fogcached-ros: Hybrid main memory KVS server

Koki Higashi[†1] Yoichi Ishiwata[†2]
Takeshi Ohkawa[†3] Midori Sugaya[†1]

Abstract: In recent years, to process a large amount of sensing data from IoT devices, it is expected to place an edge server located closer than the cloud to reduce the delay from the distance. Edge servers are required to have high responsiveness and reliability. Within the Edge researches, fogcached successfully reduces the response time because of introducing the KVS caches servers, and increase reliability uses not only DRAM but also Non-Volatile-Memory (Intel Optane Data Center Persistent Memory) to store the less access cache. It provides a dual-LRU mechanism that is extended by the memcached as a management for the hybrid memories. However, because fogcached was only evaluated by simulation, it is not clear the mechanism also effective for the practical applications. The purpose of this study is to evaluate the fogcached with the practical robotic application which is necessary to apply the edge computing. These days, robotics application is developed generally on the ROS, and SLAM (Simultaneous Localization and Mapping) application. To apply the fogcached to the ROS and SLAM, we designed an extension of ROS namely fogcached-ros, which executed on ROS to connect to fogcached. It stores and retrieves data in fogcached with API. In preliminary experiments, we thought SLAM as suitable for evaluation of edge computing. We evaluated the API overhead measurement results and found that it was less than 1ms. We concluded these API support many topics.

Keywords: ROS, SLAM, KVS, memcached, Edge Computing

1. Introduction

In recent years, autonomous mobile robots have become widespread in warehouses and facilities for the purpose of reducing the labor. The features of these services are the systematic operation of multiple and autonomous driving robots such as the road surface property measurement robots and security robots [1]. It measures the road surface from the sky with multiple drones and calculates the position and inclination angle of the object to measure it more accurately and regularly at a lower cost than humans. Furthermore, it can be widely applied, such as collecting measured data in real time, generating a map, and using it for environmental measurement. 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 two cases, there are many uses such as an autonomous driving delivery [3], and a system developed by Intel that uses multiple small robots to search for and rescue victims and missing persons [4].

These robots process a large amount of sensor data at the same time as the actuation. Conventionally, sensor data used for grasping physical information that is used for control, and it is not saved and is discarded after calculation processing. In recent years, services that send data collected by robots to the cloud generally connected to the network [5], and it is expected that a large amount of sensor data that is sent by the multiple robots will be stored in the cloud service. However, when storing data in a cloud server, remote processing to the cloud is concentrated, so the cloud server becomes overloaded and network delays such as wireless are problems.

On the other hand, in recent years, edge computing for handling a large amount of data at a shorter distance than the cloud has been studied [6]. 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. 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 the edge server.

To improve the reliability and responsiveness of the data, Ozawa et al. developed fogcached that is the implementation to NVM based on memcached, which is a typical implementation of the key-value store, and showed its effectiveness [7]. However, their effectiveness was limited within the simulation, and it is not clear the effectiveness for the practical applications.

To clarify the effectiveness as an edge computing, we consider evaluating a practical robotic application that has not been considered even if it is necessities to the responsibility and reliability for their flexible service. Therefore, in this study, to evaluate the fogcached, we use the practical robotics application such as ROS and its measure application SLAM.

To achieve the purpose, we propose a system and extension API on ROS to connect the fogcached, since it does not have any mechanism to apply the ROS and its application. We named it as fogcached-ros. In this paper, we describe the detail of the design of the extension of fogcached-ros and execute the preliminary evaluation for the practical ROS application such as SLAM using proposed API.

In this paper, the previous research for the research subject is described in Section 2, the proposal is described in Section 3, the preliminary experiment is described in Section 4, and the summary and future issues are described in Section 5.

2. Related research

With the growth of hardware research, the speed of NVM (Non-Volatile Memory) has increased, and those with speeds closer to DRAM have been developed. Intel announced DCPM (Intel Optane Data Center Persistent Memory) in 2019 [8]. DCPM is a DDR4 standard memory with a latency that is about four times that of DRAM [9].

