Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6631111, 18 pages
https://doi.org/10.1155/2021/6631111

Research Article

Cloudroid Swarm: A QoS-Aware Framework for Multirobot Cooperation Offloading

Yuanzhao Zhai, Bo Ding, Pengfei Zhang, and Jie Luo

College of Computer, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Bo Ding; dingbo@nudt.edu.cn

Received 7 December 2020; Revised 4 April 2021; Accepted 17 May 2021; Published 21 June 2021

Academic Editor: Paul Honeine

Computation offloading has been widely recognized as an effective way to promote the capabilities of resource-constrained mobile devices. Recent years have seen a renewal of the importance of this technology in the emerging field of mobile robots, supporting resource-intensive robot applications. However, cooperating to solve complex tasks in the physical world, which is a significant feature of a robot swarm compared to traditional mobile computing devices, has not received in-depth attention in research concerned with traditional computation offloading. In this study, we propose an approach named *cooperation offloading*, which offloads the intensive communication among robots as well as the computation for compute-intensive and data-intensive tasks. We analyze the performance gain of cooperation offloading by formalizing multirobot cooperative models; in addition, we study offloading decisions. Based on this approach, we design a cloud robotic framework named *Cloudroid Swarm* and develop several QoS-aware mechanisms to provide a general solution to cooperation offloading with QoS assurance in multirobot cooperative scenes. We implement Cloudroid Swarm to transparently migrate multirobot applications to cloud servers without any code modification. We evaluate our framework using three different multirobot cooperative applications. Our results show that Cloudroid Swarm can be applied to various robotic applications and real-world environments and bring significant benefits in terms of both network optimization and task performance. Besides, our framework has good scalability and can do support as many as 256 robot entities simultaneously.

1. Introduction

The idea of offloading computation from resource-constrained devices to external platforms (e.g., the cloud) emerged in the field of mobile computing due to the limited computational power, storage, and energy of the mobile device [1]. More recently, the idea has also grown in popularity in the mobile robot community because achieving "autonomy" on a mobile robot usually involves intensive or even highly paralleled computing, which can easily exceed the resources available to robots in their onboard computers [2]. The exceptional benefits of introducing computation offloading, such as enhancing processing capabilities and speeding up application execution, have achieved great success in various robotic tasks such as simultaneous localization and mapping (SLAM) [3], object recognition, and grasp planning [4].

It is common in the robotic field to use a group of robots to handle complex real-world tasks that extend a single robot's capabilities. Compared to traditional mobile computing devices, a distinguishing characteristic of mobile robots in the robot swarm is that they are inherently cooperative. Robots usually need to exchange intensive data with each other to complete tasks. In these tasks, the overall performance of the robot team can be greatly enhanced by cooperation among team members. However, data-intensive tasks usually imply substantial communication costs. With the increasing group size, the large volumes of data exchange between robots may become the bottleneck of a multirobot application, a problem that does not exist among traditional mobile computing devices (e.g., smartphones or IoTs). Continued application of computation offloading to the robot swarm does not alleviate the communication problem between robots and introduces additional communication between robots and the cloud. In this situation, the benefit of computation offloading would be counterproductive.

For example, multirobot SLAM, one of the representative robotic applications, is a compute-intensive and data-intensive task. SLAM aims to perform real-time localization and mapping "simultaneously" with a sensor (e.g., LiDAR or depth camera) moving through an unknown environment without any exogenous means of the location. Multirobot SLAM is an extension of single-robot SLAM in terms of parallel and distributed processing. As shown in Figure 1, each robot in the robot swarm processes its own localization and local map construction in an unknown environment. Then, all local maps are merged cooperatively for the global map. In the traditional setup in Figure 1(a), each robot executes the computationally intensive SLAM algorithm locally and independently, and the local maps from each robot are exchanged periodically to merge with the ever-increasing global map. The communication cost of the latter increases either when the size of the robot team increases or during attempts to improve the timeliness of the global map; this causes poor quality of service (QoS) such as high latency or even no response. In our experiments, to build a midsize indoor map with four robots, the communication among robots occupies up a bandwidth up to 40-80 Mb/s, which is unacceptable for most common robot swarms. With the traditional computation offloading method in Figure 1(b), we independently migrate the local map-building process on each individual robot to cloud servers. It means that firstly, the local sensor data needs to be sent to the cloud servers, and then, the map data needs to be transferred back to the robot from the server and finally exchanged among robots as before. Computation offloading doubles the bandwidth consumption and neutralizes the benefit of introducing computation offloading in this situation.

Therefore, we cannot simply offload the computation of each individual in a multirobot cooperative application. We argue that offloading the communication inside the robot swarm is equally vital to improving cooperative tasks' efficiency in a "cloud + robot" architecture. This reasoning is inspired by the simple idea that, because computationally intensive modules can be migrated to the cloud, the possibility exists to offload the considerable quantity of data transmission generated by robot cooperation to the cloud as well. This solution would promote cooperation efficiency by utilizing the high-bandwidth network inside the cloud platform instead of the local low-bandwidth wireless network. As shown in Figure 1, by offloading the cooperation among robots to the cloud servers, not only can the localization and mapping processes be performed on the servers, but also the output maps can be transferred within the cloud to another computation module in need. The bandwidth is less likely to be overoccupied in this situation, thereby improving the cooperation efficiency in generating global maps. Another problem in computation offloading is that the offloading performance would deteriorate due to low data rates if too many mobile users choose to offload their tasks via the same wireless access channel simultaneously. QoS is one of the most important factors to consider for robotic applications because these applications interact directly with the physical world [5]. So offloading decisions and some additional mechanisms should be studied to adopt such a new approach to boost cooperation with QoS assurance in robot swarms.

In this study, to address the challenge mentioned above, we introduce the concept of *cooperation offloading*. Cooperation offloading is a new offloading approach for multirobot cooperative tasks, which treats the cooperative robot swarm as an entirety when offloading by taking the factor of cooperation among robots as well as computation into account. It can offload the original communication and cooperation in robot swarms instead of introducing additional cooperation for computation offloading [6], making the cooperative system more efficient. The overall goal of our work is to propose a general solution that would enable existing multirobot cooperative tasks to be executed more efficiently by using the cooperation offloading method and that would satisfy the QoS requirement of the robots at the same time.

In summary, the key contributions of this paper are as follows.

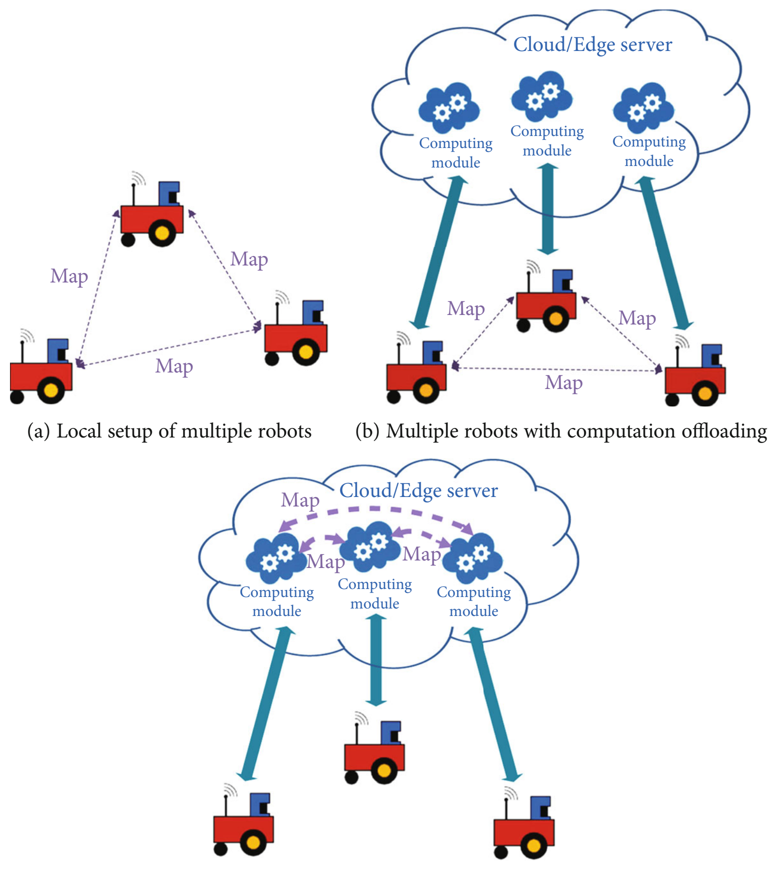
(i) We propose cooperation offloading, which offloads cooperation in robot swarms to the cloud servers. We calculate the time cost of local computing, computation offloading, and cooperation offloading by formalizing the multirobot cooperative architecture model. Then, we study the performance gain contributed by cooperation offloading and provide offloading decisions

