An Edge-Cloud Computing Model for Autonomous Vehicles

Yu Sasaki[1], Tomoya Sato[1], Hiroyuki Chishiro[1],
Tasuku Ishigooka[2], Satoshi Otsuka[2], Kentaro Yoshimura[2], and Shinpei Kato[1,3]

***Abstract*— Edge-cloud computing for autonomous driving has been a challenge due to the lack of fast and reliable networks to handle a large amount of data and the traffic cost. The recent development of 5th Generation (5G) mobile network allows us to consider an edge-cloud computing model for autonomous vehicles. However, previous work did not strongly focus on the edge-cloud computing model for autonomous vehicles in 5G mobile network. In this paper, we present an edge-cloud computing model for autonomous vehicles using a software platform, called Autoware. Using 1 Gbit/s simulated network as an alternative of 5G mobile network, we show that the presented edge-cloud computing model for Autoware-based autonomous vehicles reduces the execution time and deadline miss ratio despite the latencies caused by communications, compared to an edge computing model.**

I. Introduction

In recent years, autonomous driving has captured the attention from all over the world, such as the DARPA Urban Challenge [1], and many manufacturers have been working to realize autonomous vehicles. As the state-of-the-art embedded computers lack in scalability, most products are either used in restricted and controlled space or developed to handle partial functions, such as an emergency brake [2].

SAE Standards J3016 [3] defines *Level of driving*: Level 0 being normal driving and Level 5 being completely automated driving. Level 4 and Level 5 driving analyzes moving objects from a camera and predicts their paths in real time and, therefore, demands a large amount of computation resources.

In such situations, distributed computing is common in many fields. There are two main distributed system models for autonomous driving: edge-cloud computing model and cloud-based vehicular network (CVN) model.

The edge-cloud computing model for autonomous vehicles deploys edge devices in the vehicle to reduce latency and power consumption and cloud servers to provide a large amount of computation resources. There are a few studies of the edge-cloud computing model for autonomous vehicles. Kumar et al. [4] introduced a cloud-assisted system, called Carcel, and experimented with a golf cart and six iRobot Create robots. Although this study suggested the usefulness of edge-cloud computing for autonomous driving, Carcel required sensors on multiple locations and the car was driven at 2m/s.

[1]Graduate School of Information Science and Technology, The University of Tokyo, Japan
[2]Hitachi Ltd., Japan
[3]Tier IV, Inc., Japan
We thank Akito Ohsato for his help in Autoware modification and providing ROSBAG data to test.

Another distributed model, CVN, is a model where multiple vehicles create a wireless link network, called Vehicular Ad hoc Networks (VANETs) [5], through which vehicles share data or computational resources. Although CVNs have been studied in previous works, VANETs are only useful in platoon vehicles and difficult to use if the vehicle is driven alone.

Edge-cloud computing models for autonomous driving are rarely analyzed due to the unreliable mobile network. For example, mobile networks such as the 4th Generation (4G) and Wi-Fi have an unreliable and narrow network bandwidth. The prior work noted that the mobile network was insufficient and its latency was critical for edge-cloud computing [4].

5th Generation (5G) mobile network has 20 Gbit/s bandwidth [6] in theory and much faster than the 4G network, which has 1 Gbit/s bandwidth at best [7]. 5G mobile network is beneficial to autonomous vehicles in various ways and provides sufficient bandwidth to send a large amount of data and receive the result with little latency. Thanks to 5G mobile network, vehicle vendors can use scalable cloud servers to execute heavy tasks while running light tasks in edge devices. This is both energy-efficient and cost-effective, therefore, energy-efficient offloading in 5G mobile network are studied in other fields such as mobile edge computing [8]. Meanwhile, the practicality of applying edge-cloud computing models to autonomous driving in 5G mobile network is an unsolved matter.