†1 Shibaura Institute of Technology
†2 VA Linux Systems Japan K K
†3 Tokai University

Wu et al. proposed NVM cached [10]. NVMcached is an NVM-based key-value cache. They considered 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, and proposed a method that considers the write endurance of NVM as NVM cached. Specifically, the list of objects (metadata) is saved in DRAM, and the objects are saved in NVM. In consideration of write resistance, objects that are deleted and updated in a short time are saved in DRAM. The list saved in DRAM disappears after the crash, but the objects saved in the NVM reflect the access history from the client, and the list is reconstructed based on the access history. In addition, they aimed to achieve both performance and reliability by accelerating metadata processing with DRAM and guaranteeing object persistence with NVM. In this study, they improved the system throughput of real write-intensive workloads by up to 2.8 times compared to non-volatile memcached. 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.

Ozawa et al. proposed to use NVM to hold a large amount of data on an edge server and develop an In-Memory Key-Value Store on a hybrid main memory consisting of DRAM and NVM [7]. The evaluation also showed that the responsiveness of the edge server was improved. Ozawa et al. Used DCPM as NVM and made it accessible from memcached [11] as an In-Memory Key-Value Store (KVS). Fogcached, an extension of memcached proposed by Ozawa et al., has a structure that connects DRAM and DCPM. Specifically, an LRU with the same structure as the LRU for DRAM inside memcached 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 the 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 is limited to the use of non-volatile memory. In addition, in previous studies, there are problems such as (A) insufficient evaluation in actual applications, (B) non-implementation of cooperation with the cloud considering reliability.

3. Proposal

3.1 Purpose

In contrast to the issues mentioned at the end of Section 2, the purpose of this research is to realize the high speed and reliability of the edge server when using robot applications for problem (A) and (B).

3.2 Proposal summary

In this research, we propose that the hybrid main memory KVS server supports SLAM. In addition, we propose (1) design and implementation of fogcached considering robots for the problem (A) and propose (2) design and implement a system linked with the cloud for the problem (B).

3.3 ROS (Robot Operating System)

In this study, we consider to use the ROS [12] [13]. Since ROS (Robot Operating System) is an open source middleware library and tools that support software developers to create robot applications. Specifically, hardware abstraction, device drivers, libraries, visualization tools, message communication, package management, etc. are provided. ROS is licensed under the BSD license.

Currently, there are two types of ROS, ROS1 and ROS2. ROS1 is the initial version of ROS and has many users. ROS2 is an extended version of ROS1. ROS2 is a highly versatile ROS that supports a wide variety of robots and meets the requirements of modern systems such as the assumptions of multiple robot teams and real-time systems. This paper deals with ROS1.

In ROS, there are terms such as (a) node, (b) master, (c) message, (d) topic, (e) publish, and (f) subscribe. (a) node is a process that performs computation. ROS is designed to be modularized on a fine-grained scale, and robot control systems usually consist of many nodes. (b) The master manages the entire ROS communication. In ROS1, a node cannot look for other nodes, exchange messages, or request services without a master. (c) message is the data of communication between nodes. One message is a simple data structure consisting of type fields. (d) topic is a name to distinguish the contents of message. The message is published and subscribed through the exchange. node sends a message according to the topic given by publish. Regarding (e) and (f), ROS performs Pub / Sub communication. Pub / Sub communication is an asynchronous message service that separates the service that processes events from the service that generates events. The side that sends the data is the Publisher, and the side that receives the data is the Subscriber.

3.4 SLAM (Simultaneous Localization and Mapping)

SLAM (Simultaneous Localization and Mapping) [13] is a general term for technologies that simultaneously estimate the self-position of a moving body and create an environmental map. The SLAM application receives data from robots and sensor nodes and creates a map including its own position based on numerical values. By utilizing SLAM, it is possible to create an environmental map in an environment where the moving body is unknown. The robot uses the constructed map information to perform tasks while avoiding obstacles.