(ii) We design a cloud-based robotic framework, named *Cloudroid Swarm*. Cloudroid Swarm performs cooperation offloading in addition to computation offloading, utilizing the high-speed network infrastructure in the cloud. To assure QoS in Cloudroid Swarm, we propose a distributed link detection algorithm at the local level and link capacity adjustment at the global level to adapt to the poor and dynamic network environment

(iii) We implement Cloudroid Swarm to support cooperation offloading for multirobot applications and transparently migrate multirobot tasks to the cloud servers without any code modification. We also propose several effective mechanisms to improve the QoS and scalability of Cloudroid Swarm

(iv) We investigate the performance of the proposed QoS-aware cooperation offloading framework by evaluating three different multirobot applications in both simulation and real-world environments. The applications include cooperative SLAM, multirobot exploration, and multirobot collision avoidance. The results demonstrate the efficiency of Cloudroid Swarm in terms of network optimization, task performance, and scalability

This article is an extension of our previous conference version [7], which presents a cloud robotic framework that boosts the efficiency of cooperation for multiple robot applications in robot swarms and evaluates the framework using both the public data sets and simulator. However, it is observed that



(a) Local setup of multiple robots (b) Multiple robots with computation offloading

(c) Multiple robots with cooperation offloading

Figure 1: Different configurations of multirobot SLAM. The SLAM task is aimed at simultaneously drawing a map of the surrounding environment and locating the poses of the robots themselves. Two phases, pose localization and mapping construction, are coupled in a typical SLAM procedure. The localization phase requires the map constructed in the previous frame for matching, and the mapping phase needs the pose information in the previous frame for accurate results.

the framework cannot provide physical robots' satisfactory behavior due to many interference factors in the real world, which has unstable input and network environment. Thus, QoS property is required, especially in a highly dynamic and resource-competitive environment. To overcome this difficulty left in the previous work, we propose several QoS-aware algorithms and mechanisms to make our framework more robust. Thus, we can obtain the performance gain even in the real-world mobile robot system. Besides, to demonstrate the proposed QoS-ware framework's excellent performance, we evaluate our framework in extensive experiments, including quantitative analysis of QoS's benefits in the cooperative SLAM application and collision avoidance test conducted on real-world robots.

2. Related Work

2.1. Cloud Computing in Robotics. In the domain of cloud computing, computation offloading is regarded as an effective way to alleviate the constrained resources on mobile phones and Internet of Things (IoT) devices as well as reducing the running cost in mobile cloud computing (MCC) [8] and mobile edge computing (MEC) [9]. In the robotic field, because of the similarity of the computational patterns to those in mobile computing, computation offloading is also demanded by robot tasks. A series of recent research studies have been dedicated to boosting robotic applications. Chen et al. propose the term "Robot as a Service" (RaaS) and present a self-contained unit in the cloud computing environment [10]. However, the development and deployment of robot applications are limited to a certain programming language and architecture (Intel), without the ability to migrate the existing robot software to the cloud. Seminal work in this field is DAvinCi [11], a particle-based SLAM framework for service robots in a large-scale Internet environment. However, it requires the entire process running in the cloud to be deployed and configured manually. Another closely related study is Rapyuta [12], a framework that enables robots to offload their complex computation to the cloud. Rapyuta is a typical clone-based PaaS architecture based on the Linux Container (LXC). To solve Rapyuta's limitations of dynamical deployment for complex tasks and the lack of cloud management tools, our previous work, Cloudroid [13], supports the automatic deployment of existing robotic software packages to the cloud, thus transparently transforming them into Internet-accessible cloud services with QoS assurance. However, both Rapyuta and Cloudroid only focus on one individual robot for computation offloading and our evaluation in Section 6 shows that they are not suitable for a cooperative multiple robot applications.

2.2. Multi-user Computation Offloading. Concerning multiuser computation offloading, one of the major topics to be investigated is the decision of whether mobile users offload their computation task to the cloud or not. There is offloading transmission competition among users because several users may choose the same wireless access channel and offload tasks to the cloud simultaneously.

Some studies adopt centralized approaches [14, 15], which update the offloading decision iteratively to solve joint task offloading and resource allocation in MEC networks. Chen [16] demonstrates that it is NP-hard to optimal multiuser computation offloading solutions in a multichannel wireless interference environment, and hence proposes a game-theoretic approach for achieving efficient computation offloading in a distributed manner. Considering social and behavioral characteristics of users in the overall computation offloading process, Apostolopoulos et al. [17] exploit prospect theory instead to account for users' risk-seeking and loss-aversion behavior in offloading decisions. All the aforementioned methods are limited by the trade-off between optimality and computational complexity.

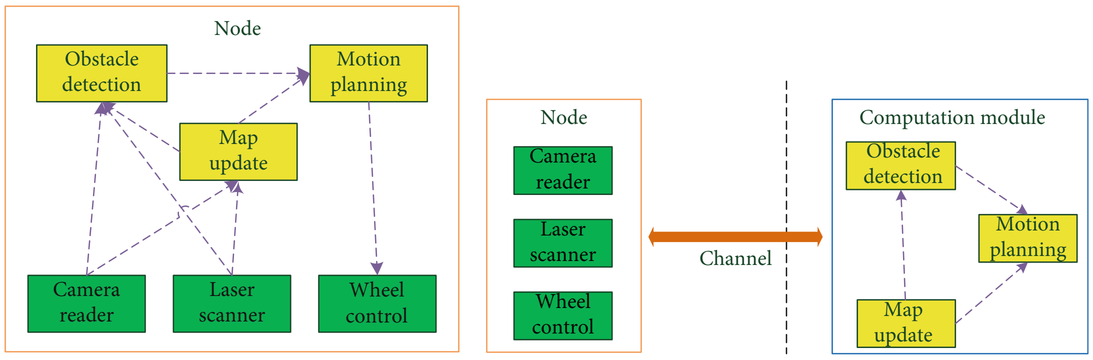
Deep learning shows excellent potential in the field of wireless communications to deal with multiuser task offloading decisions [18]. Wu et al. [19] propose a distributed deep learning-driven task offloading for collaborate edge and cloud computing, where multiple parallel DNNs are used to generate offloading decisions. Then the offloading decision with the lowest system utility is chosen as the output and the label to train deep neural networks. To characterize long-term computation offloading performance, Dinh et al. [20] propose a distributed model-free reinforcement learning offloading mechanism, which reaches 87.87% payoffs compared to the optimal condition. Since security is one of the critical issues in mobile edge computing and mobile edge computing, Huang et al. [21] propose a security and cost-aware computation offloading strategy based on the popular deep reinforcement learning approach, deep Q-network. These distributed deep learning methods assume that the mobile device has sufficient computing capability to compute and obtain the offloading decision in real time. However, mobile robots usually have limited computing and communicating capabilities to satisfy this assumption.

Besides, none of these studies consider the cooperative tasks that need data-intensive communication among users, which are common in robot swarms. What is more, all of their evaluations are carried out in a simulation environment. As we know, real-world experiments would be affected by more interference factors and omnipresent uncertainty, thus requiring stricter QoS requirements. In this paper, we propose a link capacity adjustment algorithm to ensure QoS of multiuser computation offloading, which is proven to be effective in real-world multirobot resource competition environment.

