Distributed processing framework for cooperative service among edge devices

Airi Nakamura
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
airi-1@pluslab.org

Katsuhiro Naito
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
naito@pluslab.org

Takaya Yamazato
Nagoya University
Nagoya, Aichi 464-8601, Japan
yamazato@nagoya-u.jp

***Abstract*—This paper proposes a new distributed processing framework to realize cooperative service among edge devices. The proposed framework provides service developers with an execution environment on the edge device. Since it provides abstracted sensor values and camera images, service developers can easily design their service without the specification of peripheral devices such as sensors, cameras, etc. Additionally, our framework performs reliable automatic recognition among neighbor edge devices. Therefore, the service developers can realize distributed service on neighbor edge devices according to the recognition information. This paper implements the proposed framework and creates a person detection service as the demonstration. The experimental results show the demonstration service works on the embedded Linux board.**

***Index Terms*—Internet of Things, Distributed service, Edge processing system, Flexible service deployments**

I. Introduction

The recent spread of Internet of Things (IoT) devices has been noted as a means of realizing the concept of smart cities [1]–[3]. Conventional systems typically use special IoT devices designed for unique services [4]–[6]. Therefore, the systems do not have any flexibility to extend or update the services. Additionally, they use cloud services to process information from IoT devices even if recent devices have enough processing power.

As smart cities service continue to develop, the installation of IoT devices throughout the city will increase the number of devices. As a result, cloud service in the current systems will suffer from the information concentration of many IoT devices. Since the concentration causes a high processing load and delays processing, adequate preprocessing in IoT devices will be an essential function [7], [8]. In addition, creating services for dedicated IoT devices will hinder data sharing among devices and flexible service deployment. Therefore, smart city service requires a new framework for cooperative processing among IoT devices to provide flexible and scalable services [9].

The authors have proposed a new flexible service deployment environment for edge processing systems [10]. The proposed environment uses container technology to realize independent deployment of each service. As a result, any developers can install or update their service on edge devices. On the contrary, distributed smart city service has some challenges in realizing cooperative service among IoT devices in a limited area. Firstly, each IoT device should process services cooperatively. Secondly, cooperative services should share data among IoT devices. Thirdly, IoT devices should recognize each other safely.

This paper proposes a new distributed processing framework to realize cooperative service among edge devices. The proposed framework provides automatic recognition mechanisms among edge devices with digital certificates. It also provides cooperative environments to realize a service on neighbor edge devices. Since the proposed framework is a comprehensive system of the proposed environment, it also supports flexible service deployments. As a result, service developers can distribute their service to many edge devices efficiently and realize cooperative service on the edge devices. The prototype implementation shows the sample person detection service to process image data cooperatively among neighbor edge devices.

II. Proposed System

Figure 1 shows the proposed system. The proposed system is an open architecture for providing multiple edge-device services. Therefore, it assumes three types of collaborators: the service developer, the device owner, and the cloud administrator. The proposed system consists of an edge framework and cloud services.

The edge framework provides processing environments for multiple services. Since the environment supports container technology to process each service, each service can work simultaneously and independently. Some services may require neighbor information based on sensors. The framework handles equipped sensors, converts actual sensor values to abstracted information such as temperature, and shares the information with services. As a result, developers can efficiently process neighbor information based on the abstracted name. The benefit of employing a containerized virtual environment is easy deployment and update of services.