3.5 Client Application

SLAM technology is installed in drones, UAVs, and robots, and many services are used. In addition, SLAM handles various sensor data and requires a lot of computer resources. Therefore, the application to be evaluated is SLAM.

3.6 Proposed system.

Figure 1 shows the overall view of the designed system. The design contains a client application computer and edge server and a cloud. The application on the client side is built with the existing open source ROS application. As a ROS application, we installed a SLAM application (gmapping) on the turtlebot3 [14]. On the edge server, we installed the fogcached [7] implemented in the

previous research. From the edge server, on which a ROS Application (observer) data is sent and receive to the client. In addition, the observer sends and receive data in and out to fogcached. As for the cloud, it is received by the ROS node, stored in the database, and retrieved in the same way as the edge server. The communication method is unified by using ROS throughout this system.

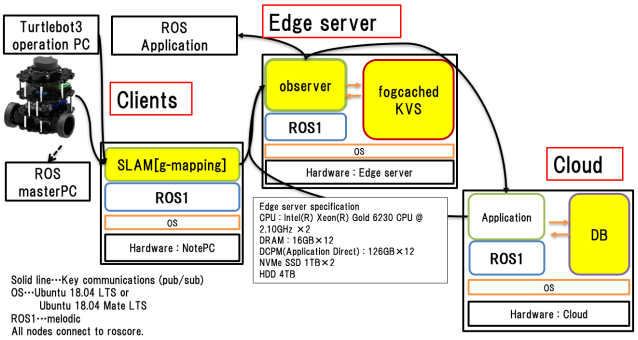

Figure 1: System diagram

3.7 Design and implementation of SLAM and server integration

With conventional ROS, there is no way to communicate with fogcached. In addition, the topic on conventional ROS communication is basically overwritten, and only the latest information remains. Figure 2 shows a conventional ROS communication diagram. As shown in Fig. 3, the proposed system communicates with fogcached so that past data can be saved. We thought that this would improve convenience and reliability. The ROS application observer has two nodes which are observer_set and observer_get.

In addition, communication between the observer and fogcached is performed by two API. The interface is shown in Fig. 4. The detailed operation is described in 3.8.

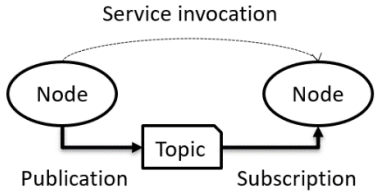

Figure 2: Traditional ROS communication

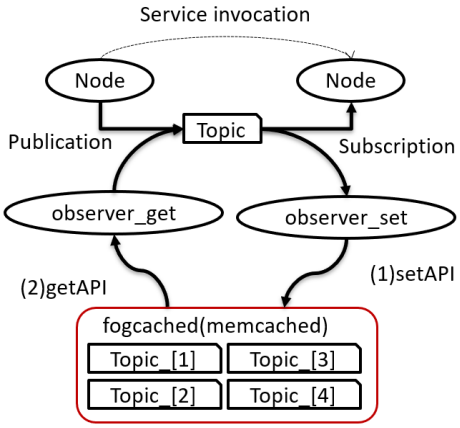

Figure 3: Proposed system ROS communication

```
int set_memcached(topic, class message, ID);
int get_memcached(topic, class& message, ID);
```

Figure 4: API interface

3.8 Communication API

As shown in Fig. 4, we created two API to connect fogcached. As shown in Fig. 3, communication between ROS and fogcached is performed by setAPI and getAPI. These are executed in the ROS node. Since fogcached is KVS, it is necessary to generate Key and Value. Both APIs receive topic and ID. The setAPI also receives the message, but the getAPI message returned by reference. Figure 5 shows the processing flow using the API interface. In both APIs, topic and ID become key as a concatenated character string with an underscore (_) in between topic and ID. The upper limit of key is set to 50 bytes. The message of setAPI is serialized and converted into a character string. The serialized character string is entered in value, but the heap area is dynamically allocated. In setAPI, these keys and values are saved in fogcached. The message was serialized in setAPI, but the message deserialized in getAPI and the restored message in fogcached is returned. In addition, API support many topics, not just one type. Figure 6 shows the setAPI call when it is executed in the Subscriber callback. However, some processing is omitted in the comments.