2.3. Multi-robot Cooperation. The architecture of multirobot cooperative applications has been studied for years. One of the crucial research topics is the communication model, which indicates the data exchange pattern used in multirobot systems [22]. It is demonstrated that the minimum amount of network consumption in a swarm of n robots where there is communication between any two robots could be $O(n^{1.5})$, which is not linearly scalable when the number of entities increases [23]. Cloud-based studies are carried out to avoid this limitation. In [24], robots are grouped into different clusters. Communication between different clusters is promoted via the cloud, whereas the direct local transmission method is used for internal communication within each cluster. However, because of the mobility of robots, maintaining the cluster division is complicated, and it is also difficult to determine the boundary between local and cloud methods. A cloud-based research using multiple low-cost robots is proposed in [3]. This approach leverages the Rapyuta [12] robot framework, and all the data exchange occurs in the cloud. However, it is a task-specific solution that can only be applied to a specific 3D mapping task. Chen et al. propose a framework of robotic cooperation on computation offloading to enable both robot–robot and robot–cloud cooperation [25]. By implementing a method that is different from all the abovementioned solutions, we utilize advances both in the local and the cloud computing environments to complete different kinds of cooperative tasks of robots. Identifying and optimizing communication bottlenecks exploit the large bandwidth inside the cloud while maintaining the benefit of flexibility in the original local network environment.

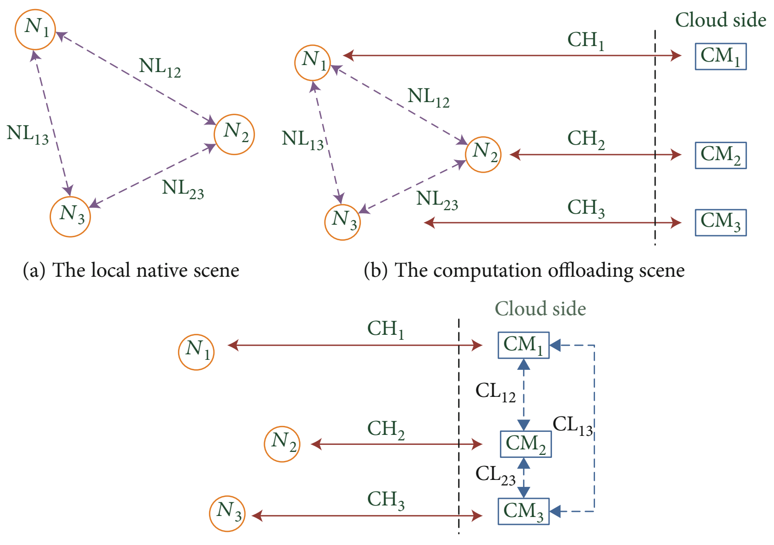
3. Cooperation Offloading Decision

3.1. Multi-robot Cooperative Models. Because our method targets decentralized multirobot applications, we utilize the widely used Publish-Subscribe model, which is also the main messaging pattern in Robot Operating system (ROS) (https://www.ros.org/), as the foundation of our formalization. A task is assigned to different processing units inside robots, in the form of processes running on the onboard computer system. We denote each process as an *operator*. As shown in Figure 2(a), we classify the operators into two categories. The nonmigratable operators directly interact with the physical world, e.g., laser scanner, camera reader, and the velocity controller of moving wheels. And the migratable operators do not directly interact with the peripheral device, typically performing intensive computing. The communication pattern between operators is based on topic. One operator can publish messages on a specific topic, whereas all other operators that subscribe to this topic will receive the message. When the case comes to computation offloading, the migratable operators are safely transferred to the cloud for computing acceleration and are wrapped into a computation module. The local robot node and computation module are connected with the dedicated channel for data exchange. Though publishers and subscribers are at the two ends, the channel has the ability to forward the message across



(a) The local native scene (b) The computation offloading scene

FIGURE 2: We illuminate a node for the navigation application on a wheeled mobile robot as an example. The nonmigratable operators, which handle the input of sensors and the control of wheels, are depicted in green. The migratable operators are depicted in yellow. When the node comes to the computation offloading case, the yellow migratable operators are transferred to the cloud and communication with the local operators by the full-duplex channel.



(a) The local native scene (b) The computation offloading scene

(c) The cooperation offloading scene

FIGURE 3: Different models of multirobot cooperative scenes.

the network between the two sides transparently. As the illumination in Figure 2(b), offloading is achieved by deployment configuration with keeping source code unmodified.

As shown in Figure 3, we model decentralized multirobot architectures, where n robots perform the cooperative task. Figure 3(b) depicts a multirobot computation offloading scenario. The migratable operators are safely transferred to the cloud (we do not explicitly differentiate between edge servers and remote cloud servers because edge servers can be modeled as cloud servers with lower latency but thinner computing resources) for enhancement and are wrapped into a single *computation module*. The local robot node and computation module are connected with the dedicated *channel* for data exchange. Note that data for cooperation in the robot swarm still need to be exchanged in local data links. Figure 3(c) illustrates our proposed cooperation offloading model, which adds shortcut links between computational modules. These robots make a graph with n nodes, where the node for the ith robot is N_i. From every node N_i to another N_j, there exists a local data link, which is denoted as NL_{ij}. In the case of computation offloading for a multirobot arrangement, the cloud side computation module of N_i is denoted as CM_i, and the channel connecting N_i and CM_i is CH_i. In the case of cooperation offloading, the additional cloud link between CM_i and CM_j is CL_{ij}.

3.2. Time Cost for Offloading Decisions. Besides the characteristics of computation offloading, cooperation offloading has its unique features, such as high communication costs among robots as well as between robots and servers, which bring new challenges to offloading decisions. To simplify the estimation without loss of generality, we consider the following particular situation:

(i) There exists only one topic in the robot swarm

(ii) All m computation modules publish on the topic and continuously send messages

(iii) All m computation modules subscribe to this topic and continuously receive messages emitted from all other $m-1$ computation modules

(iv) For a robot swarm, either all tasks are performed locally or all of them are offloaded to servers. We propose this assumption because cooperation offloading considers cooperative tasks as an entirety

First, we consider the local situation in Figure 3(a). Let s^m be the computing speed of mobile robots and s^s be the computing speed of servers. Each robot i needs to perform part of a cooperative computationally intensive task denoted by $T_i = (d_i, h_i, x_i)$, where d_i is the data size of the migratable part of T_i, h_i is the number of CPU cycles, and x_i is the amount of data exchange with other robots for the robot i to complete the entire task. Suppose all robots have the same communication ability and R_{NL} is the communication rate between local robots. Thus, the computing time for the T_i of local computing is h_i/s^m and the cooperative time is $(n \times x_i)/R_{\mathrm{NL}}$. And the total time cost of local computing for robot i can be expressed as

$$t_i^{\text{local\_computing}} = \frac{h_i}{s^m} + \frac{n \times x_i}{R_{\mathrm{NL}}}. \tag{1}$$

In Figure 3(b), let R_{CH_i} be the channel's communication rate between robot i and its computation module in the cloud side. If we simply apply computation offloading to each robot independently, although the computing time is reduced to h_i/s^s, the cooperative time is the same as before, and the additional transmission delay between robot i and the cloud is $(d_i + x_i)/R_{\mathrm{CH}_i}$.