In this paper, we present an edge-cloud computing model for autonomous vehicles in 5G mobile network. We create 1 Gbit/s ethernet network between an NVIDIA DRIVE PX2, an NVIDIA's automotive computer, and a desktop machine to simulate 5G mobile network between an edge device and a cloud server. Since 5G has 20 Gbit/s at best and the target bandwidth is 1.2Gbit/s, 1 Gbit/s ethernet network is enough to simulate the bandwidth of 5G mobile network. We quantitatively evaluate the effectiveness of the presented edge-cloud computing model in Level 4 and Level 5 autonomous driving, called Autoware [9].

In the rest of the paper, the background is introduced in Section II. The overview of the presented edge-cloud computing model is explained in Section III. The evaluations of the presented edge-cloud computing model are introduced and the results are discussed in Section IV. We compare our work with related one in Section V and conclude this paper in Section VI.

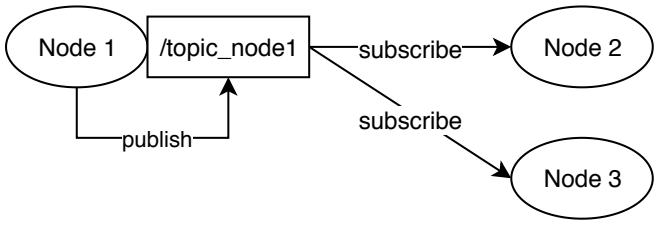


Fig. 1. Publisher/subscriber model in ROS

II. Background

A. ROS

The most of automated driving applications use Robot Operating System (ROS) [10] as a main framework. ROS is an open-source meta-operating system for robots and provides extensive tools and libraries designed for processing data from sensors and actuators. An ROS process is called a *node*, and developers can easily specify which node runs on which machine.

Fig. 1 shows a publisher/subscriber model in ROS. A node communicates with another node using a message queue called *topic*. A system using ROS framework creates a directed acyclic graph structure where a vertex represents a node and an edge of two vertexes represents a communication between two nodes. The communication between nodes is sent by TCPROS protocol which uses persistent, stateful TCP/IP socket connections. Hence, even when two nodes are on different machines, one node can seamlessly acquire the *topic* data over the network.

B. Autoware

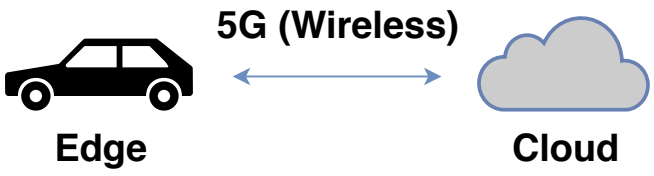
Autoware [9] is an ROS-based open-source platform developed for autonomous vehicles. Autoware collects the signals from various sensors such as camera, IMU, LiDAR, and GNSS, then, estimates the location of the vehicle, plans the path, and controls the vehicle. Autoware is designed for Level 4 and Level 5 autonomous driving and tested in demonstrated experiments in Japan [11], [12]. Now Autoware is maintained by the Autoware Foundation [13], which is a non-profit organization and composed of more than 20 global members. We use Autoware to simulate the real-world autonomous driving situation.

III. Presented Edge-Cloud Computing Model

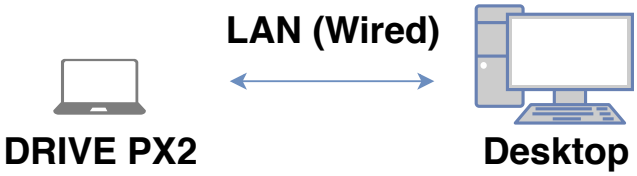
In this section, we explain the presented edge-cloud computing model. Before explaining the model, our simulation model and ROS component model are introduced.

Fig. 2 shows the actual 5G cloud model and simulation model. The actual 5G cloud model uses edge devices in the vehicle and cloud servers connected with 5G mobile network. In contrast, the simulation model uses an NVIDIA DRIVE PX2 and Desktop PC as an edge device and a cloud server, respectively.

We consider the situation where a vehicle runs on actual roads, and focus on two main features in Autoware [9]: the self-localization and path planning. We use Normal Distributions Transform (NDT) matching [14] and A* path planner [15] as the self-localization and path planning, respectively, because these features are mainly used for Autoware-based autonomous vehicles.



