SUMMARY Recently, edge servers located closer than the cloud have become expected for the purpose of processing the large amount of sensor data generated by IoT devices such as robots. Research has been proposed to improve responsiveness as a cache server by applying KVS (Key-Value Store) to the edge as a method for obtaining high responsiveness. Above all, a hybrid-KVS server that uses both DRAM and NVMM (Non-Volatile Main Memory) devices is expected to achieve both responsiveness and reliability. However, its effectiveness has not been verified in actual applications, and its effectiveness is not clear in terms of its relationship with the cloud. The purpose of this study is to evaluate the effectiveness of hybrid-KVS servers using the SLAM (Simultaneous Localization and Mapping), which is a widely used application in robots and autonomous driving. It is appropriate for applying an edge server and requires responsiveness and reliability. SLAM is generally implemented on ROS (Robot Operating System) middleware and communicates with the server through ROS middleware. However, if we use hybrid-KVS on the edge with the SLAM and ROS, the communication could not be achieved since the message objects are different from the format expected by KVS. Therefore, in this research, we propose a mechanism to apply the ROS memory object to hybrid-KVS by designing and implementing the data serialization function to extend ROS. As a result of the proposed fogcached-ros and evaluation, we confirm the effectiveness of low API overhead, support for data used by SLAM, and low latency difference between the edge and cloud.

key words: ROS, SLAM, KVS, memcached, NVM, edge computing

1. Introduction

In recent years, autonomous mobile robots that operate in warehouses and facilities have become widespread for the purpose of reducing labor. These robots are expected to be multiple and autonomous. For example, a robot that measures road surface properties [1] achieves its purpose more accurately and regularly at a lower cost than humans by calculating the position and inclination angle of objects from the sky with multiple drones. In addition, the security robot [2] operates multiple robots at the same time to detect suspicious persons, detect abnormalities in the surrounding area, and report. In addition to these examples, there are autonomous traveling delivery robots [3] and systems in which multiple small robots work together to search and rescue victims and missing persons [4].

Many of these robots perform actuation by sensing a large amount of environmental information and analyzing the data in order to realize autonomous driving. In the past, when many robots were operating standalone, the sensing data used to grasp physical information was not saved and discarded after the control calculation due to computer resource constraints. However, in recent years, services such as sending data taken by robots to the cloud by drones connected to the network [5] have become widespread, and sensing data is transferred and stored in cloud services. In data transmission from a large number of robots, processing is concentrated in the cloud, and there are problems such as overload of the cloud server and network delay due to wire-less communication to the cloud at a long distance.

On the other hand, in recent years, research on edge computing for handling a large amount of data at a shorter distance than the cloud has been proposed [6]–[8]. In edge computing, a design that emphasizes responsiveness to many sensor nodes has been proposed for the purpose of real-time response at a short distance to sensor nodes existing on the edge side close to the device [8]. As one of the responsiveness improvement technologies, a mechanism has been proposed in which a cache is deployed at the edge to effectively cache data up to the cloud on an edge server [9].

Edge servers operating at short distances are required to have high responsiveness and reliability. As a method for obtaining high responsiveness, research has been proposed to improve responsiveness by applying KVS (Key-Value Store) to the edge and applying it as a cache server [10]. Ozawa et al. proposed the "fogcached" mechanism to realize both reliability and responsiveness by extending KVS and switching between DRAM and NVMM (Non-Volatile Main Memory) according to the number of accesses. The hybrid-KVS named ‘fogcached’ is based on ‘memcached [11]’, which is a typical implementation of key-value store and has been extended to NVMM in order to improve data reliability and responsiveness. It showed effectiveness as the simulation base [10], however, its effectiveness has only been evaluated using simulation, and its effectiveness is not clear in actual applications.

The purpose of this study is to evaluate the effectiveness of the hybrid-KVS server using SLAM (Simultaneous Localization and Mapping) [12], which is a widely used application in robots and autonomous driving. It is appropriate for applying edge server and requires high edge performance and reliability. SLAM is generally implemented on
ROS (Robot Operating System) [13] middleware and communicates with the server through ROS middleware. However, if we use hybrid-KVS on the edge with SLAM and ROS, communication could not be achieved since the message objects are different from the format expected by KVS.

Therefore, in this research, we propose a fogcached, which makes possible to use ROS application SLAM by providing the APIs that connect the fogcached automatically. There is no implementation of hybrid-KVS (fogcached) utilize both of DRAM and NVMM supporting the ROS SLAM application other than our proposed implementation.

We summarize the contributions the following points.

1. Propose the fogcached-ros that makes possible to use ROS implemented SLAM application for the hybrid KVS (fogcached) environment.
2. Evaluate the fogcached-ros and fogcached (hybrid KVS) under the edge-cloud environment using the actual application SLAM.

With this proposal, we will make the fogcached (edge server) delivering reliability and responsiveness that can be put into practical use in a real application. By designing and implementing the API of the ROS middleware in the lower layer of SLAM, past data can be automatically saved in hybrid-KVS (fogcached) server that contains the NVMM through our proposed middleware without modifying SLAM application. Moreover, we evaluate it on the actual fogcached DRAM and DCPM environments. DCPM (Intel Optane Data Center Persistent Memory) [14] is an NVMM hardware implementation. As a result of the proposed fogcached-ros and evaluation on the environment, we confirm the effectiveness of low API overhead, support for data used by SLAM, and a low latency difference between edge and cloud. Moreover, we evaluate our proposed edge server environment that is built with the fogcached-ros and fogcached, using the SLAM application, our total system (fogcached and fogcached-ros) are evaluated.