The cloud service is the manager of each edge device. It provides an authentication service for the service developer, the device owner, and the cloud administrator. It also provides a digital certificate to each edge device to ensure its legitimacy. Since service developer can upload their service containers to the cloud, the edge device can easily download the containers from the cloud and process them on its edge device. Therefore,

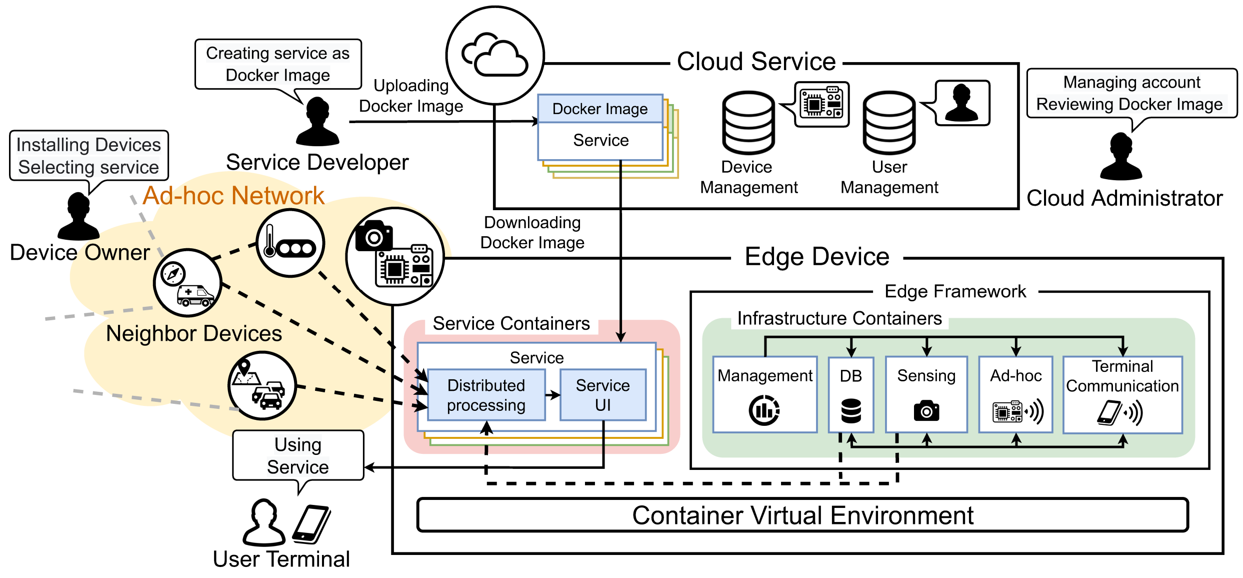


Fig. 1. System model of distributed processing framework

the cloud service does not collect and process data from edge devices like conventional systems. As a result, each edge device can perform distributed processing even if it may be offline.

A. *User types*

- Service Developer
  The service developers are responsible for creating the service container. Since the proposed framework provides a container execution environment to realize their service, they can develop their service as a container. Additionally, the framework also provides neighbor device information to them. Therefore, their service container can perform distributed processing by exchanging information among neighbor devices. Some service containers may access sensors on the devices. Typically, each sensor is available for exclusive use. Therefore, the framework accesses the sensors exclusively and provides sensor information to multiple containers. Since the container services require abstracted sensor information such as temperature, humidity, illuminance, etc., the framework converts abstracted values from actual sensor values. As a result, the service developers can create their own distributed service among devices without considering the details of devices and sensor specifications. Additionally, they can distribute their container service to many devices by uploading to the cloud service because each device can obtain the container image from the cloud service.
- Cloud Administrator
  The cloud administrators are the manager of the whole system. They register and manage the authentication information for each user type: service developers and device owners. Therefore, the cloud service authenticates whole users to prevent unauthorized users from accessing the system. Additionally, they check service developers' uploaded service containers to certify them without malicious service. As a result, they distribute the certified service container to devices according to the request of the service.
- Device Owner
  The device owners are owners of edge devices. Typically, they buy edge devices and install them into their facilities and access the cloud through the edge devices to join the proposed system. The devices assume that supported sensors or camera devices are available. Additionally, they can select a container service from the service container lists on the cloud. As a result, they can provide dedicated service to users in the vicinity of the device even if they cannot develop a service.

B. *Cloud service*

The cloud service has some sub-services for the user, container, and device management.