(a) Actual 5G cloud model



(b) Simulation model

Fig. 2. Actual 5G cloud model and simulation model

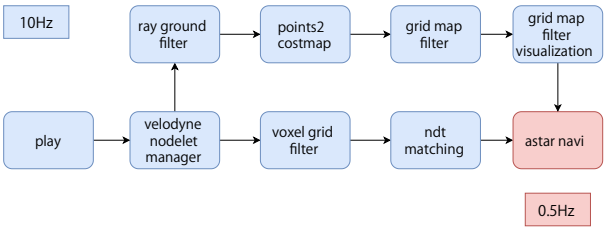


Fig. 3. Overview of our ROS component model

Fig. 3 shows the overview of our ROS component model. `astar_navi` has the period of 0.5Hz (2,000ms) and other components have the period of 10Hz (100ms). Our ROS component model is composed of the following 9 components.

1) `play` reads point clouds from ROSBAG data (recording and playing back data).
2) `velodyne_nodelet_manager` creates 360 degree point clouds by Velodyne LiDAR data.
3) `voxel_grid_filter` downsamples the point cloud data (Velodyne LiDAR data in this case) by taking a spatial average of the points in the cloud.
4) `ndt_matching` handles NDT matching, which is used as the self-localization algorithm. NDT is a laser scan matching algorithm of self-localization that uses two point cloud data. One is made beforehand (reference scan) and the other is acquired from vehicle sensors (input scan).
5) `ray_ground_filter` removes ground from point clouds.
6) `points2costmap` creates 2D projection from 3D point clouds.
7) `grid_map_filter` creates non-entering area from 2D obstacle points.

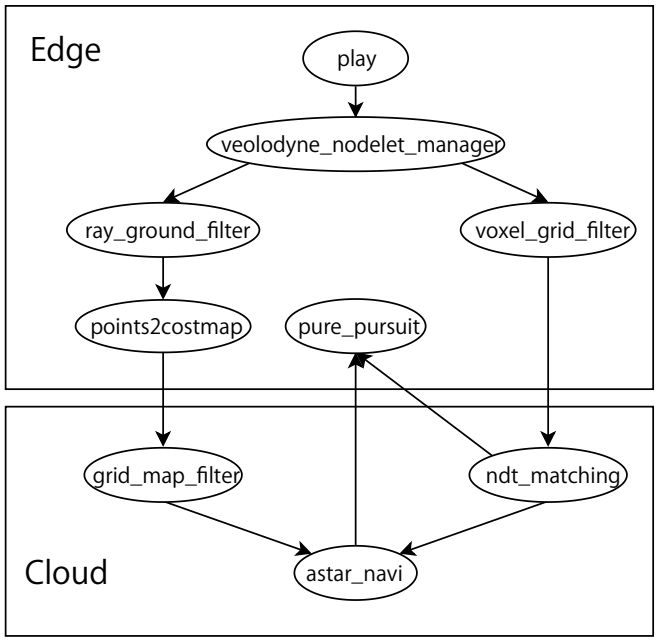


Fig. 4. Presented edge-cloud computing model

8) `astar_navi` handles A* planner, which heuristically calculates the shortest path from the start point to the goal point.
9) `pure_pursuit` is responsible for controlling vehicle actuators.

Fig. 4 shows the presented edge-cloud computing model. The edge devices execute `play`, `velodyne_nodelet_manager`, `ray_ground_filter`, `points2costmap`, `voxel_grid_filter`, and `pure_pursuit`. In contrast, the cloud servers execute `grid_map_filter`, `ndt_matching`, and `astar_navi`.