The organization of this paper is as follows: firstly, we describe prior research in this subject in Sect. 2, our Proposal in Sect. 3, Preliminary Experiment Section 4, API Evaluation Section 5, Cloud Connection Evaluation Section 6, a Summary and Future Issues are described in Sect. 7.

2. Related Work

In recent years, non-volatile memory (NVM) has been attracting attention as a means of improving performance and reliability. Intel announced DCPM (Intel Optane Data Center Persistent Memory) in 2019 [14]. DCPM is a DDR4 standard memory with a latency that is about four times that of DRAM [15].

Wu et al. proposed NVMcached [16]. NVMcached is an NVM-based key-value cache. Considering that the conventional technology for maintaining consistency such as copy-on-write and journaling is heavy processing in applications that require high-speed processing such as KVS, we proposed a method that considers the write endurance of NVM as NVMcached. Specifically, the list of objects (metadata) is saved in DRAM, and the objects are saved in NVM. Objects that are deleted or updated in a short time in consideration of write resistance are saved in DRAM. After a crash, the list saved in DRAM disappears, however, the objects saved in NVM reflect the access history from the client, and the list is reconstructed using that as a clue. We aimed to achieve both performance and reliability by guaranteeing object persistence with NVM while accelerating metadata processing with DRAM. In this study, they improved the system throughput of real write-intensive workloads by up to 2.8 times compared to non-volatile memcached [17]. However, there is a problem that data does not move between DRAM and NVM and does not move to NVM when the importance of stored data increases.

Edge computing studies include those that focus on responsiveness [9], [18]–[21] and those that focus on reliability [22]. Responsiveness is enhanced by improving the caching algorithm of the objects in the data cache. Reliability research can be achieved by using non-volatile devices or by satisfying data reliability by algorithms such as optimization. However, these studies focus on either one, and there are few studies that achieve both responsiveness and reliability.

Ozawa et al. proposed to use NVMM (Non-Volatile Main Memory) to hold a large amount of data on an edge server and develop an In-Memory Key-Value Store (KVS) on a hybrid main memory consisting of DRAM and NVMM [10]. The evaluation also showed that the responsiveness of the edge server was improved. They used DCPM as the NVMM and made it accessible from memcached [17] as a KVS. They named the implementation ‘fogcached’ - an extension of memcached - that has a structure connecting DRAM and DCPM. Specifically, a least recently used algorithm (LRU) with the same structure as the LRU for DRAM inside fogcached was created on DCPM, and the movement of memory objects between the two LRU structures (Dual-LRU) was connected by a pipeline called MOVE. The memory object automatically moves to DCPM when the access frequency becomes low, achieving reliability while maintaining responsiveness. Compared to extstore, which is an existing implementation of memcached, the latency of the entire system is improved by about 40% and the throughput is improved by about 19%. However, reliability has not been discussed, and there is a problem that is limited to the use of non-volatile memory.

3. Proposal

3.1 Overview of the Proposal

The issues discussed so far can be summarized as follows: first, issue (A) is that the evaluation of the actual application is insufficient, and issue (B) cooperation with the cloud in consideration of reliability has not been implemented nor evaluated. To solve the issues (A) and (B), the purpose of
this research is to provide a mechanism to achieve both responsiveness and reliability in the edge server when using a robot application, and to realize evaluation in an actual application. In order to achieve this, we propose a middleware that makes SLAM compatible with the hybrid main memory KVS server. Since there are no implementations that link SLAM and the KVS server, we propose the novel implementation fogcached-ros in consideration of robots for problem (A) as proposal (1). In addition, as proposal (2), we propose the design and implementation of a system linked with the cloud, to solve the problem (B). In this research, by designing and implementing the API on ROS middleware, which is a lower layer of SLAM, past data can be automatically saved in the NVMM on the cache server through the middleware without modifying SLAM. With this proposal, we will have an edge server that delivers reliability and responsiveness that can be put into practical use in a real application.

In the following section, we will firstly describe the basic technologies in the robot application SLAM and ROS then describe the proposed system.

3.2 SLAM

SLAM (Simultaneous Localization and Mapping) [12] is a general term for technologies that collect environmental information from sensors in moving objects such as robots and perform self-position estimation and environmental mapping at the same time by analyzing the data. By utilizing SLAM, it is possible to realize autonomous movement of a wide range of moving objects such as automatic driving as well as robots. In this study, we use ‘gmapping’ [23], which is widely used in the SLAM algorithm that assumes two-dimensional lidar as a sensor input. In particular, it is one of the most widely used SLAM methods for land-based mobile robots. Moreover, gmapping, although a 2D SLAM, it is suitable for map generation in a large-scale environment due to the advantage of explicit loop capture [24], and has been used in the Tsukuba Challenge in Japan [25]. SLAM handles a large variety of sensor data and as such requires a lot of resources. We consider these features are suitable to evaluate the hybrid KVS server that is expected to be useful for the practical application. Many of these SLAM applications are implemented on ROS.