- User management
  The user management sub-service handles actual user data to manage their profile and authentication. Since service developers manage their container service on the cloud, they also control the administrative privileges of each container service. Device owners also register their edge devices to the cloud. Therefore, the administrative privileges of each device also follow the user management information.
- Container management
  Each service container has its profile for controlling parameters and specifications. For example, some service container requires some sensor values. Therefore, the container management sub-service should manage the required sensor lists for each service container. As a result, owner users can select an available service container for their edge device by checking the list.

- Device management
  The device management sub-service manages the profile of each edge device. Since device owners select a container service from the available service list, it manages the container service list for each edge device. Each edge device downloads the selected container image according to the container service list and starts the container service. In addition, they can re-download the container image when developer users update the container image.

C. *Edge Framework*

The edge framework consists of two types of containers: additional service containers and infrastructure containers. Additional service containers are service containers developed by service developers. The edge device downloads them from the cloud service. Since service developers can update the service container image, the edge device also updates the container as necessary. Device owners can reselect the service container to stop. In this case, it deletes the container image on the edge device.

Infrastructure containers provide the proposed framework functions. The functions consist of five services: terminal, database, ad-hoc, sensing, and management sub-services. Each sub-service performs as an independent container. Therefore, the proposed framework can update each sub-service independently.

The terminal sub-service provides an access interface to users' equipment such as smartphones, pads, etc. The database sub-service provides a database function for users and service containers. Therefore, users can select a service from the available service list on the edge device, and service containers can obtain framework information. The ad-hoc sub-service provides a neighbor discovery function for neighbor edge devices. It also confirms its legitimacy with neighbor edge devices using mutual authentication of digital certificates. Therefore, service containers can use reliable communication among neighbor devices. The sensing container provides abstracted sensor information. Typically, an application can access actual sensors exclusively. Therefore, multiple applications cannot access the sensors simultaneously. The sub-service access whole sensors instead of all service containers and converts the actual sensor values to abstracted values such as temperature. As a result, service developers can design their service to access sensor information by abstracted names. The management sub-service handles the computation resource of the edge device. It measures system resource conditions and controls each service container according to user and neighbor containers' requests. Since the infrastructure containers provide the core service of the proposed framework, device owners cannot delete these containers. Details of the containers are shown below.

- Database Container
  The database container is one of the infrastructure containers for storing requested services and surrounding connection information. It manages the three types of tables: device information, neighbor device information, and request buffer shown in Table I. The device information table stores the list of available service containers on the device. Since each service works as a container, they have their IP address to access them. The neighbor device information table stores the list of neighbor edge devices recognized by the ad-hoc sub-service. Each service container can access its container on neighbor devices based on the information to perform distributed processing. The request buffer table handles the service providing information to users and service requesting information from neighbor service containers. According to the table information, the framework can manage the service container.
- Terminal Communication Container
  The terminal communication container is an interface for users around the edge device. Since the edge device provides Wi-Fi access points for neighbor terminals, users can access the service interface provided by the terminal communication container. Users can obtain the available service lists on the device and request a service. The terminal communication container stores the requested information in the table on the database container.
- Ad-hoc Container
  The ad-hoc container guarantees reliable communication to neighbor devices for the service. Since the edge device has a unique Wi-Fi interface for ad-hoc mode communication among neighbor devices, the ad-hoc container

TABLE I
DATABASE TABLE CONFIGURATION

Device Information TABLE	
Service Name	Name of services that can be provided
Container IP	Container IP of the service container
Service Type	Conditions of Service Operation
Neighbor Device Information TABLE	
Service Name	Neighbor Service Name
Device IP	The IP of the neighbor device
Container Port	Neighbor Service Container Port
Request Buffer TABLE	
Request ID	Priority of services
Request Service	Name of requested service
Requester Type	Identification of the requesting device
Requester IP	The IP of the requested device