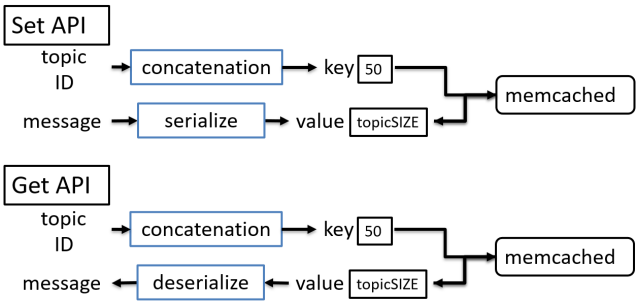

Figure 5: API processing flow

```
//callback function in Subscriber
void jointStateMsgCallback
(const sensor_msgs::JointState::ConstPtr &msg)
{
  char key[KEY_BUFFER_SIZE];
  char id[ID_BUFFER_SIZE];
  // Substitute topic for key
  // Determine id arbitrarily
  set_memcached(key,msg,id);
}
```

Figure 6: setAPI example

4. Preliminary experiment

4.1 Preliminary experiment outline

This experiment is a preliminary experiment to confirm whether SLAM is useful as an application for 3.2 Proposal (1). The purpose is to investigate the amount of data on ROS

communication when SLAM is executed. Table 1 shows the equipment used in the experiment and the computer resources. ROS Task is an open source application and ROS tool for turtlebot3. In the experiment, we record all data on ROS communication using rosbag. The measurement was performed for 30 minutes in the environment of about 23 m^2 shown in Fig. 7. Figure 8 shows the flow of topics on ROS.

Table 1: Computer resources and ROS tasks

ROS Task	OS	Memory	HDD	PC
ROS master **SLAM gmapping** **rosbag record**	Ubuntu 18.04 LTS	8GB	100GB	Windows10 Hyper-V Intel corei7 8550U
turtlebot3 bring up	Ubuntu Mate 18.04 LTS 32bit	1GB	32GB	Raspberry Pi 3B+
teleop (keyboard operation)	Ubuntu Mate 18.04 LTS 32bit	1GB	32GB	Raspberry Pi 3B+

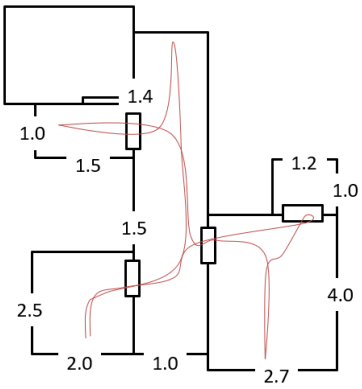

Figure 7: Measurement environment floor plan (m)

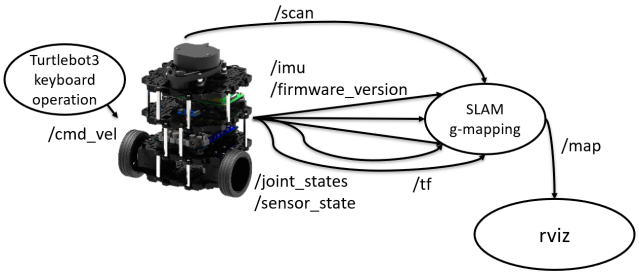

Figure 8: ROS communication diagram

4.2 Preliminary experiment results

The result of SLAM gmapping is shown in Fig. 9. The SLAM results are displayed by the visualization tool rviz. Comparing Fig. 7 and Fig. 9 with the measurement to floor, it can be seen that the map is created with the correct ratio.