3.3 ROS (Robot Operating System)

In this study, we use the ROS (Robot Operating System) [12], [13]. ROS is an open-source middleware library and tools that support the creation of robot applications. In this section, we briefly explain about the ROS. Firstly, ROS is composed of nodes. A node is equivalent to a UNIX process. A robot control system usually consists of multiple processes. On ROS, the process is configured as a node. Next, a master node is a special process (node) that manages the entire ROS communication and knows all communication targets. Communication between nodes is performed in the data format message. One message is a simple data structure consisting of type fields. A topic is a name to distinguish the contents of the message. In particular, publisher and subscriber are the names of roles of the node, which are divided into the role of writing data to topic (publisher) and the role of subscribe to read the written topic data (subscriber), respectively. The ROS application performs asynchronous publisher and subscriber communication by assigning the roles and to each node. Through publisher and subscriber communication, ROS implements the service that generates events.

3.4 Proposed System

As we described in the Sect. 1, the edge servers operating at short distances are required to have high responsiveness and reliability. Figure 1 shows the assumed overall view of the edge-cloud server system for the robotics service.

Since we are assuming the robotics service environment here, the client robot needs to store all of the sensing data of its surroundings. Since the robot has only limited storage available, they send all of the data to the deployed edge server as long as the radio is connected. The edge server is expected to meet both the high responsiveness and data reliability requirements of KVS.

To satisfy these requirements, we propose an architecture in which the robot is configured as a client and is connected to the edge with KVS fogcached server storing the data hierarchically with the high reliability and low latency for practical robotics applications (Fig. 2).

The client-side application is configured using SLAM and ROS as practical application and middleware. The robot hardware was TurtleBot3 [26], and SLAM gmapping was run as a ROS application. For the edge server, fogcached [10], which is an implementation of the hybrid main memory KVS, was run on the hardware of the hybrid main memory DRAM and DCPM. We believe that the hybrid main memory KVS system can satisfy both of the objectives of achieving high data responsiveness and reliability for the data. To realize edge services, we proposed a middleware, fogcached-ros, which receives the data sent from the SLAM + ROS client on the edge server and forwards them to fogcached at the backend. If there is a cache miss (no data in fogcached) when fetching data from
the edge server from the client, the data can be retrieved from the cloud. The cloud communicates only with the edge server and stores the data sent from fogcached-ros. All of these communications can be used in a unified manner using the ROS component interface.

3.5 Practical Benefits for the Multiple Robot Service

In the proposed system, it is assumed that all the sensor data collected by the robot is saved in the server. There are two practical benefits of saving to the server. (1) It is possible to integrate sensor data of multiple robots, generate higher-order information, and provide services with high utility value. (2) The reliability and permanence of sensor data can be improved.

Regarding (1), many SLAM applications used on ROS can aggregate sensor data collected from different clients at different time and places to create a single map. This service, which has been used in autonomous driving in recent years, has a great advantage that robots with different creators can perform services such as navigation based on the created map [27]. Furthermore, SLAM can perform navigation by combining maps created by sensors measured at different time. Therefore, it is important for the client to aggregate all the data on the server without discarding it. This is difficult to achieve with a stand-alone robot.

Regarding (2), data persistence is important. In recent years, it has become increasingly important to store runtime information and use it later for accountability purposes. Since robots are particularly required to be safe, saving the PWM values of brakes and motors is expected to provide accountability in the event of an accident and countermeasures against problems. ROS can also record the data that flows between nodes and store the data itself. Therefore, it is important to preserve all data.

In addition, in order to store all of this information in the cloud, there are delays during transmission when sensor data is sent sequentially and limits on wireless connection [18]. Hence, the edge server is located in the middle, where improving the responsiveness and guaranteeing the reliability of the data are more important issues. Therefore, we believe that our proposed system, which builds the edge with responsiveness and reliability in mind and realizes collection of sensor data from a robot, has great advantages. Especially when communicating from distributed SLAM to the cloud, communication overload and delay are expected. Since the cache server is located closer, there is an advantage in solving these problems. Furthermore, since fogcached, which is assumed to be a cache server, is a distributed cache server, it is possible to consider using the server for each distributed SLAM.

3.6 Design and Implementation of SLAM and Server Integration

As we described in Sect. 3.3 ROS is middleware in which a communication unit called a node has a role of publisher and subscriber, and asynchronous communication is performed via an intermediate shared memory called a topic (Fig. 3). In the conventional ROS communication model, there is no means to communicate with the KVS server, namely fogcached. Also, since the topic is basically overwritten by a node during communication, only the latest information is saved.

In this research, we designed and implement fogcached-ros that extended middleware of ROS. It provides additional API for automatically communicating with fogcached via the topic, then past data can be automatically saved in NVMM on fogcached. We believe that this will not only be more convenient but also improve data reliability. The fogcached-ros provides nodes called observer_set() and observer_get() as shown in the Fig. 3.