Note that we consider both the upload and download delays here. Then, the total time cost of multirobot computation offloading is

$$t_i^{\text{computation\_offloading}} = \frac{h_i}{s^s} + \frac{n \times x_i}{R_{\mathrm{NL}}} + \frac{d_i + x_i}{R_{\mathrm{CH}_i}}. \tag{2}$$

Let $d_{\max}$ be the maximum value of $\{d_1, d_2, \cdots, d_n\}$, and let the other variables have similar maximum definitions. Taking a global view of the entire task, the time cost depends on the last robot to complete the execution. Without loss of generality, we assume that robot i entails processing the largest amount of data and requires the most data to exchange and is thus the most time-consuming. Thus, traditional computation offloading for robot swarm improves performance when $t_{\max}^{\text{local\_computing}} > t_{\max}^{\text{computation\_offloading}}$:

$$\frac{h_i}{s^m} + \frac{(n \times x_i)}{R_{\mathrm{NL}}} > \frac{h_i}{s^s} + \frac{(n \times x_i)}{R_{\mathrm{NL}}} + \frac{d_i + x_i}{R_{\mathrm{CH}_i}} \Leftrightarrow h_i \times \left(\frac{1}{s^m} - \frac{1}{s^s}\right) > \frac{d_i + x_i}{R_{\mathrm{CH}_i}}. \tag{3}$$

We argue that (3) is difficult to satisfy in most situations, especially in the event of intensive communication between robots (large x_i), and many robots compete for the offloading resource in a poor network environment (small or unstable B_{CH_i}). Considering the situation that $h_i/s^m < (d_i + x_i)/R_{\mathrm{CH}_i}$, even if the processing speed of the server is infinitely large (i.e., $s_s \to +\infty$), computation offloading results in counterproductive performance. With cooperation offloading, the use of communication shortcuts in the cloud side in Figure 3(c) would enable messages to be exchanged directly via the cloud links. The communication speed inside the cloud is extremely high, such that the cooperative time inside the cloud can be ignored. Furthermore, it is unnecessary to send the data for communication x_i back to the local side for cooperation. So we can ignore the download delay here. Thus, the total time cost of multirobot cooperation offloading is

$$t_i^{\text{cooperation\_offloading}} = \frac{h_i}{s^s} + \frac{d_{\max}}{R_{\mathrm{CH}_i}}. \tag{4}$$

Compared to (2), we find that cooperation offloading can reduce the time cost by reducing the communication data among robots in our scenario. The cooperation offloading decisions can be made by $t_{\max}^{\text{local\_computing}} > t_{\max}^{\text{cooperation\_offloading}}$:

$$\frac{h_i}{s^m} + \frac{(n \times x_i)}{R_{\mathrm{NL}}} > \frac{h_i}{s^s} + \frac{d_i}{R_{\mathrm{CH}_i}} \Leftrightarrow h_i \times \left(\frac{1}{s^m} - \frac{1}{s^s}\right) > \frac{d_i}{R_{\mathrm{CH}_i}} - \frac{(n \times x_i)}{R_{\mathrm{NL}}}. \tag{5}$$

We can define the difference between the two sides of the inequality as the performance gain. Then, we can learn that cooperation offloading achieves more performance gain than computation offloading under the same conditions in multiuser offloading scenes from (3) and (5). On the other hand, we can monitor B_{CH_i} in real time using the method proposed in Section 4.1. When the condition of (5) holds, which is more easily established than (3), it is a sensible option to use cooperation offloading to improve the system performance (line 2 in Algorithm 2).

4. QoS-Aware Framework

To enable the cooperation offloading, we propose a framework named Cloudroid Swarm, which leverages the network environment inside the cloud to support communication between computation modules. Built on the foundation of Cloudroid [13], Cloudroid Swarm still exploits the splitting computation module for each robot. The new components designed for each robot are the *network module* in the cloud side and the *network operator* inside local robots. For the entire application, the main improvement is the establishment of the *topology engine*, the control plane for task-wide cooperation, as shown in Figure 4.

Network module: for each robot participating in the application, a network module termed NM_i runs on the cloud side and is launched along with the corresponding computation module CM_i. Its primary responsibility is to handle communication resulting from cooperative multirobot computation offloading. NM_i, in place of CM_i, is directly

```
Input: Time interval between two successive detections: $T$
  Width of sliding window: $W$
Output: Communication rate, $R^{(t)}$ and latency, $lat^{(t)}$ of the channel at time t
  initialization: $echo \Leftarrow \emptyset, B^{(t)} = 0, flying\_time^{(t)} = 0$
1:  while $Src$ is not shut down do
2:      Emit a new message from the source end to the destination end
3:      for each $msg$ returned at $[t - T, t)$ do
4:          $echo_i \Leftarrow echo_i \cup \{msg\}$
5:          $flying\_time^{(t)} \Leftarrow flying\_time^{(t)} + RRT(msg)$
6:      end for
7:      for each $msg$ returned at $[t - W - T, t - W)$ do
8:          $echo_i \Leftarrow echo_i \setminus \{msg\}$
9:          $flying\_time^{(t)} \Leftarrow flying\_time^{(t)} - RRT(msg)$
10:     end for
11:     $R^{(t)} \Leftarrow \beta * sizeof(echo_i)$
12:     $lat^{(t)} \Leftarrow flying\_time^{(t)} / sizeof(echo_i)$
13:     $t \Leftarrow t + T$
14: end while
```

Algorithm 1: Sliding-window algorithm.

```
Input: Collection of robots using the same wireless access to offload: $\mathcal{N} = \{N_1, N_2, \cdots, N_m\}$
        Load value of channels at every timestamp: $load_{CH}^{(t)}$
        Global control parameter: $\lambda$
Output: The offloading strategy and capacity value of channels at the next timestamp: $cap_{CH}^{(t+T)}$
1:  for each time interval $T$ do
2:      Choose the cooperation offloading strategy
3:      while The task is not completed do
4:          $lat_{CH}^{(t)}, R_{CH}^{(t)}, R_{NL}^{(t)} \Leftarrow$ Sliding-window Algorithm
5:          if Inequality (5) is True then
6:              Choose the cooperation offloading strategy
7:          else
8:              Choose the local computing strategy
9:          end if
10:         if Cooperation Offloading then
11:             for each $N_i \in \mathcal{N}$ do
12:                 $demand \Leftarrow (load_{CH_i}^{(t)} - cap_{CH_i}^{(t)})$
13:                 $weight \Leftarrow \exp(lat_{CH_i}^{(t)}/\lambda) / \sum_{k=1}^{n} \exp(lat_{CH_k}^{(t)}/\lambda)$
14:                 $cap_{CH_i}^{(t+T)} \Leftarrow cap_{CH_i}^{(t)} + demand * weight$
15:             end for
16:         end if
17:         $t \Leftarrow t + T$
18:     end while
19: end for
```

Algorithm 2: QoS-aware link capacity adjustment algorithm.

connected with channel CH$_i$ to intercept all the network messages from/to it. In addition, because of its network awareness, NM$_i$ can sense other network modules and send messages directly to them via cloud links inside the cloud.

Network operator: in Cloudroid Swarm, communication on the robot side is handled by the network operator, with a similar function to the network module in the cloud side. This kind of operator, which is directly connected with the channel and node links, acts as the bridge between the other operators inside the robot and the outside world. Similar to nonmigratable operators, network operators are also processes in robot nodes. But network operators have special characters that communicate with other cloud components, including topology engines and other modules in the cloud.