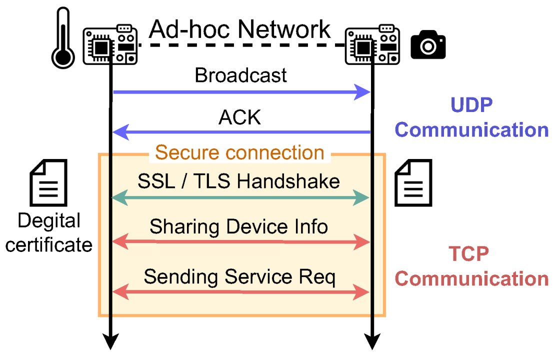


Fig. 2. Mutual recognition process among edge devices

can scan the neighbor devices. It also exchanges digital certificates to realize secure, reliable communication by Secure Sockets Layer (SSL)/Transport Layer Security (TLS) when it finds a neighbor device.
Figure 2 shows the packet sequence to establish reliable communication among neighbor devices. The ad-hoc container periodically transmits a hello message to the broadcast address. The neighbor device responds with an acknowledgment message when it receives the hello message. Then, it starts the authentication process for the correspondent device based on SSL/TLS authentication. Since each edge device has a digital certificate published from the cloud service, they can achieve mutual authentication with the correspondent device. It also stores the information of the certificated devices in the neighbor device information table.

- Management Container
The management container is the core control function of the proposed framework. It downloads the service container image from the cloud when a device owner starts a newly available service. It also updates the service container image when a service developer updates the image.
When the resource shortage occurs due to multiple service processing, it determines the processing priority according to the type of service container. As a result, the prioritized service container can work first. In addition, the management container continuously measures the computation resource on the edge device and releases the resource of services when the services are not in use.
- Sensing Container
The sensing container access sensors and camera devices instead of service containers because exclusive access is only available for real neighbor devices. As a result, some services can share a sensor or camera device to obtain neighbor information. In addition, typical service developers tend to use abstracted information instead of real sensor values. Therefore, the sensing container can convert the real sensor values to abstracted ones such as temperature, humidity, illuminance, air pressure, etc. As a result, service developers can easily design their service with the abstracted name information without knowing the actual specification of neighbor devices.
- Service Container
A service container is a type of additional container downloaded from a cloud service. It provides specific services such as human tracking, human detection, environment measurements, etc. Since the service container works independently on the proposed framework, service developers can concentrate on their service design even if multiple service containers work on the same edge device. Additionally, they can use abstracted sensor values and camera images from the sensing containers. Since the cloud service distributes the service container image instead of service developers, it is easy to publish their service to many edge devices.

TABLE II
EXECUTION ENVIRONMENT

Edge Device	Raspberry Pi 4 Model B (Raspbian 10.7)
Camera	SANWA Webcam CMS-V59BK
User Terminal	iPod touch (iOS version 12.5.1)
External Wifi Module	TP-LINK T4U archer

III. IMPLEMENTATION

We have implemented a prototype system of the proposed service framework. The prototype system supports a service container that provides a person detection service as the demonstration. The prototype system uses two devices equipped with cameras and a user terminal. The prototype system consists of multiple containers inside the devices, which communicate with each other. The service container indicates to the user terminal when it has detected a person.

We used Raspberry Pi 4 as the edge device and iPod touch as the user terminal. Raspberry Pi 4 equips SANWA Webcam CMS-V59BK as the camera device on the edge device. Docker application works on the edge device as the container virtualization technology. Since the edge device provides two types of wireless interfaces for distributed processing and service delivery, it installs two wireless LAN adapters. One adapter provides an ad-hoc network connection to the neighbor device, and the other provides an access point function to user terminals. Table II shows the implementation environment.

The demonstration service realizes a distributed processing among two edge devices by the service containers to detect a person in a camera image. Since the user terminal accesses Device A, Device A provides the user interface of the demonstration service through the access point function. It also works as the person detection function. Device B works as the only person detection function and collaborates with Device A to notify the person detection to the user terminal.

Since the proposed framework requires the infrastructure containers, both devices preload these containers: management, database, ad-hoc, sensing, and terminal communication containers. Additionally, it also downloads the service container from the cloud service.