Figure 3 shows the flow in which data is returned to the same topic, and this also allows the data to be output to the different topics. Through this mechanism, all of the sensor data on SLAM that uses the ROS middleware using this topic will send to the fogcached edge server that implemented DRAM/NVMM through the fogcached-ros API and will be stored on the server automatically. It means that reliability can be achieved through this mechanism, since the responsiveness can be improved by storing the data in fogcached and the data can be stored permanently by storing it in NVMM. In addition, communication between fogcached-ros and fogcached is performed by two types of APIs: (1) and (2). This method is not application specific. Fogcached-ros is topic-dependent. Therefore, it can be applied to these applications as long as they support topics such as navigation [27] and opencv [28] as well as SLAM topics. The
3.7 Communication API

We developed the two types of APIs for linking ROS and fogcached (Fig. 4). As shown in Fig. 3, communication with fogcached through the ROS node is performed by (1) setAPI() and (2) getAPI(). The fogcached is a KVS server and needs to generate Key and Value. Both (1) and (2) APIs receive the topic name and ID from ROS through the API (Fig. 3). In that API, the ID is a character string arbitrarily set by the user and is used to identify the data of the same topic. In the API, (1) setAPI also receives a message at the same time, but (2) getAPI message is a reference. Figure 5 shows the processing flow using the API interface.

Both APIs (1) and (2) create a KVS key as a linked character string from the topic and an ID set from the ROS node. The upper limit of the key is set to 50 bytes. The message of setAPI() is converted to a binary character string by using the serialize function of ROS. Set the binary character string generated by serializing message as value. At this time, malloc dynamically allocates the OS heap area according to the topic. In setAPI, these keys and values are stored in fogcached. The getAPI() restores from the binary and returns the restored message. Also, both APIs support not only one type but many topics. Figure 6 shows the API call when it is actually executed in the subscriber callback.

### Table 1: ROS task and the composed environment.

| ROS Task         | OS      | Mem | HDD  | Computer    |
|------------------|---------|-----|------|-------------|
| ROS master       | Ubuntu  | 8GB | 100GB| Intel Core i7-8550U (Windows10 Hyper-V) |
| SLAM gmapping    | 18.04 LTS | 1GB | 32GB | Raspberry Pi 3B+ |
| rosbag           | Ubuntu Mate 18.04 LTS | 1GB | 32GB | Raspberry Pi 3B+ |
| TurtleBot3 bring up | Ubuntu Mate 18.04 LTS | 1GB | 32GB | Raspberry Pi 3B+ |
| teleop (keyboard operation) | Ubuntu Mate 18.04 LTS | 1GB | 32GB | Raspberry Pi 3B+ |

4. Preliminary Experiment

4.1 Overview

The purpose of this experiment is to investigate how much data is used in ROS communication when SLAM is executed as described in part 3.2 of the proposal (1). Table 1 shows the equipment used in the experiment, computer resources, and the ROS application execution. ROS Task is a TurtleBot3 application and ROS tool. The experiment uses rosbag and records all ROS communication data. Figure 7 shows the flow of topics on ROS. The measurement was performed for 30 minutes in the environment of about 23 m² in Fig. 8.

4.2 Result

Table 2 shows the amount of data for each topic recorded by rosbag, which is the task that consists of the gmapping. Figure 8 shows the time transition of the total amount of data. The total amount of data obtained by SLAM increased...
in proportion to time. In addition, the cumulative amount of data measured in 30 minutes doubled. The amount of data in each topic in Table 2 hardly changed. The size of the topic is changing in tf and rosout, but there is almost no fluctuation. And topic map is the largest in all. The space measured this time is again about 23 m², however it is actually expected to be used in towns and structures. In this case, the basic amount of total data is expected to increase according to the measurement time. It is desirable to use the data cache of the edge server to improve the responsiveness and reliability of processing a large amount of data.

5. Evaluation under the Edge Environment

5.1 Overview

We implemented the proposed API, and measured the API overhead. The client application used is SLAM (gmapping) and TurtleBot3 running on the simulation software gazebo. On the edge server, we executed fogcached-ros as proposed [10]. Table 3 shows the resources and tasks of the computer that uses the client application. Fogcached on the edge server uses 4 GB of DRAM and 32 GB of DCPM. We ran SLAM for 30 minutes, measured the API overhead from fogcached-ros, and evaluated the API.

5.2 Result

Table 4 shows the message data size of each topic.

Figure 10 shows the percentage of setAPI overhead and the percentage of getAPI overhead. ‘libmemcached’ is the time it takes to send and receive data using the libmemcached library [30]. The serialization is an operation by serialization included in ROS. The time required to execute both serialize and deserialize was less than 1 ms, which was less than 50% of the total. Both setAPI() and getAPI() function execution have a positive correlation between message data size and overhead. Also, as the data size increases, the ratio of time required for serialize and deserialize increases compared to the time for the entire API. Looking at the overall API ratio, the map, which is the largest piece of data, was 629 μsec (about 43%) for serialize and 596 μsec (about 45%) for deserialize. On the other hand, the smallest data, imu, was about 12% for serialize and about 7% for deserialize.

Figure 11 shows the total API overhead per 1,000 seconds. In addition, the transmission cycle differs greatly de-
pending on the topic as shown in Table 4. Since the map is transmitted only 0.07 times per second, it can be said that the overhead of the map is very small when considering it in units of 1000 seconds. In the case of 1000 seconds, map is about 0.5% or less of imu.

If the message size is very large, the overhead is affected by serialize and deserialize. However, in the case of a message of about 1 KiB, the serialize and deserialize have almost no effect on overhead. Therefore, it is considered that there is no problem even if the map has a large overhead per hour as the usage time becomes longer. Frequent ones have a large overhead like imu, but the transmission frequency cannot be controlled on the API side, and therefore it is not directly related to the API, so it is not considered in this paper.