Preliminary Evaluations: The reason why the presented edge-cloud computing model is suitable for Autoware-based autonomous vehicles is because we have conducted the preliminary experiments with several edge-cloud computing models. In these edge-cloud computing models, these components are assigned to an edge device or a cloud server, respectively, in order to evaluate the execution time and communication cost. The results of the preliminary evaluations show that `ndt_matching` on DRIVE PX2 has the worst case execution time of 283ms. In addition, `astar_navi` on DRIVE PX2 often takes more than 10,000ms. The results indicate that DRIVE PX2 cannot execute NDT matching or A* planner within respective deadlines: 100ms and 2,000ms. Therefore, we conclude that the main processes of NDT matching (`ndt_matching`) and A* planner (`grid_map_filter` and `astar_navi`) should be run on the cloud server. Under this condition, we have chosen this model because of the lowest traffic (40 to 100MiBs). Other edge-cloud computing models exceeded 120MiB and 1 Gbit/s ethernet network could be the bottleneck. Therefore, these models would not be suitable.

We have made several modifications to Autoware, which are a mere fix to run on two machines. We do not modify the actual data processing algorithm that might compromise the execution time. Our main modification is creating the *sync node* to observe the message queue in each experiment and evaluate the execution time. Our implementation is available at https://github.com/pflab-ut/distributed-autoware.

TABLE I
SPECIFICATION OF EDGE DEVICE AND CLOUD SERVER

	Edge	**Cloud**
CPU	2x Tegra X2	1x Core i7-8700
Architecture	2x NVIDIA Denver + 4x ARM Cortex A57	Intel x86 6 Core (12 Thread)
CPU Frequency	2.0GHz (Denver) + 2.0GHz (Cortex A57)	3.2GHz
Memory	16GB LPDDR4	32GB DDR4
GPU	1x Pascal GPU	1x GTX1080
Total CUDA Core	512	1280
GPU Memory	4GB GDDR5	8GB GDDR5
Linux Kernel	4.9.38-rt25-tegra	4.16.5-041605-generic
Ubuntu Version	16.04	
ROS version	kinetic	
Autoware version	1.9.1	

IV. EVALUATIONS

A. Setups

The evaluations use the edge device and cloud machine, which are connected to the same network with 1 Gbit/s ethernet to simulate 5G mobile network. Table I shows the specification of the edge device and cloud server.

We use the simulation feature in Autoware. The simulation data used in the evaluations are those obtained by using the Velodyne HDL-32E LiDAR which covers 360 horizontal Fields Of View (FOV). The Velodyne HDL-32E LiDAR has 32 channels and covers +10 to 30 vertical FOV. The scan frequency is 10Hz and scan matchings are executed with 100ms period.

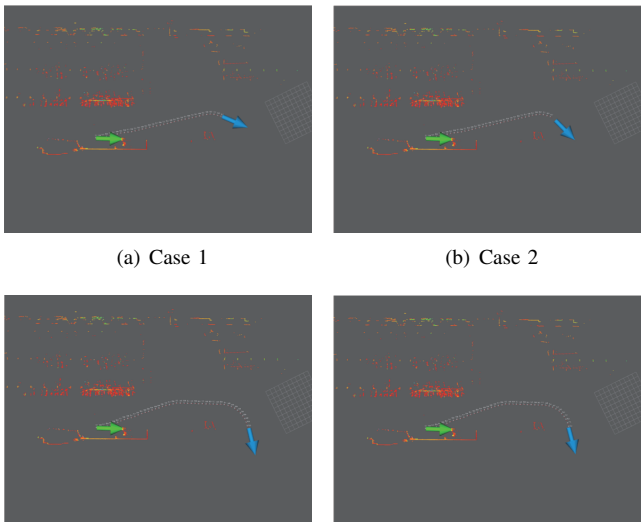
The input scan data used for simulations are available on ROSBAG STORE (https://rosbag.tier4.jp/index/), #178. These data are taken around Nagoya University, Japan, and the vehicle in this simulation travels in the same path for 3 minutes.

We denote the experiments, which are executed on the edge machine as E_{edge}, those on the cloud machine as E_{cloud}, and those under the edge-cloud computing model as E_{ec}.

Since our ROS component model has two different periods as shown in Fig. 3, we use A* planner and NDT matching because these features are mainly used for Autoware-based autonomous vehicles.

B. A* Planner

The execution time of A* planner depends on the location of the vehicle and the location of the goal. Therefore, we make four cases in which one autonomous vehicle parks on the campus street in Nagoya University, Japan, and starts to drive. These cases set the same start point and different goal points, and measure the execution time of A* planner to find the path to each goal.