Both devices start to sense neighbor devices on the ad-hoc network when they start up. They exchange digital certificates through SSL/TLS communication when they find each other. Then, they exchange the information about the service containers and store it in the database. They also start up the access point function for neighbor user terminals.

Figure 3 shows the diagram of service operating procedures. The operating procedures are described below.

1) The user's terminal connects to the access point of Device A.
2) The user terminal sends the user's IP address and service name as a service request to the terminal communication container of Device A.
3) The terminal communication container registers the service request information to the database container.

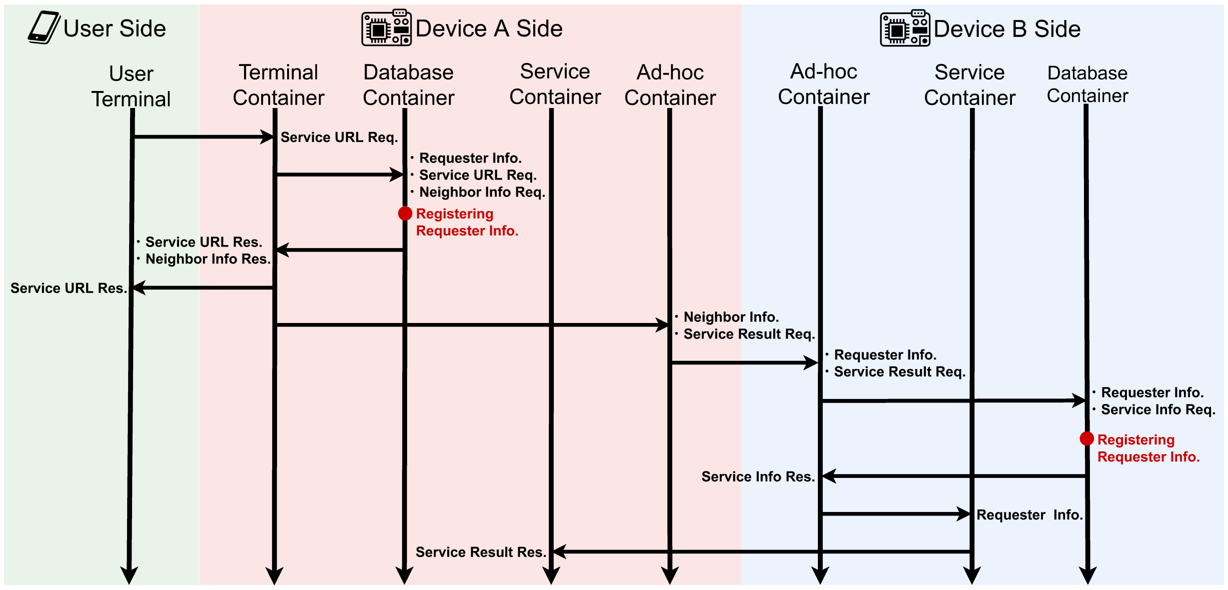


Fig. 3. Service operating procedures of the demonstration

4) The database container returns information about neighbor devices and server Uniform Resource Locator (URL) to the terminal.
5) The terminal communication container sends the server URL to the user terminal.
6) The terminal communication container sends the neighbor device information and the requested service name to the ad-hoc container.
7) The ad-hoc container makes a service request to the ad-hoc container of Device B through the ad-hoc network based on the neighbor device information. It also sends the service name and the port number of the service container of Device A.
8) The ad-hoc container of Device B registers the received service request information in its database container.
9) The database container returns the service container information of the requested service to the ad-hoc container.
10) The ad-hoc container passes the IP address of Device A and the port number of the service container to the service container.
11) The service container on Device B sends the results of the person detection process to the service container on Device A.
12) The service container on Device A receives the results of the service processing from Device B. It merges it with the results of the person detection service on Device A and notifies the results through the user interface.