6. Evaluation Connected with the Cloud

6.1 Overview of the Quantitative Evaluation

In this experiment, the cloud cooperation of part 3.1 of the proposal (2) is evaluated in an environment including the cloud. In order to show the usefulness of the fogcached and fogcached-ros on the actual edge-cloud environment, we prepared an edge server and a cloud and conducted an evaluation experiment.

As we firstly described, fogcached is the extended KVS server that can use both of DRAM and NVMM as a main memory. It achieve the purpose of cache responsibility and reliability. However, it could not achieve the effectiveness on the practical application and not evaluated under the actual environment. Therefore, we developed fogcached-ros, which makes possible to use ROS application SLAM by providing the APIs that connect the fogcached automatically. In this section, using the SLAM application, our total system (fogcached and fogcached-ros) should be evaluated. By these evaluation, we could clarify the how much the data stored on the fogcached cache server and cloud, and what is the cause of effect on the hit rate and latency. In these evaluation, we assumed that all data is stored in the cloud and some data is cached at the edge. In order to investigate the latency distribution and the ratio of the number of hits of DRAM and DCPM, we set multiple conditions and experiments as shown in Table 5.

As with the API evaluation, the experiment assumes TurtleBot3, SLAM, and teleop as clients. In the API evaluation environment, it is not possible to reproduce the simultaneous transmission of a large amount of data, so we implemented a Client that publishes the topics of imu, scan, joint_states, tf, and map, and conducted experiments. In this experiment, memcached-1.5.16 [15] is used as the cloud. Furthermore, assuming that multiple robots actually transmit data, a large amount of ROS data is cached in fogcached and saved in the cloud.

Figure 12 shows a schematic diagram of the experiment. Since the cloud is assumed, a communication delay is added. Fogcached-ros normally communicates with the cloud and sends and receives data, but this time, we installed ROS on the machine on the cloud side and published the topic from the edge to the cloud server.

The client side measures the latency until the data can be acquired. In order to investigate the ratio of the number of hits of DRAM and DCPM, we experiment by changing the amount of memory to be mounted and change the settings for the different conditions. The experimental conditions are shown in Table 5.

Under the experimental conditions, the “Standard” was the standard pattern, the number of IDs was 1,000,000, and the set:get rate ratio was 1 : 9. The standard memory configuration was DRAM 4 GB and DCPM 16 GB. For the other conditions, only one condition was changed from Standard. By changing only one condition and comparing multiple conditions, we clarify the factors that have a decisive influence on the performance of the ROS system.

The conditions are following. The Map_10% is the ROS pub/sub makes only the topic of map at 1/10...
speed. The communication speed at this time is the map rate. ID\_change1 changes the maximum number of IDs to 75,000, and ID\_change2 changes the maximum number of IDs to 1,250,000 for comparison. DRAM6, DRAM, and DRAM10 have different memory DRAM ratios. In comparison with these patterns, latency is measured and the transition of the number of hits by memory is measured and considered.

### 6.2 Evaluation Result

The evaluation of latency was plotted on a graph using kernel density estimation as the latency distribution. In addition to imu, the topics used in the experiment include scan, joint\_states, and map, but since these were not different from the results of imu in all patterns, imu is referred to in this paper.

1) Comparison of different memory rates on DRAM and DCPM

DRAM and DCPM targeted by fogcached, which is hybrid-KVS, have different latencies as memory devices. In this evaluation, we will clarify how the difference in the rate of the memory device affects the responsiveness under the edge-cloud configuration. As this practical environment, we measure the responsiveness when accessing the edge (fogcached-ros, fogcached) server and the cloud server, Fig. 13 shows the results of the latency distribution between the cloud and imu.

In Fig. 13, the two peaks on the X-axis show low latency around 0.000 and middle latency from 0.001 to 0.002. The former is the access to the edge server, and the latter is the access to the cloud. The peaks with high density show the higher latency of access to cloud than the edge server. A clear difference is observed between the latency on the edge and cloud.

On the other hand, comparison of results made by changing the memory ratio of 4 patterns for DRAM and DCPM is of little significance. From the results, it was found that by increasing the DCPM of the edge server to 16 GB, more data is stored on the edge cache server. Even if the difference appears in the DCPM size, there was no large difference in the latency distribution related to the memory ratio, so it is considered that the memory ratio is less related to the responsiveness than to the DCPM size.

2) Comparison of the number of hits on different DRAM and DCPM ratios

In this evaluation, it is confirmed whether the SLAM data is moved to the DCPM side according to the number of accesses. This is because the fogcached implementation automatically moves the data between DRAM and DCPM according to the number of accesses [10]. The purpose is to verify that it has a function to make it in an actual application. At the same time, regarding changes in the ratio of DRAM and DCPM, the effect of the data movement algorithm will be clarified by observing whether the data exists in DRAM or DCPM when extracting the data. Figure 13 shows the time transition of the number of DRAM hits (Get from DRAM) and the number of DCPM hits (Get from DCPM) when the memory ratio between DRAM and DCPM was changed from (a) to (d).