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Fig. 5. Start, goal, and results of A* planner of case 1, 2, 3, and 4. The green arrow denotes the start point and the blue arrow denotes the goal point. The white line represents the presented path of A* planner. Red, green, and yellow points represent points acquired from LiDAR sensors.

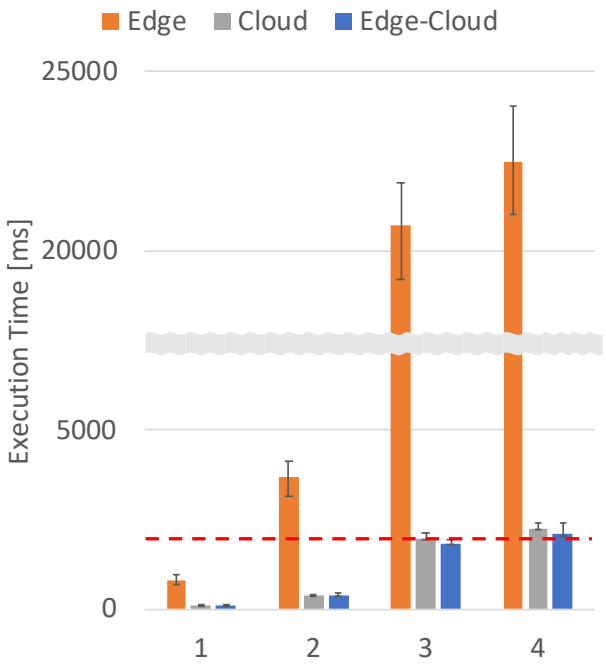


Fig. 6. Execution time of A* planner of case 1, 2, 3, and 4. The red dotted line represents 2000ms, the deadline of A* planner.

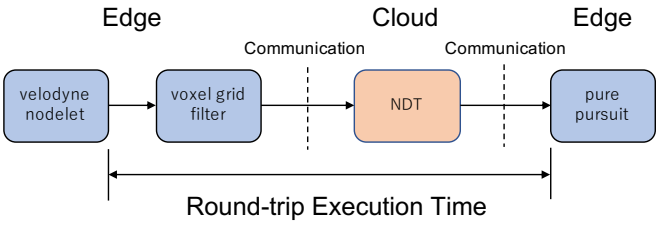


Fig. 7. Round-trip execution model

Fig. 5 shows the start, goal and, results of A* planner of case 1, 2, 3, and 4. The green arrow denotes the start point and the blue arrow denotes the goal point. We execute 100 trials of each case and measure the execution time to find the path to each goal. The white line represents the generated path of A* planner.

Fig. 6 shows the execution time of A* planner of case 1, 2, 3, and 4. For goal 1, the execution time of A* planner is within 2,000ms deadline in every model. E_{edge} misses the deadline in the goal 2 while E_{cloud} and E_{ec} are well within the deadline. For goal 3, E_{edge} is 24,037ms and E_{cloud} is 2,450ms. However E_{ec} is within the deadline, 1,949ms. Finally, for goal 4, although E_{ec} misses the deadline, E_{edge} has about 10.4 times longer execution time compared to E_{ec}. As A* planner is a CPU dependent task, E_{cloud} and E_{ec} have significant advantages against E_{edge}. Furthermore, since the computation is distributed into two machines, the results also show that E_{ec} performs better than E_{cloud}, up to 8%.

These results strongly clarify the contribution of the presented edge-cloud computing model. In addition, they address the importance of the model because even the slight change of the goal point will result their execution time to be varied drastically.