Figure 4 shows the block diagram of the demonstration service container. The demonstration container consists of two functions: person detection and user interface.

The person detection function acquires camera images from the camera sensing container in real-time and performs person detection. It also notifies the detection results to the user interface function when it detects a person.

The user interface function has a web service for user terminals. Since the docker configures port forwarding to the web service, user terminals can access it through the access point.

TABLE III
CONTAINER IMAGE STORAGE RESOURCES

Additional Service Containers	
Service Container	4,733 MB
Infrastructure Containers	
Terminal Container	680 MB
Database Container	1,200 MB
Ad-hoc Container	680 MB
Camera Sensing Container	4,020 MB

IV. Performance Evaluation

We have evaluated the performance of the prototype system. Figure 5 shows the example of the screen image of the demonstration service. The web service notifies the detected person image to users when the person detection function detects a person. The users can check the result of the person detection service on Devices A and B with their user terminal.

We have also measured the mutual detection and distributed processing processes ten times and taken averages of them. In the mutual detection process, the measurement starts when Device A broadcasts a hello message to the ad-hoc network. It ends when Devices A and B send and receive the device information. The average measured time for the mutual detection process was 111 ms. Therefore, each edge device can recognize the other quickly, even if they are moving. The measurement starts with receiving a service request from a

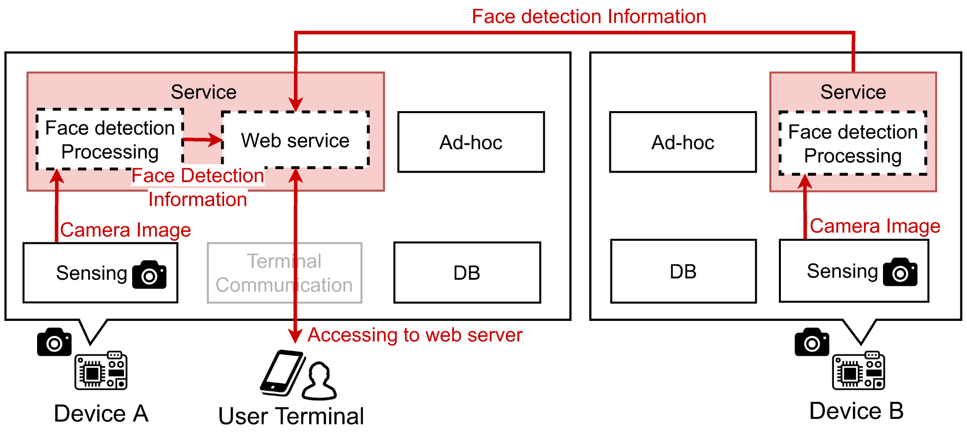
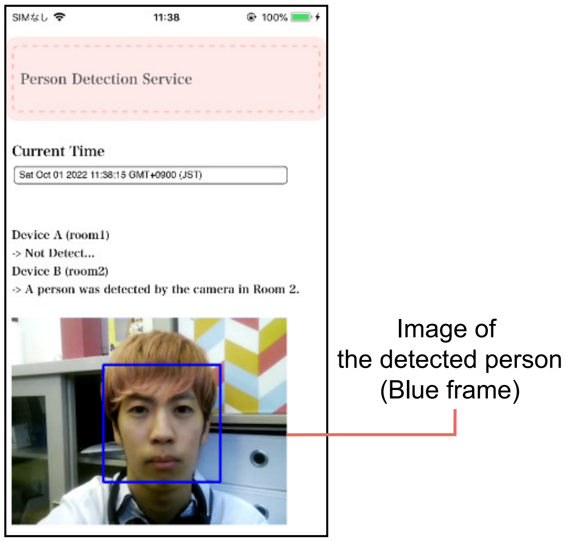


Fig. 4. Block diagram of the demonstration service


SIMなし 11:38 100%

Person Detection Service

Current Time

Sat Oct 01 2022 11:38:15 GMT+0900 (JST)

Device A (room1)
-> Not Detect...
Device B (room2)
-> A person was detected by the camera in Room 2.