As the ratio of DRAM increased from (a) to (d), the number of DRAM hits increased in the number of hits to the cache on the Y-axis. However, in each pattern, the cache hit rate in DCPM was finally higher than that in DRAM by about 80%, and it was found that the reliability of the data was ensured by moving the data from the DRAM to DCPM.

3) Experimenting with different ID types

In this evaluation, we evaluate the latency distributions of imu and map are shown in Fig. 15 and 16. Imu and map are topics handled on SLAM. Imu is
the data obtained by calculating the inertial force from the acceleration data of TurtolBot3. Map represents a 2D grid map, and map functions as a topic for writing map information generated by SLAM based on the surrounding environment data obtained from the robot. An array containing 2D map data for display each cell in the array is the occupancy of the map. In this experiment, in addition to these, scan, joint_states, and tf were also measured, but since these results were not different from imu, imu and map were evaluated. Also, ID types are related to the number of data stored in fogcached. In this experiment, we will clarify the effect of increasing or decreasing the number of data that can be stored.

We compared different IDs. When there are 750,000 types of IDs, they are more likely to remain in the edge cache than the standard 1,000,000, and the number of accesses at the edge is about 159% compared to the access at the cloud. When there are 1,250,000 types of IDs, it is less likely to remain in the cache than the standard, and the number of accesses at the edge is about 42% compared to the access at the cloud. On the other hand, in the map shown in Fig. 15, the difference between edge and cloud is small, and the latency distribution is large. Which memory to hit, edge or cloud, was not related to the type of ID. There was no difference in the number of hits to DRAM and DCPM in this experiment.

6.3 Discussion

1) Discussion for the design

The APIs we have provided, transfers the data as serialized data, have the following advantages: The first advantage is its versatility. Serialize data can be used universally for various interfaces when sending and receiving with embedded system of robot to the high-performance computer. The second advantage is that the amount of data is smaller than text. The secondly, we consider that the amount of data is smaller than the text.

However, when we want to handle data more universally in the future, it is necessary to consider the protocols that send and receive data from the different middleware. To satisfy the requirements, it is possible to consider employing ROSBridge [30] which converts the ROS data format to JSON format. ROSBridge has the advantage of being widely used because it can handle up to the level of higher-order protocols. An increase in processing delay and a decrease in throughput performance have been pointed out [31] due to an increase in the amount of memory used and the CPU load when using ROSbridge. Hence, we need to consider the trade-off between versatility and performance.

It is also necessary to consider the use of ROS Parameter [32] in terms of providing standard functions for using KVS mechanisms in ROS. Since parameters store data in Master (roscore) or Parameter Server which is not designed for high performance as noted in the official wiki [33], it is necessary to take measures if data flow is increased or overloaded when sending data to KVS. This means that it is necessary to design policies and rules for this data flow. The main purpose of this research is to set up a KVS using DRAM/DCPM as a highly reliable and responsive edge server, and to realize and evaluate the edge/cloud system that can be used by applications via APIs to ROS. Therefore, this time, we used the proposed method as an API that can be executed lightly with low overhead.

In this proposal, we provide an API that explicitly serializes data in the form of setAPI and getAPI. The advantage of providing an API is that developers can use its functions intentionally. When communicating from ROS to KVS, various methods can be considered. For example, there is a way to not serialize the data. However, there is a demerit that mutual conversion is required. By providing an API, developers can make a selection considering these advantages and disadvantages.

However, on the other hand, it is undeniable that the specification of providing the API imposes a burden on the ROS developer. The method of specifying the topic in the XML configuration file etc. without rewriting the ROS and saving the data in the cloud is It is considered to be a convenient method. If there is a request in the future, we will consider implementing it.

2) Discussion for the basic evaluation

Following experiment (1), we considered the reason for there being no large difference in the ratio of the amount of memory. We found that the difference in performance between DRAM and DCPM was only nano seconds. Since this experiment was a time measurement and the tools generally observed millisecond units, it is probable that the performance difference was not clearly observed in the results. Therefore, it can be said that there is a merit because it is considered that the influence of performance deterioration is small even if the ROS application hits with the memory of DCPM.

In addition, in the comparison of increase/decrease of ID types in (3), the data of 750,000, which is the smallest combination of IDs, remained in edge, which is 1.5 times the standard. This is because the maximum number of IDs has decreased and the number of IDs used has decreased, so access to keys with the same ID has increased, making it difficult for them to be discarded from the cache. This means that if we would like to process a large amount of non-identical SLAM data from multiple robots, we will be
dealing with many types of data. This is the same as the situation where there are many types of identifier IDs, so if we increase the cache to the edge server using SLAM, the method of reducing the number of IDs would be effective.

Furthermore, this time, the result is that the variance of the map latency distribution is large in all patterns. From this result on the edge server, it is considered that the effect of edge computing is low for large data such as the map.

3) Discussion for the evaluation for predictive guarantee for the cache hit rate accordance with the number of access

To clarify the effect of hit rate on the edge cache by changing the number of access for the server, we setup the additional evaluation. If it achieved, we could discuss the predictive guarantee for the cache hit rate accordance with the number of access.

We prepared a data set (right) that fairly considered the tendency of access distribution to fogcached (edge server). Compared to the previous experiment that was one with random access to fogcached, and as shown in Fig. 17 (left), the access distribution was normal distribution.