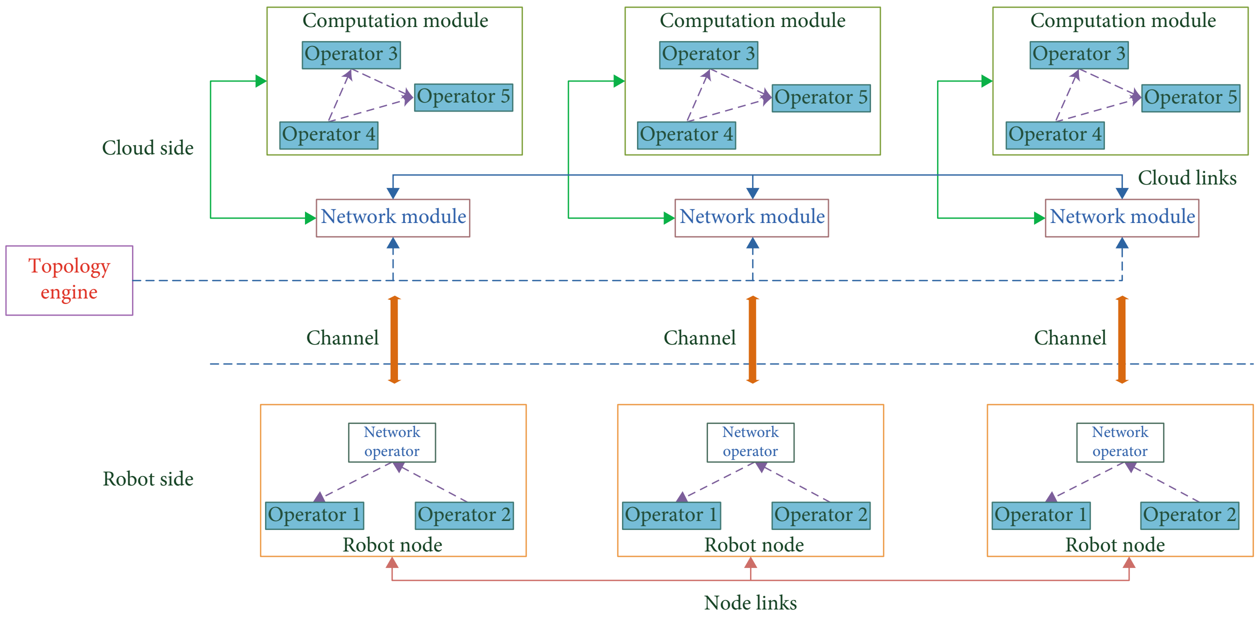


Figure 4: Architecture of Cloudroid Swarm

Topology engine: the topology engine acts as the coordinator on the cloud side, processing the global topological information of the entire task. It is designed to be started along with the launch of the application routine and maintain interaction between the network modules and network operators. With the real-time metrics of links, the global topology engine performs the global planning method to control the capacity, which can control the message flow in dynamic equilibrium.

A notable challenge in cooperation offloading is the uncertainties it introduces, which influence the specific QoS properties of the multirobot applications. Therefore, the architecture must be designed to include real-time monitoring of the capacity of links and global network planning to manage the behavior of the computation modules for improved performance. Based on the components mentioned above, we design a set of QoS awareness mechanisms on the robot, cloud, and network topology to minimize the impact of uncertainties caused by poor network conditions or resource competition. Note that the client-side and cloud-side QoS mechanisms are described in detail in our previous paper [13], and in the current paper, we mainly introduce the key mechanisms of the network side. Scheduling and optimization of communication at the scale of the application can hardly drive the optimal solution while maintaining real-time performance because these functions are involved in the interaction with dynamic workload and network conditions. However, with deep insight into this distributed optimization problem, we can split the entire optimization approach into two different levels:

(i) On the local level for each robot, the network module and network operator can work collaboratively to determine the path to forward the message and detect the link status

(ii) On the global level, even though applications are a black box from the perspective of the cloud platform, the behavior of message transmission can be inspected by the network stack. Then, they can be scheduled by modifying the capacity of the links

4.1. Link Detection.

For all the channels between robots and their corresponding computational modules, we introduce three variables to describe the communication quality of link L: (1) $\text{load}_L^{(t)}$, for measuring the current transmission load at timestamp t; (2) $\text{lat}_L^{(t)}$, indicating the transmission latency of the message at timestamp t; and (3) $\text{cap}_L^{(t)}$, which represents the capacity of the link. These variables are measured continuously during the entire task. It should be pointed out that it is difficult to obtain the exact value at time t, and the average from $t - T$ to t is used alternatively in most cases.

Because the wireless network conditions can change dramatically with the movement of the robot, the bandwidth and $\text{lat}_L^{(t)}$ of channels vary from time to time. Estimating these time-related variables is essential for planning the route of messages to improve the performance. Though it is impossible for a central coordinator to sense the message flow in each data link in exact real time, it is still feasible to periodically acquire the communication metric at a suitable time interval with the collaboration of network operators on node N_i and network module NM_i. In our scenario, we are only concerned with the end-to-end parameters of the data links. Therefore, we apply a sliding-window method for detection, which has a negligible impact on the network.

As shown in Figure 5, at the beginning of each time interval T, the source end emits a message with a fixed size to the other end of the link. Upon receiving this message, the destination end echoes the same message back to the source. The round trip time is recorded by the source module as $\text{RRT}(\text{msg})$. Simultaneously, a sliding window with a fixed length of W is maintained, which spans from $t - W$ to t. We can count the

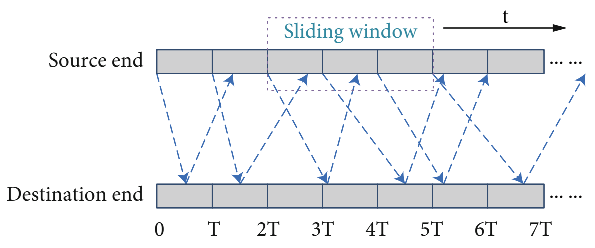


Figure 5: The sliding-window method for link detection.

number of echo messages in the window to measure the bandwidth, and average the round trip time to calculate $\mathrm{lat}_L^{(t)}$. The time interval T used for detecting capability and performing a capacity assignment for each link is set to 200 ms. This value is determined by various experiments on real-world applications to obtain more precise metrics and execute more fine-grained control to the data links, yet with negligible impact on the entire network. The whole algorithm is shown in Algorithm 1.

4.2. Global Link Capacity Adjustment. The added network components enable direct data exchange inside the cloud with the sacrifice of increasing the traffic in the channels between robots and servers. The latter is more likely to become a bottleneck because of constrained bandwidth and competition among robots that choose the same wireless access route to offload. So we need to carefully adjust their capacity to guide the flow and prevent the bottleneck from occurring. We design the global capacity assignment policy deployed at the topology engine and dynamically adjust the $\mathrm{cap}_L^{(t)}$ of channels in each timestamp, thereby scheduling message flows. The policy is based on the following principles:

(i) When a new operator is launched at one end of a link, a larger number of messages are transferred via this link. In this case, the capacity of this link tends to increase

(ii) The source module sends a large number of messages in a short period, and then, it recovers to the previous state. We argue that these conditions indicate an emergency signal arising from the source module, for example, the obstacle encountered by a wheeled robot or passages in front of an autonomous vehicle. This situation should be quickly reflected in the capacity

(iii) The number of messages on one or more existing topics is increased. In this situation, the capacity of the relational data links should be increased for long-term governance

Based on the above analysis, we design a QoS-aware distributed link capacity adjustment algorithm, which is presented in Algorithm 2. For each task, we choose the cooperation offloading strategy by default. Then, we detect link status using the sliding-window method described in Section 4.1. If bandwidth is too low to satisfy the cooperation offloading decision, the system will choose the local computing strategy to guarantee the basic QoS. Otherwise, we choose the cooperation offloading to boost cooperation and adjust the capacity of the channels between robots and the cloud. The topology engine executes the policy routine proposed above and sends the results back to the local component for assigning capacity in the next timestamp (from line 11 to line 15 in Algorithm 2).

4.3. Computational Complexity. We provide the theoretical analysis of the computational complexity of the proposed algorithm. Algorithm 1, the sliding-window algorithm, is distributed on each mobile robot. So the computational complexity of the sliding-window algorithm does not scale with the group size, n (i.e., $O(1)$ complexity).

Algorithm 2 has two main steps, which are cooperation decision making and link capacity adjustment. Since we assume that all tasks are either performed locally or offloaded to servers for a robot swarm, we can obtain the optimal cooperation offloading decisions by comparing the maximum total time cost of local computing and cooperation offloading. Then, we adjust the capacity of channels between robots and the cloud if using cooperation offloading. The computational complexity of both steps increases at a linear rate $O(n)$ of the group size. So the computational complexity of the QoS-aware link capacity adjustment algorithm is $O(n)$.

Overall, the computational complexity increases linearly with the increasing number of robots, making our platform have good scalability.

5. Implementation

We implement the prototype of Cloudroid Swarm based on the ROS programming model to adapt to existing multirobot applications. Cloudroid Swarm is an extension of the single-robot-oriented framework, Cloudroid, which can transparently migrate computation-intensive modules to the cloud servers and wrap them as computation modules for enhancement. While exploiting the existing systems, we still devise significant mechanisms to improve the QoS and scalability of Cloudroid Swarm. These improvements include the following:

Topic remapping: to intercept messages from/to computation modules, we utilize the building function of the ROS launching mechanism to remap all the originally subscribed/published topics to new ones. In this situation, the network modules, which are related to both the original and remapped topics, can successfully manipulate and forward the messages that are transferred between the computational module and other components.