Figure 10 is a semi-logarithmic graph of the time transition of the amount of topic data recorded by rosbag. There were a total of 17 types of topics on ROS communication, and there was almost no change in the amount of data in each topic. The largest amount of data at one time was map, which was 147500 bytes. In addition, the cycle in which each data was communicated was different. Although the map data communicated at one time is large, the map has a longer cycle than other topics. Of all the topics, the topics whose data volume has changed are tf and rosout. tf is a coordinate transformation to associate the position in the sensor or free space with the velocity value. rosout is a record of the entire ROS communication. The result of tf is shown in Fig. 11, and the result of rosout is shown in Fig. 12. The amount of tf data fluctuated periodically and was in the range [95,210]. Figure 11 shows only about 12.0 sec, but it lasted for 30 minutes, which is the experimental time. Unlike tf, rosout only measured the time as shown in the table. While other topics were being communicated quantitatively, rosout was communicated only a few times within the 30-minute experiment time. Therefore, neither tf nor rosout affect the transition of total data volume.

Figure 13 shows the time course of the total amount of data. The total amount of data during SLAM execution tended to increase with the measurement time. In addition, the cumulative amount of data measured in 30 minutes was about 179 Mbytes, and the amount of data required for SLAM with one unit increased by about 6 Mbytes / s.

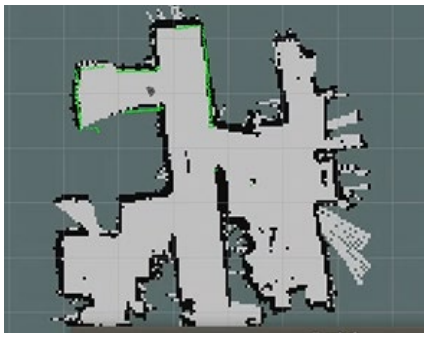

Figure 9: Gmapping result

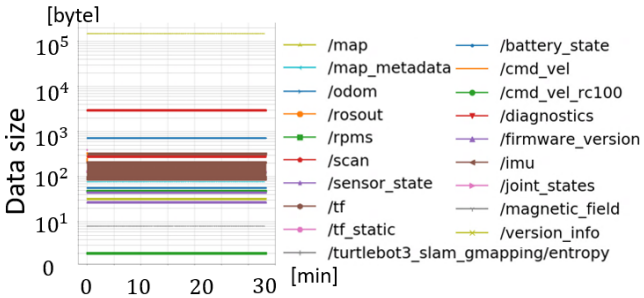

Figure 10: Time transition of data volume of all topics

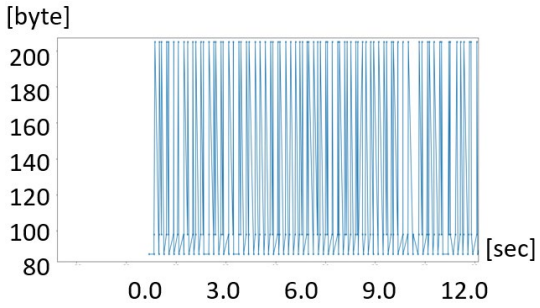

Figure 11: topic-Time transition of tf data volume

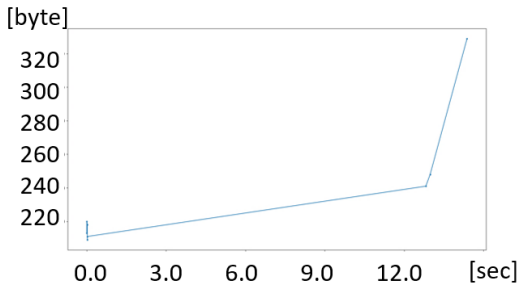

Figure 12: Time transition of topic-rosout data volume

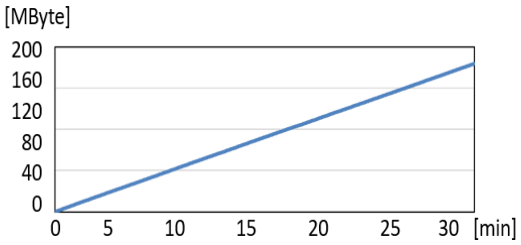

Figure 13: Time transition of cumulative data volume

4.3 Preliminary experiment consideration

The space measured by SLAM in this preliminary experiment was a very small area of about 23 m^2. However, SLAM is expected to be used in towns and structures. When measuring in units of towns and structures, the measurement time increases and the total amount of data increases, so a large area is required for data storage.