To explain a little about the data set in Fig. 17 (left), in this experiment, data sets with different distributions of access numbers for each data are used. First, (1) the distribution of the number of accesses to the data is all equal (All equal rate), (2) the proportion of the data with the most accesses is high and the proportion of the data with the least number of accesses is low (Low access, Low rate), (3) of all the data, the ratio of data with a large number of accesses is low, and the ratio of data with a small number of accesses is high (Low access, High rate). The distribution of these access numbers is shown in the figure. In addition, the number of accesses to fogcached and the number of accesses to the cloud for all data are adjusted according to this access number distribution.

Figure 18 shows the hit rate for each data set. The hit rate increased as the number of accesses increased. And the hit rate changed almost linearly. Random access and data set (3) hit rates converge to about 18%. The data set (2) (Low access, low rate) had a particularly high hit rate on the whole, rising to about 25% and increasing.

From the experimental results, it can be said that the data hit rate to edge can be predicted from the distribution of the number of accesses to fogcached. Whether or not the data remains in the edge server cache depends on the access method on the server side. Therefore, it can be concluded that reliability is guaranteed if the server-side settings are based on access prediction.

7. Conclusion

In this paper, we proposed to make the hybrid main memory KVS server compatible with SLAM. In the preliminary experiment, we decided that SLAM was suitable for the evaluation of edge computing, so we extended ROS and designed and implemented fogcached-ros, which has an API to communicate with fogcached. As a result of evaluating the overhead of the API, the overhead of serialization was less than 1 ms, so it is considered that the API can handle many topics. Furthermore, we evaluated the responsiveness and reliability assuming the cloud for a system linked with the cloud. It was found that since the latency difference due to the speed difference between DRAM and DCPM is small, there is little effect on ROS communication regardless of which one is hit, and that the effect of edge computing may be low for large data. In cloud evaluation using a data set, the hit rate can be predicted from the access distribution, and it is known whether or not it remains in the data cache. Therefore, it was found that reliability is guaranteed if the server-side settings are based on access prediction.

A future task is to find an effective edge computing effect for large data. In addition, it is necessary to evaluate how much data is accumulated as a data cache by clarifying how much data is pushed out from fogcached. Furthermore, it is possible to improve versatility by moving to ROS2.

About the future work in the design of API, this time, we implemented the setAPI/getAPI to topic, however, it is possible to be extended to service as bidirectional communication. Since it is thought that the utility value will increase further, extending it to service will be an issue for the future. Also, it is necessary to consider the ROS bridge [30]. Since it can also use higher-order protocols, it may be possible to provide this mechanism for more general purposes. In the future, we would like to consider using it from the viewpoint of generalization of the proposed specifications.

In this implementation, the specification is that the ROS developer rewrites the code. In the future, if requested, we will consider specifications such as saving data to the
cloud by specifying topics in an XML configuration file, etc., without rewriting the application program.

Acknowledgments

This research was supported by the Japan Science and Technology Agency (JST), CREST, JPMJCR19K1.