Container cluster orchestration: because all modules, including the computation modules, network modules, network operators, and topology engine are self-contained Docker instances, the coordination and management of them are essential for Cloudroid Swarm. In this regard, we adopt a popular open-source container cluster orchestration tool known as Kubernetes (KubeEdge: https://kubeedge.io/). It enables load balance, deployment replication, and elastic consolidations. In addition, we also use KubeEdge (KubeEdge: https://kubeedge.io/) for edge management if our system contains edge servers.

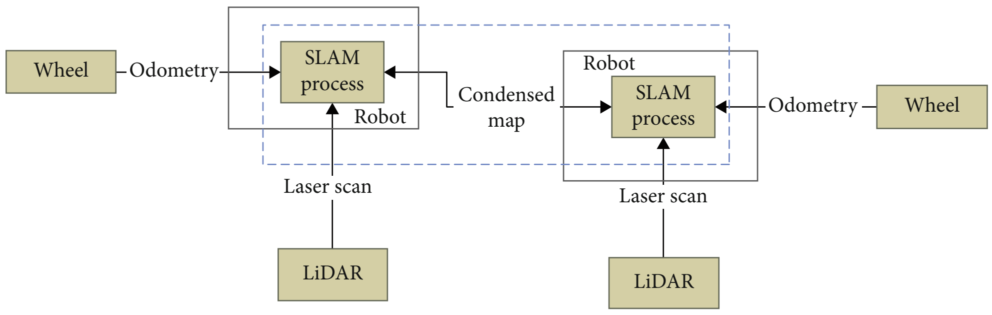


FIGURE 6: The architecture of CG_MRSLAM.

TABLE 1: The input public data set used by each robot.

Robot index[a]	Duration	Trajectory length	Laser scan inputs	Odometry inputs
1	155.72 s	40.380 m	1457	1548
2	115.63 s	21.735 m	1074	1152
3	111.91 s	18.135 m	1046	1116
4	72.75 s	16.118 m	679	723

[a]The names of corresponding data sequences are *freiburg2_pioneer_slam*, *freiburg2_pioneer_slam2*, *freiburg2_pioneer_slam3*, and *freiburg2_pioneer_360*, respectively.

Message deduplication: because the ROS programming model commonly uses a topic-based communication pattern, the publisher must send identical message data to each subscriber in local native setups, for the reason that ROS network stack has no knowledge of the underlying topology. In Cloudroid Swarm, the approach of message deduplication is applied to handle inefficiencies of this nature. When node N_i sends messages on a topic to which the other computation modules subscribe, only one copy of the data needs to be transferred via channel CH_i. All the topics published by N_i are delegated by NM_i on the cloud side, and when NM_i receives the message, it takes the stored subscriber list shared by N_i to forward the message to the other computation modules. Situations in which a message is sent from the network module to other local robots are processed similarly.

Optional message pull: the ROS message model defines that as long as there exists a subscriber of a topic, the publisher must send every message whether the subscriber uses it or not. Under the split model of Cloudroid, this situation will press a large impact on the network bandwidth. To solve this problem, we allow the user to define the optional argument of each message to define whether publishers use “pull” or “push” mode to send messages when robots upload the application to Cloudroid Swarm. The push mode is the default behavior of the ROS model, and on the other hand, the pull mode enables the on-demand sending of the message only at the time when the subscriber request this message. We design the pull mode as optional because it may cause more complicated behavior in a real-time consideration of some messages. However, in the performance evaluation of our system, we have noticed a significant improvement in network bandwidth and QoS under the on-demand pull mode.

Time sequence message elimination: another essential optimization point targets the time-efficient topics. Receiving the latest message is vital for a subscriber to maintain real-time performance. Instead of sending each message using the default FIFO (first in first out) behavior of ROS, for this kind of topic, we optimize it by always publishing the latest message over the network. Other previous historical messages have been of little use and can be safely eliminated for increased network efficiency.

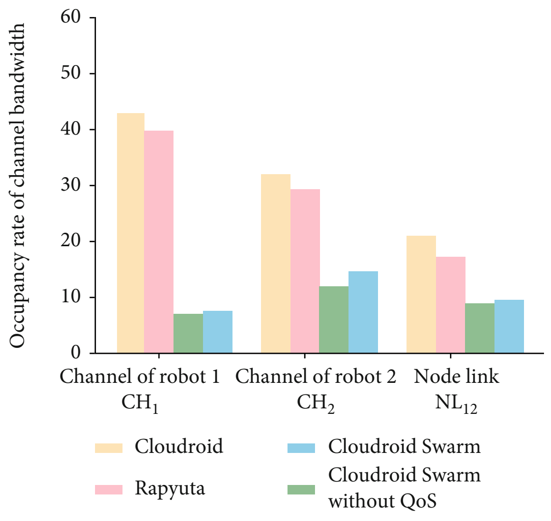
Custom compress transport: message compression is enabled to reduce the network footprint, and we choose Google Protobuf (Google Protobuf serialization: https://developers.google.com/protocol-buffers/) to (de)serialize messages. For communication via cloud links (if more than one server is configured), we exploit the ZeroMQ (ZeroMQ: http://zeromq.org/) distributed messaging system, which is more friendly to the cloud environment and provides significant efficiency.

We reuse the infrastructure of ROS and Cloudroid to implement certain components of Cloudroid Swarm. Our framework is designed at the platform level and is transparent to the overlying robot application, which allows the original application to be safely migrated to the cloud without the need for any code modification. In addition, the code for the topology engine is approximately 1,100 lines of code, whereas another 1,400 lines of code are devoted to the network module and network operator.

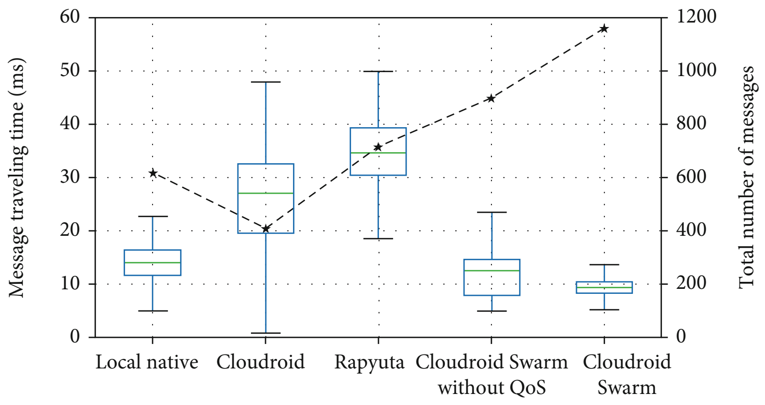
6. Evaluation

This section presents our evaluation of the performance of Cloudroid Swarm with three different types of multirobot applications, whose inputs are from the public data set, simulation environment, and real-world turtlebot system, respectively. These representative tasks are multirobot SLAM, collision avoidance, and exploration, all of which call for cooperation between robots and involve a large amount of data exchange among robots. The evaluation of each application also includes an experiment we conduct on other offloading or local native configurations for comparison. To the best of our knowledge, work that focuses on cooperation offloading has not yet been reported. Thus, during the evaluation, we compare our work with the following baselines.

(i) *Local native*: without any assistance from the cloud, all the computation and cooperation occur locally



(a) Bandwidth usage of data links



(b) Metrics of the condense map message. The message count is represented by the black line, whereas the latency (including the median and variance in each scene) is shown in the form of bars (lower or narrower is more favorable)

FIGURE 7: Communication performance in each of the scenes.