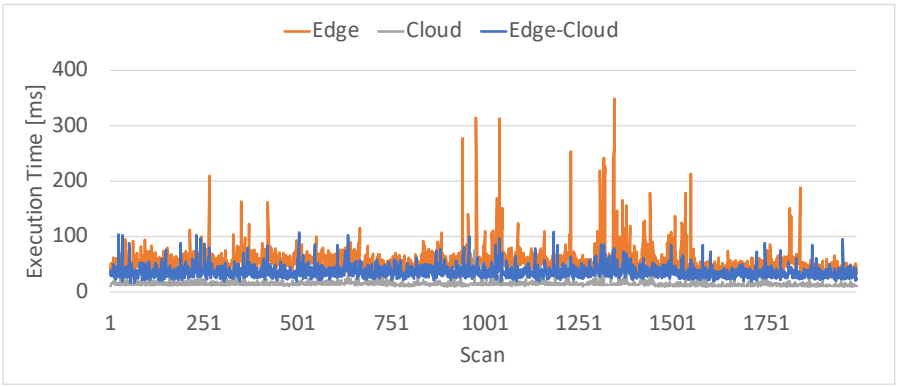
C. NDT Matching

In NDT matching, the specific parameters must be defined with considering the trade-off between execution time and accuracy. Our parameters of NDT matching are as follows. The voxel size is 1 meter. The maximum step size of Newton's method is 100. Convergence threshold is 0.0001. The input point clouds from these LiDARs are downsampled by `voxel_grid_filter`. When downsampled, input point cloud spaces are divided into a cubic voxel. The size of each cubic voxel, called *leaf size*, is set to 2.0. For parameters of A* planner, the resolution is 0.2, the maximum distance of `grid_map_filter` is 1 meter, and the filter is `points2costmap`. We measure two metrics in NDT matching: execution time and deadline miss ratio.

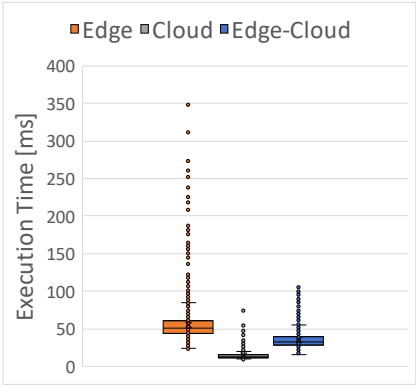
(i) **Execution Time**: We measure the round-trip execution time that takes to complete NDT matching. Fig. 7 shows the round-trip execution time (edge-cloud-edge) in NDT matching, as shown in Fig. 4. We run 2,000 scans of NDT matching for E_{edge}, E_{cloud}, and E_{ec}.

Fig. 8(a) shows the round-trip execution time of every scan. On average, E_{cloud} takes 13.9ms and E_{edge} takes 55.4ms, while E_{ec} takes 35.1ms, 57% faster than E_{edge}. In Fig. 8(b), the worst case execution time of E_{edge} is 348ms and that of E_{ec} is 103ms. This indicates that the presented edge-cloud computing model reduces the execution time compared to the edge computing model and slightly increases the execution time compared to the cloud computing model.

On E_{ec}, only 2 of 2,000 scans are longer than 100ms. This implies that even with unpredictable traffic cost, the presented edge-cloud computing model has more clustered