References

[1] V.A. Knyaz and A.G. Chibunichev, “Photogrammetric techniques for road surface analysis,” ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol.XLI-B5, pp.515–520, July 2016.
[2] Security Robot System, https://smprobotics.com/security_robot/robot-security-system/ (accessed 2021-01-04)
[3] K. Ghaffarzadeh and N. Jiao, “Mobile robots, autonomous vehicles, and drones in logistics, warehousing, and delivery 2020–2040,” IDTechEx Research.
[4] V. Honkote, D. Kurian, S. Mathukumar, D. Ghosh, S. Yada, K. Jain, B. Jackson, I. Klotchkov, M.R. Nimmagadda, S. Dattawadkar, P. Deshmukh, A. Gupta, J. Timbadiya, R. Pali, K. Narayanan, S. Soni, S. Chhabra, P. Dhamra, N. Sreenivasulu, J. Kollikkunnel, S. Kadavakollu, V.D. Sivaraj, P. Aseron, L. Azarenkov, N. Robinson, A. Radhakrishnan, M. Moiseev, G. Nandakumar, A. Madhukumar, R. Popov, K.P. Sahu, R. Peguvandla, A.D.R. Ruiz, M. Bhartiya, A. Srinivasan, and V. De, “Distributed autonomous and collaborative multi-robot system featuring a low-power robot SoC in 22nm CMOS for integrated battery-powered minibots,” IEEE International Solid-State Circuits Conferences, pp.48–50, Feb. 2019.
[5] FlytBase, https://flybase.com/ (accessed 2021-01-06)
[6] M.D. Donno, K. Tange, and N. Dragoni, “Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog,” IEEE Access, vol.7, pp.150936–150948, Oct. 2019.
[7] J. Yuan and X. Li, “A reliable and lightweight trust computing mechanism for IoT edge devices based on multi-source feedback information fusion,” IEEE Access, vol.6, pp.23626–23638, April 2018.
[8] K. Bilal, O. Khalid, A. Erbad, and S.U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers,” Computer Networks, vol.130, pp.94–120, Jan. 2018.
[9] H. Gu and H. Wang, “A distributed caching scheme using non-cooperative game for mobile edge networks,” IEEE Access, vol.8, pp.142747–142757, July 2020.
[10] K. Ozawa, T. Hirofuchi, R. Takano, and M. Sugaya, “fogcached: DRAM-NVM hybrid memory-based KVS server for edge computing,” International Conference on Edge Computing, Lecture Notes in Computer Science, vol.12407, pp.50–62, Springer, Sept. 2020.
[11] memcached, http://memcached.org/ (accessed 2021-01-06)
[12] J.S. Lum, “Utilizing robot operating system (ROS) in robot vision and control,” National Technical Reports Library, U.S. Department of Commerce, 2015.
[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng, “ROS: An open-source robot operating system,” ICRA Workshop on Open Source Software, 2009, https://scholar.google.com/citations?view_op=view_citation&hl=en&user=01x7S-4AAAAJ&citation_for_view=01x7S-4AAAAJ:u5HHmVDpt08C
[14] Intel® Optane™ Persistent Memory, https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html (accessed 2021-01-06)
[15] T. Hirofuchi and R. Takano, “A prompt report on the performance of Intel Optane DC persistent memory module,” IEICE Trans. Inf. & Syst., vol.E103-D, no.5, pp.1168–1172, May 2020.
[16] X. Wu, F. Ni, L. Zhang, Y. Wang, Y. Ren, M. Hack, Z. Shao, and S. Jiang, “NVMcached: An NVM-based key-value cache,” Proc. ACM SIGOPS Asia-Pacific Workshop on Systems, Article No. 18, pp.1–7, Aug. 2016.
[17] W. Jiang, G. Feng, S. Qin, and Y. Liu, “Multi-agent reinforcement learning based cooperative content caching for mobile edge networks,” IEEE Access, vol.7, pp.61856–61867, May 2019.
[18] X. Wang, Y. Han, V.C.M. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” IEEE Commun. Surv. Tutor., vol.22, no.2, pp.869–904, Jan. 2020. 10.1109/comst.2020.2970550
[19] X. Xia, F. Chen, Q. He, J.C. Grundy, M. Abdelrazek, and H. Jin, “Cost-effective app data distribution in edge computing,” IEEE Trans. Parallel Distrib. Syst., vol.32, no.1, pp.31–44, Jan. 2021.
[20] J. Ren, G. Yu, Y. He, and G.Y. Li, “Collaborative cloud and edge computing for latency minimization,” IEEE Trans. Veh. Technol., vol.68, no.5, pp.5031–5044, May 2019.
[21] K. Haseeb, I.U. Din, A. Almogren, Z. Jan, N. Abbas, and M. Adnan, “DDR-ESC: A distributed and data reliability model for mobile edge-based sensor-cloud,” IEEE Access, vol.8, pp.185752–185760, Oct. 2020.
[22] J. Huang, J. Liang, and S. Ali, “A simulation-based optimization approach for reliability-aware service composition in edge computing,” IEEE Access, vol.8, pp.50355–50366, March 2020.
[23] Gmapping, http://wiki.ros.org/gmapping (accessed 2021-05-03)
[24] M. Tomono and Y. Haru, “Current status and future prospects of SLAM,” System/Control/Information, vol.64, no.2, pp.45–50, 2020.
[25] TukubaChallenge, https://tsukubachallenge.jp (accessed 2021-01-03)
[26] TurtleBot3, “turtlebot3-emual,” https://emual.robolis.com/docs/en/platform/turtlebot3/overview/ (accessed 2021-01-06)
[27] ROS (TurtleBot3) navigation, http://wiki.ros.org/turtlebot3_navigation (accessed 2021-01-03)
[28] vision_opencv, http://wiki.ros.org/vision_opencv (accessed 2021-01-03)
[29] libmemcached, https://libmemcached.org/libMemcached.html (accessed 2021-01-06)
[30] ROS bridge, http://wiki.ros.org/rosbridge_suite (accessed 2021-05-03)
[31] K.S. Sikand, L. Zartman, S. Rabiee, and J. Biswas, “Robofleet: Secure open source communication and management for fleets of autonomous robots,” arXiv:2103.06993
[32] ROS Parameter, http://wiki.ros.org/rospy/Overview/Parameter%20Server
[33] ROS Parameter Server, http://wiki.ros.org/Parameter%20Server

Koki Higashi is a student in Information Engineering at Shibaura Institute of Technology from 2017. He is a student member of IEICE, IPSJ of Japan.
Yoichi Ishiwata is Chief Developer of Ales Inc. He developed ART-Linux that is a real-time OS for robotics. He received master’s course degree from the University of Tokyo in 1991. His research interests include system software, robotics computing, and AI.

Takeshi Ohkawa received the B.E., M.E. and Ph.D. degrees in electronics from Tohoku University in 1998, 2000 and 2003 respectively. He joined Utsunomiya University in 2011 as an assistant professor. Since 2019, he is an associate professor at Tokai University. His current research interests include design technology of an FPGA to realize a low power robot vision system, and model driven development of FPGA systems. He is a member of IEEE, ACM. He is also a member of IEICE, IPSJ of Japan.

Midori Sugaya is a professor of the Shibaura Institute of Technology (SIT), Faculty of Science and Engineering, Japan. She received her Ph.D. from the Waseda University in 2010. Previously, she was a researcher of CREST project, and also belonged to the Research and Development Center of the project from 2008–2010. Her research interests include system software and dependable computing, and emotion aware robotics. She is a member of IEEE, ACM. She is also a member of IEICE, IPSJ of Japan.