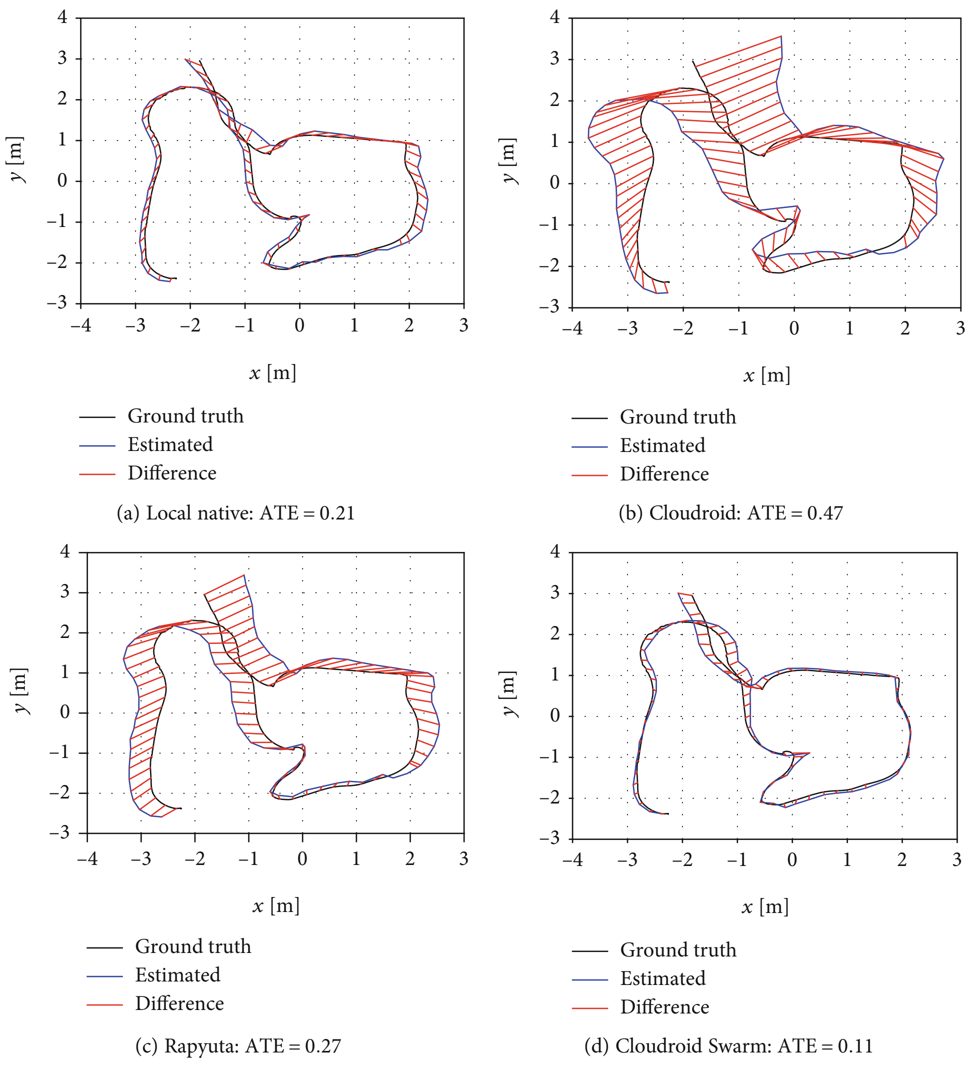


FIGURE 8: Built trajectories and ATE values of robot 3 for the four setups.

on the robots. This is also the target environment for the design of the three applications

(ii) *Cloudroid*: Cloudroid is a general framework for computation offloading. Certain computationally intensive tasks are configured to be migrated to the cloud side for enhancement

(iii) *Rapyuta*: although the architecture of Rapyuta supports computation offloading, similar to Cloudroid, Rapyuta is consolidated with more cloud-based techniques such as a load balancer to provide more flexible control for developers

(iv) *Cloudroid Swarm without QoS*: Cloudroid Swarm is our framework designed for multirobot cooperation offloading with network optimization for QoS. We set this baseline without a QoS mechanism, such as link detection and global link capacity adjustment for ablation studies

We deploy an outstanding commercial public cloud with four computation hosts as the testbed for all cloud-based setups on the cloud side. Each host is configured with a four Intel Xeon E5-2682 CPU, 16 GB RAM, and hosts are interconnected with 1 Gbps Ethernet. On the robot side, the physical platform for the four robots is the wheel-driven robot TurtleBot3 (Turtlebot: http://www.turtlebot.com/), which is equipped with LiDAR for laser scanning. The onboard robot processing computer used is Raspberry Pi 3 Model B (Raspberry Pi 3 Model B: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/), with a CPU containing four cores, 1 GB RAM, and BCM43438 wireless LAN (802.11b/g/n standard with up to 72.2 Mbps net throughput).

6.1. Evaluation Case 1: Cooperative SLAM. This section describes our evaluation of the efficiency of Cloudroid Swarm on CG_MRSLAM [26], a ROS-based framework designed to enable multiple robots to participate in a cooperative SLAM process. Each robot continuously and incrementally sends the map built by itself to another robot nearby using peer-to-peer communication during the task. When other robots receive these local maps, they integrate them into their own maintained map to build the global one. From the perspective of message flow, the architecture of CG_MRSLAM is depicted in Figure 6, where the area enclosed within the blue line can be migrated to the cloud platform for enhancement in our setup of cloud offloading. Operators that are intensive in terms of computation and communication, such as localization and mapping, can be migrated to the cloud platform for increased execution efficiency in our cloud offloading setup. The robots only process the laser scan and odometry inputs related to the hardware. Unlike the original local setup, where a message can only be sent when two robots are sufficiently close in proximity, our cooperation offloading method eliminates the distance limitation.

To compare the influence of communication on the final accuracy of the task, we choose to conduct the experiments using public sensor and actuator data [27], which was captured by the Technical University of Munich using the Pioneer robot (Pioneer P3-DX: https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html). The four data sequences in Table 1 were collected in the same indoor scene, and we apply each one separately to our robots as the simulation input data.

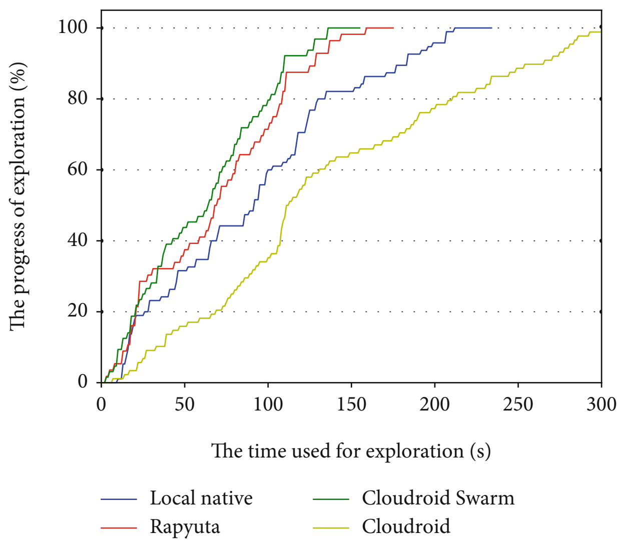


Figure 9: Exploration progress of AAU.

Table 2: The overlapping of exploration.

Setup	Overlapping ratio
Local native	21%
Cloudroid	37%
Rapyuta	35%
Cloudroid Swarm	5%

(1) Communication performance

To evaluate the network optimization efficiency of the QoS-aware link capacity adjustment algorithm proposed in Section 4.2, we introduce an additional cooperation offloading setup without QoS algorithms and mechanisms for baseline in this section; the link capacity of the channel is equally shared among users.

To demonstrate our framework's ability to relieve the pressure of communication, we also investigate the bandwidth usage during the task in Figure 7(a). The results show that for three representative links (CH_1, CH_2, and NL_{12}), the usage of links in Cloudroid Swarm is the lowest, especially for the channels between robots and the cloud, which are more easily to be bottlenecks. In particular, CH_2 exhibited a 57% decrease in bandwidth usage compared with Cloudroid, whereas for CH_1, the decrease exceeded 80%. It is also observed that for all links, adding QoS mechanisms increases the bandwidth occupancy slightly. This is because, except for data exchange, our sliding-window algorithm for link detection also takes up bandwidth.