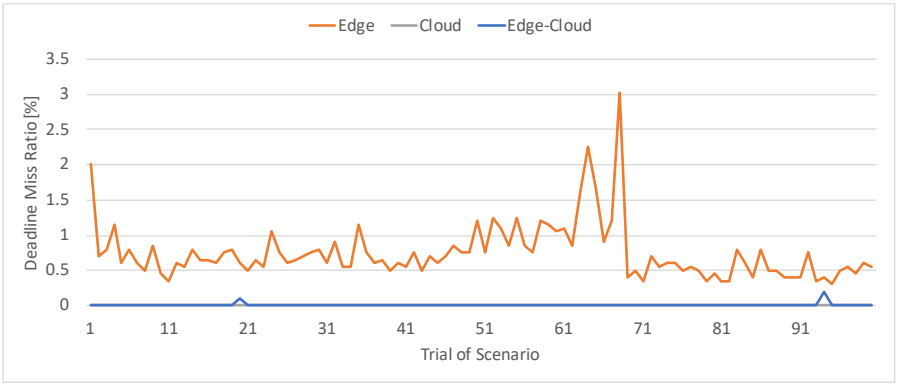


(a) Individual execution time

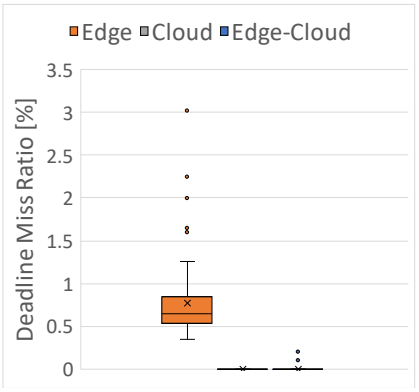


(b) Execution time box plot

Fig. 8. Round-trip execution time of NDT matching



(a) Individual deadline miss ratio



(b) Deadline miss ratio box plot

Fig. 9. Deadline miss ratio of NDT matching

execution time than the edge computing model. The existence of outlier is fatal in NDT matching because the accuracy of the algorithm heavily relies on the previous data. When the execution time of a certain scan is longer than 100ms, the next scan cannot use the previous data and make self-localization less accurate and longer to compute, which leads to another overload in NDT matching. This supports the fact that some of the spikes in Fig. 8(a) are clustered.

(ii) **Deadline Miss Ratio**: At the same time, we measure the deadline miss ratio of NDT matching as the following definition.

$$Deadline\ Miss\ Ratio = 1 - \frac{N_{pp}}{N_{vel}}, \tag{1}$$

where N_{pp} is the number of message `pure_pursuit` received and N_{vel} is the number of message `velodyne_nodelet_manager` sent.

The deadline miss ratio can be derived from the overview of the message queue. Since the message queue size of the every topic is set to 1, and `velodyne_nodelet_manager` tries to send new data every 100ms. If the component of `ndt_matching` misses its deadline (100ms), it does not fetch the latest data from `velodyne_nodelet_manager`. This causes the overflow of the queue when `velodyne_nodelet_manager` tries to send the data in the next scan. Thus, the number of queue overflows equals to the deadline miss ratio. One scenario runs 2,000 scans like Fig. 8(a) as one trial. We run 100 trials of the scenario and measure the deadline miss ratio in each trial. Therefore, the total number of measurements is 200,000 (= 2,000 scans * 100 trials).

Fig. 9(a) shows the deadline miss ratio of each model. Out of 100 trials, E_{ec} achieves 98 trials without any deadline miss, which is almost the same as E_{cloud}. Meanwhile, E_{edge} has the maximum of 3% deadline miss ratio and every set has at least one deadline miss. As Fig. 9(b) shows, the best case in E_{edge} is 0.35%, which is worse than the worst case in the edge-cloud computing model (0.20%). This experiment ensures us the stability in the execution time of the presented edge-cloud computing model. Only 6 of 200,000 measured are longer than 100ms deadline, which puts the success ratio (N_{pp}/N_{vel}) of the presented edge-cloud computing model as 99.997%.

V. Related Work

Edge-cloud computing is an uprising field due to the widespread use of IoT and mobile devices. One of the well-known applications is mobile edge computation offloading

(MECO) [16], which is used in mobile applications. Mobile devices have constraints in their computation power of mobile devices and their battery life. Many applications, depending on their uses, offload some of their computation on to their servers over the internet. There are many optimization methods introduced which parts of the application are offloaded to the cloud. Mao et al. focused on energy consumption and computation power [17], while Huang et al. focused on network latency and computation power [18].

Kumar et al. focused on the edge-cloud computing model for autonomous vehicles [4]. They introduced an ROS-based cloud-assisted system, Carcel, and conducted demonstrated experiments with a golf cart and six iRobot Create robots. Carcel collects data by the UDP connection from the vehicle and sensors placed in multiple locations, analyzes the data, calculates the path using RRT* algorithm [19] at a cloud server, and returns the result to the vehicle. Carcel reduced the average time to detect obstacles such as pedestrians by 4.6 times compared to contemporary systems without access to the cloud. While Carcel requires multiple sensors as roadside infrastructures, our system does not require any additional sensors other than map data and sensors on the vehicle, and hence its use is flexible.