From the above, SLAM is expected to handle a large amount of data. Therefore, for evaluation of the data cache of the edge server. It is useful to use SLAM applications.

5. Evaluation

5.1 Evaluation method

We measured the API overhead shown in Proposal (2). The client applications used are SLAM (gmapping) used in the preliminary experiment and turtlebot3 on the simulation software gazebo. It also runs fogcached and the ROS application observer on the edge server. However, fogcached has enabled the binary protocol. Table 2 lists the edge server computer resources and tasks. Table 3 shows the specifications of the edge server. In addition, fogcached used 4GB of DRAM and 32GB of DCPM. However, if a cache miss occurs, deserialization will not be executed, so it is excluded from consideration of overhead. We evaluate API by running SLAM for 30 minutes and measuring the API overhead from the observer.

Table2: Computer resources and ROS tasks

ROS Task	OS	Memory	PC
SLAM gmapping	Ubuntu 18.04 LTS	8GB	Windows10 Hyper-V Intel corei7 8550U
teleop (keyboard operation)			
turtlebot3 in gazebo			
ROS master	Ubuntu Mate 18.04 LTS	196GB	Edge server
observer			

Table 3: Edge server specifications

OS	DRAM	DCPM	CPU
Ubuntu 18.04LTS	16GB ×12	126GB×12 AppDirectMode Device dax	Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz ×2

5.2 Evaluation result

Figure 14 shows the message data size for each topic. Figure 15 shows the overhead rate of the set API, and Figure 16 shows the overhead rate of the get API. SET and GET are the time it takes to send and receive data using the libmemcached library [15]. Serialization and deserialization are operations by the serializer included in ROS. The time required to perform both serialization and deserialization did not reach 1ms. Furthermore, both setAPI and getAPI show a correlation between message data size and overhead. As the message data size increased, the overall API overhead increased. Also, as the data size increases, the percentage of time it takes to serialize or deserialize tends to increase relative to the overall API time. Also, the largest data, map, took 629 μs to serialize and 596 μs to deserialize. And no message had a higher overhead than map.

Figures 17 and 18 show the total API overhead per 1000 seconds. The transmission frequency varies greatly depending on the Topic. Imu is sent more than 100 times per second, while map is sent only 0.07 times per second. In addition, scan is transmitted 4.9 times per second, join_states is transmitted 23 times per second, and tf is transmitted 66 times per second. It is observed that the overhead on the map is negligible when viewed per 1000 seconds.