Fig. 5. Demonstration view of the service

user near Device A. It ends when Device B starts processing to collaborate with Device A. The average measured time for the device coordination process was 7.36 ms. Additionally, we have measured the storage usage of each container. Table III shows the storage resources used by the images of each container.

V. Conclusion

This paper has proposed a new distributed processing framework to realize cooperative service among edge devices. Since the proposed framework provides reliable device recognition mechanisms, the service can easily collaborate among neighbor devices. The framework also provides the service's distribution function to edge devices from the cloud. As a result, service developers can distribute their service to many devices efficiently and realize cooperative service on edge devices. The prototype implementation showed the demonstration of the person detection service. The experimental evalufations showed that the proposed framework works adequately on the Raspberry Pi 4, a well-known embedded Linux board.

Acknowledgment

This work is supported in part by the Collaborative Research Project with Ultimatrust Co., Ltd., Grant-in-Aid for Scientific Research (C)(21K11877), the Japan Society for the Promotion of Science (JSPS), and the Collaborative Research Project of the Research Institute of Electrical Communication, Tohoku University.

References

[1] E. Bertino, K.-K. R. Choo, D. Georgakopolous, and S. Nepal, "Internet of Things (IoT): Smart and Secure Service Delivery," ACM Trans. Internet Technology, vol. 16, no. 4, pp. 1–7, 2016.
[2] A. Harit, A. Ezzati, and R. Elharti, "Internet of things security: challenges and perspectives," The Second International Conference on Internet of things, Data and Cloud Computing, pp. 1–8, 2017.
[3] Y. Fathy, P. Barnaghi, and R. Tafazolli, "Large-Scale Indexing, Discovery, and Ranking for the Internet of Things (IoT)," ACM Comput. Surv, vol. 51, no. 2, pp. 1–53, 2018.
[4] S. Myneni, G. Agrawal, Y. Deng, A. Chowdhary, N. Vadnere, and D. Huang, "SCVS: On AI and Edge Clouds Enabled Privacy-preserved Smart-city Video Surveillance Services," ACM Trans. Internet Things, vol. 3, no. 4, pp. 1–26, Sep. 2022.
[5] M. Bhatia, S. Kaur, and S. K. Sood, "IoT-Inspired Smart Toilet System for Home-Based Urine Infection Prediction," ACM Trans. Comput. Healthcare, vol. 1, no. 3, pp. 1-–25, may 2020.
[6] F. Liu, Y. Guo, Z. Cai, N. Xiao, and Z. Zhao, "Edge-enabled Disaster Rescue: A Case Study of Searching for Missing People," ACM Trans. Intell. Syst. Technol, vol. 10, no. 6, pp. 1–21, 2019.
[7] M. Abdallah, C. Griwodz, K. T. Chen, G. Simon, P. C. Wang, and C. H. Hsu, "Delay-Sensitive Video Computing in the Cloud: A Survey," ACM Trans. Multimedia Comput. Commun. Appl, vol. 14, no. 3, pp. 1–29, 2018.
[8] B. Zhenbo, Z. Shiyou, P. Hongjun, W. Yuanhong, and Y. Hua, "A Survey of Preprocessing Methods for Marine Ship Target Detection Based on Video Surveillance ," 2021 7th International Conference on Computing and Artificial Intelligence (ICCAI 2021), pp. 1-–7, 2021.
[9] A. Giri, S. Dutta, S. Neogy, K. Dahal, and Z. Pervez, "Internet of things (IoT): A survey on architecture, enabling technologies, applications and challenges," The 1st International Conference on Internet of Things and Machine Learning, Association for Computing Machinery, pp. 1–12, 2017.
[10] A. Nakamura, T. Yamazato, and K. Naito, "Proposal of flexible service deployment environment for the edge processing system," IEICE Communications Express, vol. 11-B, no. 8, pp. 503–508, 2022.