Another distributed model, CVN, is a model where multiple vehicles create a wireless link network to communicate with each other, called VANETs [5]. Through VANETs, a vehicle can share map data and computational resources to improve vehicle infrastructures, or a vehicle can share its positions to other vehicles to improve inter-vehicle communication and safety of passengers. Since VANETs require multiple vehicles, they are difficult to use when non-autonomous vehicles are involved, especially when a heavy vehicle between vehicles leads to frequent disconnection problems called shadowing [20]. Unlike VANETs, this paper focuses on improving standalone autonomous vehicles by utilizing scalable cloud servers to compute heavy tasks.

VI. Conclusion

We presented the edge-cloud computing model for autonomous vehicles. The presented edge-cloud computing model assigns ROS components to edge devices or cloud servers, respectively, in order to satisfy the requirement of the execution time and communication cost. We use 1 Gbit/s simulated network as an alternative of 5G mobile network. The simulation results by ROSBAG data demonstrate that the presented edge-cloud computing model reduces the execution time and deadline miss ratio compared to the edge computing model and slightly increases them compared to the cloud computing model.

In future work, we will investigate the behavior of the presented edge-cloud computing model in actual 5G mobile network, especially focusing on the schedulability analysis considering communication delay and losses that would occur under mobile networks are desirable. In addition, we will add the cloudlet (micro datacenters in close to proximity to edge devices) [21] to the presented edge-cloud computing model for more realistic use cases.

References

[1] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, and M. N. C. et al., "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[2] B. Fildes, M. Keall, N. Bos, A. Lie, Y. Page, C. Pastor, L. Pennisi, M. Rizzi, P. Thomas, and C. Tingvall, "Effectiveness of low speed autonomous emergency braking in real-world rear-end crashes," *Accident Analysis & Prevention*, vol. 81, pp. 24–29, 2015.

[3] SAE International, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," SAE International, Warrendale, PA, Standard, 2014.

[4] S. Kumar, S. Gollakota, and D. Katabi, "A Cloud-Assisted Design for Autonomous Driving," in *Proceedings of the First Edition of the MCC workshop on Mobile Cloud Computing*. ACM, 2012, pp. 41–46.

[5] H. Hartenstein and K. P. Laberteaux, *VANET: Vehicular Applications and Inter-Networking Technologies*. John Wiley & Sons, 2010.

[6] International Telecommunications Union, "Minimum requirements related to technical performance for IMT-2020 radio interface(s)," International Telecommunications Union, Geneva, Switzerland, Tech. Rep., 2017.

[7] ——, "Requirements related to technical performance for IMT-Advanced radio interface(s)," International Telecommunications Union, Geneva, Switzerland, Tech. Rep., 2008.

[8] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[9] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An Open Approach to Autonomous Vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *Proceedings of the International Conference on Robotics and Automation Workshop on Open Source Software Workshop on Open Source Software*, vol. 3, 2009, pp. 1–6.

[11] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," in *Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems*. IEEE, 2018, pp. 287–296.

[12] H. Chishiro, K. Suito, T. Ito, S. Maeda, T. Azumi, K. Funaoka, and S. Kato, "Towards Heterogeneous Computing Platforms for Autonomous Driving," in *Proceedings of the 15th IEEE International Conference on Embedded Software and Systems*. IEEE, 2019, pp. 1–8.

[13] The Autoware Foundation, https://www.autoware.org/.

[14] P. Biber and W. Strasser, "The Normal Distributions Transform: A New Approach to Laser Scan Matching," in *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3. IEEE, 2003, pp. 2743–2748.

[15] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[16] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[17] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[18] D. Huang, P. Wang, and D. Niyato, "A Dynamic Offloading Algorithm for Mobile Computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, 2012.

[19] S. Karaman and E. Frazzolis, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[20] M. Boban, T. T. V. Vinhoza, M. Ferreira, J. Barros, and O. K. Tonguz, "Impact of Vehicles as Obstacles in Vehicular Ad Hoc Networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 1, pp. 15–28, 2011.

[21] M. Satyanarayanan, "The Emergence of Edge Computing," *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.