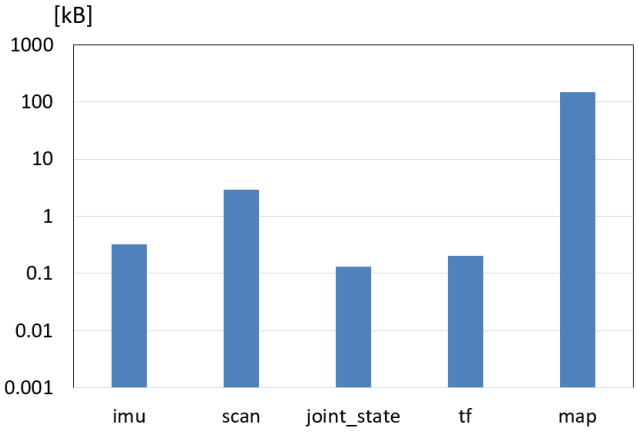

Figure 14: Topic message data size

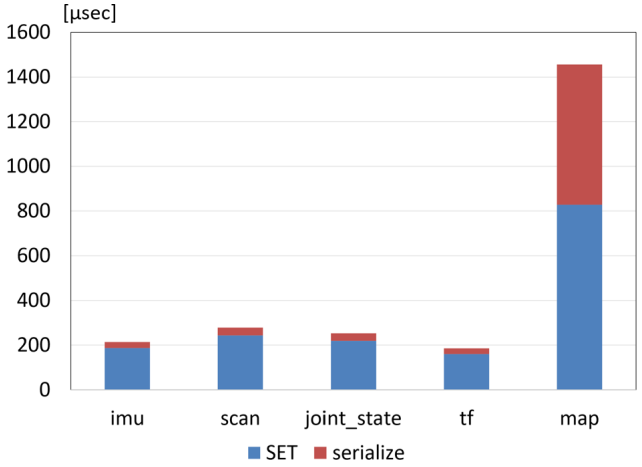

Figure 15: SetAPI overhead per one time

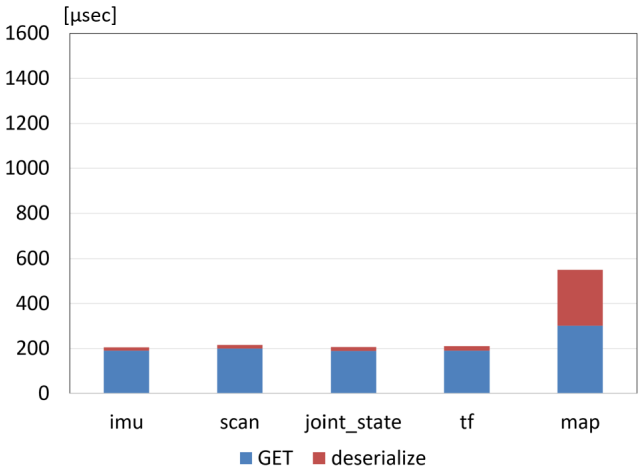

Figure 16: GetAPI overhead per one time

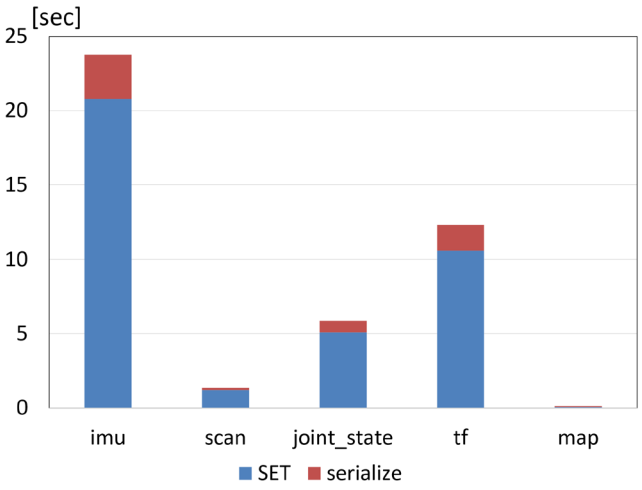

Figure 17: SetAPI Overhead per 1000 seconds

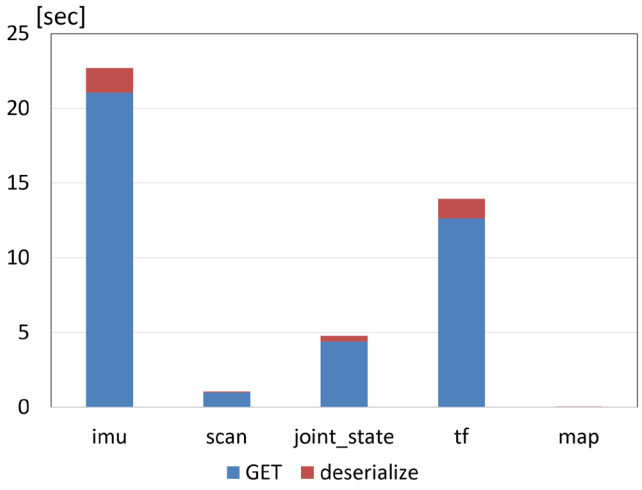

Figure 18: GetAPI Overhead per 1000 seconds

5.3 Discussion

Overhead is affected by serialization when the message size is very large. However, for messages of about 1kB, the serialization overhead has little effect.

