edge servers.

Thus, AMec with the proposed method can reduce the number of edge servers by 62.5 % ~ 66.7 % compared to MEC under the condition that some interruption can be tolerated.

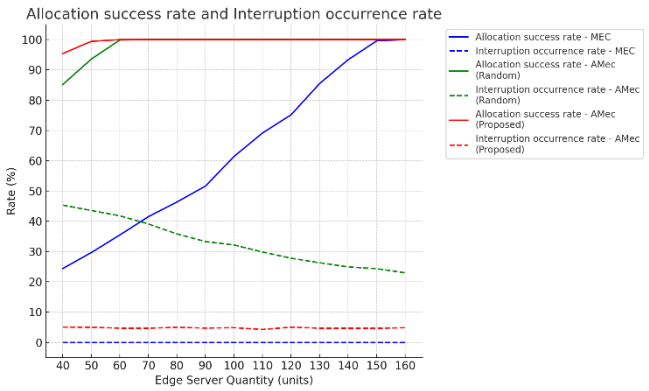

Fig. 7 Result of MEC vs. AMec (with two types of allocation methods)

V. Conclusion

Although AMec was expected to reduce the number of edge servers, there were concerns about frequent interruption and pod's relocation due to executing their main job such as information gathering and routing reconfiguration, unlike MEC.

To avoid the pod's relocation, we proposed an allocation method that calculates the interruption occurrence rate when each pod of the application is placed in each node by using the usage duration characteristics of the application and allocates the pods in a way that the value is below a threshold.

As a result, under conditions where a certain level of interruption can be acceptable, it is confirmed that AMec with the proposed method can reduce the number of edge servers by 62.5 % ~ 66.7 % compared to MEC.

This time, we were able to demonstrate that the AMec can reduce the number of edge servers compared to the MEC, even with a relatively simple algorithm that randomly selects from nodes where the interruption occurrence rate is below a certain threshold. However, it is hard to claim that the Pod placement is strictly optimal.

As future work, we are considering an allocation algorithm that solves an optimization problem aimed at maximizing the allocation success rate and minimizing the interruption occurrence rate for all apps generated every minute. It is expected to achieve better allocation success rates and lower interruption occurrence rates.

Moreover, we have presently defined the application's working duration using a normal distribution. In future research, we aim to demonstrate that AMec can adapt to any kind of distribution for the continuation time, including distributions based on real data.

References

[1] Q. -V. Pham et al., "A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art," in IEEE Access, vol. 8, pp. 116974-117017, 2020, doi: 10.1109/ACCESS.2020.3001277.

[2] P. Gunning, "Bare-Metal Compute, Storage and Networking in Metro Optical Access," 2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC), Fukuoka, Japan, 2019, pp. 1-3.

[3] "Edgecore Wedge100BF-32QS," Apresia Systems, 2023, Accessed: Sep. 27, 2023. [Online]. Available: https://www.apresia.jp/products/whitebox/edgecore/wedge100bf-32qs.html

[4] "F220 | Fitelnet," Furukawa Electric, 2023, Accessed: Sep. 27, 2023. [Online]. Available: https://www.furukawa.co.jp/fitelnet/product/f220/index.html

[5] S. Okamoto, K. Sugiura, M. Murakami, N. Yamanaka, "Proposal of Block-stream as a Service-based Access-Metro Edge Computing Technologies", Program of the 2020 IEICE Society Conference, 2020.

[6] K. Muramatsu, M. Murakami, Y. Uematsu, S. Okamoto, N. Yamanaka, "Dynamic task assignment experiment for container-based in-network heterogeneous distributed MEC environment", iPOP (IP/ IoT & Processing + Optical Network), May 2023

[7] K. Muramatsu, M. Murakami, Y. Uematsu, S. Okamoto, N. Yamanaka, "Experiments of node's automatic participation and collaboration with external services on AMec (Access-Metro Edge Computing)," International Conference on Emerging Technologies for Communications, Nov. 2023, P1-13.

[8] P. Martinez-Julia, V. P. Kafle and H. Harai, "Exploiting External Events for Resource Adaptation in Virtual Computer and Network Systems," in IEEE Transactions on Network and Service Management, vol. 15, no. 2, pp. 555-566, June 2018.

[9] NICT, "JGN: High Speed R&D Network Testbed," Accessed: June 28, 2023. [Online]. Available: https://testbed.nict.go.jp/jgn/

[10] Y.Uematsu et al., "Future Nationwide Optical Network Architecture for Higher Availability and Operability Using Transport SDN Technologies," IEICE Transactions on Communications, 2018, vol. E101.B, no. 2, pp. 462-475, Feb. 2018.

[11] Amazon Web Services, Inc., "Amazon Simple Workflow Service Level Agreement," 2022, Accessed: Nov. 27, 2023. [Online]. Available: https://aws.amazon.com/jp/swf/sla/.