The usage is longer than 1000 seconds, and the higher the transmission frequency, the higher the overhead. Therefore, it is considered that there is no problem for a map with a large overhead per time as the usage time becomes longer. Frequent ones have a large overhead like Imu, but this is not directly related to API. Therefore, it is not considered in this paper.

6. Conclusion

In this paper, we proposed that the hybrid main memory KVS server supports SLAM. In preliminary experiments, we determined that SLAM was suitable for using the edge server, therefore We extended ROS to create a mechanism to communicate with fogcached and measured the overhead as a evaluation. As a result, the serialization overhead was less than 1ms. The largest data, map, took 629 µs to serialize and 596 µs to deserialize. However, since this map is sent infrequently and there is no overhead when viewed in units of 1000 seconds, we

concluded that there is no problem for a map with a large overhead per time as the usage time becomes longer. Therefore, these API support many topics.

In the future, we would like to improve versatility by moving to ROS2. In addition, we would like to specifically examine the improvement of reliability by using the non-volatility of DCPM.

Reference

[1] V.A.Knyaz, A.G.Chibunichev,"Photogrammetric Techniques for Road Surface Analysis, " ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLI-B5, 2016, pp.515-520

[2] SMP Robotics Systems Corp., "Security Robot System" https://smprobotics.com/security_robot/robot-security-system/ (accessed 2020-09-28)

[3] Dr Khasha Ghaffarzadeh, Dr Na Jiao, "Mobile Robots, Autonomous Vehicles, and Drones in Logistics, Warehousing, and Delivery 2020-2040", IDTechEx Research

[4] Vinayak Honkote, Dileep Kurian, Sriram Muthukumar, Dibyendu Ghosh, Stish Yada, Kartik Jain, Bradley Jackson, Ilya Klotchkov, "Distributed Autonomous and Collaborative Multi-Robot System Featuring a Low-Power Robot SoC in 22nm CMOS for Integrated Battery-Powered Minibots, " 2019 IEEE International Solid-State Circuits Conference, pp.48-50, San Francisco, USA, 07 March 2019.

[5] FlytBase, Inc., "FlytBase" https://flytbase.com/ (accessed 2020-09-29)

[6] Kashif Bilala,b, Osman Khalidb, Aiman Erbada, Samee U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers, " Computer Networks Volume 130, pp.94-120, 15 January 2018.

[7] Kouki Ozawa, Takahiro Hirofuchi, Ryousei Takano and Midori Sugaya, "DRAM-NVM Hybrid Memory - Based KVS server for Edge Computing," 2020 International Conference on Edge Computing, August.2020.

[8] Intel, "Intel® Optane™ Persistent Memory". https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html (accessed 2020-08-04)

[9] Takahiro Hirofuchi, Ryousei Takano,"A Prompt Report on the Performance of Intel Optane DC Persistent Memory Module, " IEICE Transactions on Information and Systems E103.D(5), pp.1168-1172, May 2020.

[10] X. Wu, F. Ni, L. Zhang, Y. Wang, Y. Ren, M. Hack, Z. Shao, and S. Jiang, "NVMcached: An NVM-based Key-Value Cache". in Proceedings of the ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 1–7, August.2016.

[11] "memcached" http://memcached.org/ (accessed 2020-08-04)

[12] "ROS", https://www.ros.org/ (accessed 2020-08-04)

[13] Lum, J.S., "Utilizing Robot Operating System (ROS) in Robot Vision and Control.," National Technical Reports Library U.S Department of Commerce,2015

[14] ROBOTIS, "turtlebot3-emanual" https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed 2020-9-29)

[15] "libmemcached", https://libmemcached.org/libMemcached.html (accessed 2020-9-29)