During the evaluation, we record the latency for each map message data and the total number of messages transferred

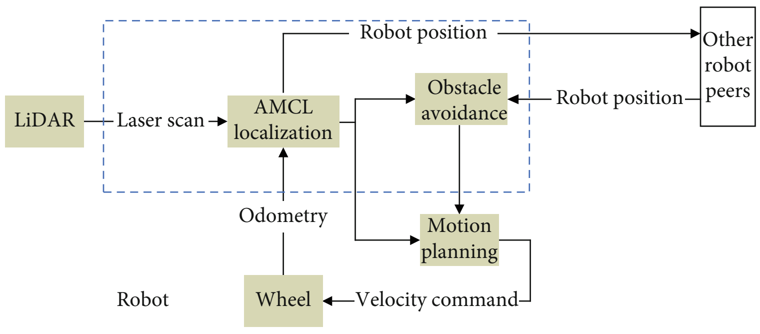


FIGURE 10: The architecture of Collvoid.

FIGURE 11: The robot group used for real-world applications.

during the task, as depicted in Figure 7(b) From the number of messages depicted by the black line, we learn that the number of messages increases significantly with Cloudroid Swarm, compared with the other three setups. More messages exchanged indicates more cooperation among robots. With direct message transmission between network modules, the latency with our framework is also largely reduced and becomes more stable; the variance is the smallest of all scenes. It is observed that when we introduce the QoS-aware link capacity adjustment algorithm, Cloudroid Swarm obtains the lowest average and maximum message traveling time and the highest number of messages, indicating the best network optimization performance. This is because our framework will choose the local computing strategy when the message traveling time of cooperation offloading is longer than the message traveling time of local computing. The QoS mechanisms guarantee the task performance under poor or dynamic network environments. Note that without enhancement using cooperation offloading, our original framework, Cloudroid, has the most unstable network performance with a large variance in message latency.

(2) Task accuracy performance

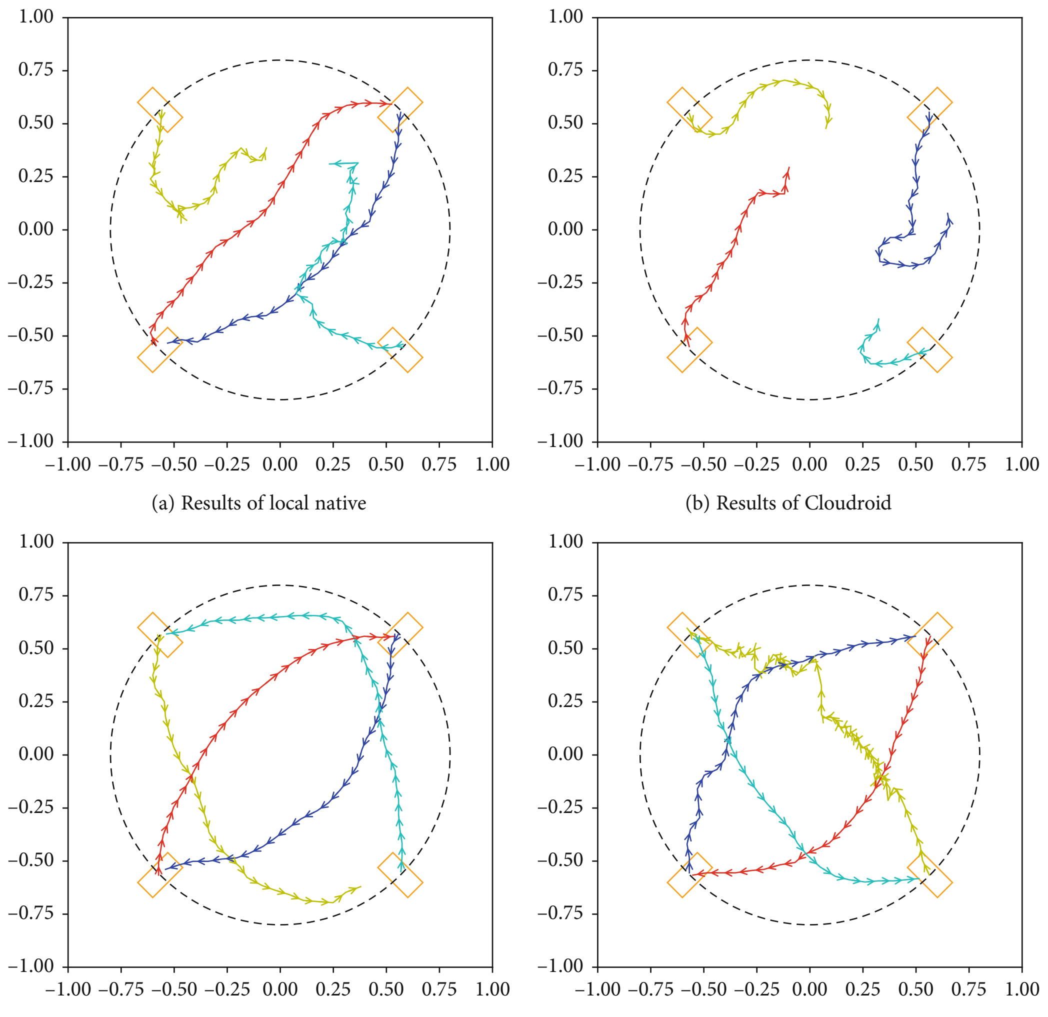
We also conduct the task accuracy evaluation using the ground-truth trajectory data provided, inspecting and discussing the trajectory we generated from the CG_MRSLAM. Our results with the data set named *freiburg2_pioneer_slam3* for all four setups are depicted in Figure 8. The ground-truth trajectory is also shown for comparison, and the red line represents the transitional error. For tracking precision, we use *ATE*, a metric defined in [27], to describe the difference between the ground-truth and the estimated trajectory. ATE is calculated using the least-squares method to find a rigid-body transformation T, which maps the estimated trajectory E_n onto the ground truth G_n. Then, the root-mean-square error over all time indices of the transformation components is evaluated using the following expression:

$$\mathrm{ATE}(E_n) = \left(\frac{1}{n}\sum_{i=1}^{n} \left\|\mathrm{trans}\left(E_i^{-1} T G_i\right)\right\|\right)^{1/2}. \tag{6}$$

Comparing the red accumulated error and ATE value, we learn that in Figure 8, especially in subfigures (b) and (c), because parts of the condensed map cannot be transferred smoothly between robots with limited bandwidth, the localization phase of CG_MRSLAM easily drifts from the ground truth. This phenomenon causes a considerable increase of the ATE. We find the ATE of Cloudroid and Rapyuta to be 2.24 and 1.28 times higher, respectively, than the local native setup. This indicates that with computation offloading, the message overload is so high that it makes the performance even worse. These results also correspond with the analysis shown in Section 3.2. With our Cloudroid Swarm, the ATE decreases to 0.11, which nearly doubles the performance of the local native. Thus, cooperation offloading instead of only computation offloading is adaptive to this task.

6.2. Evaluation Case 2: Multi-robot Exploration. Multirobot exploration, a task that collaboratively explores the frontier of an unknown environment by a robot group, is also evaluated by our environment. During this task, the information about the frontier and border is transmitted to other robots to negotiate the explored areas.

The application suite we leverage is the collaborative exploration framework proposed by Alpen-Adria-Universität Klagenfurt [28], abbreviated as AAU. Different robots individually conduct frontier exploration, which sends the local map and robot location to all the other robots for merging into a global map. The architecture of AAU has many similarities with the architecture of CG_MRSLAM, and both use laser scans and odometry to sense the environment. Although both CG_MRSLAM and AAU broadcast the local map to peers, AAU chooses the entire local map whereas CG_MRSLAM only sends the condensed map. However, because exploration is an application that requires the robot to be able to move freely, the timeliness of the interrobot message is a dominating



(a) Results of local native

(b) Results of Cloudroid

(c) Results of Rapyuta

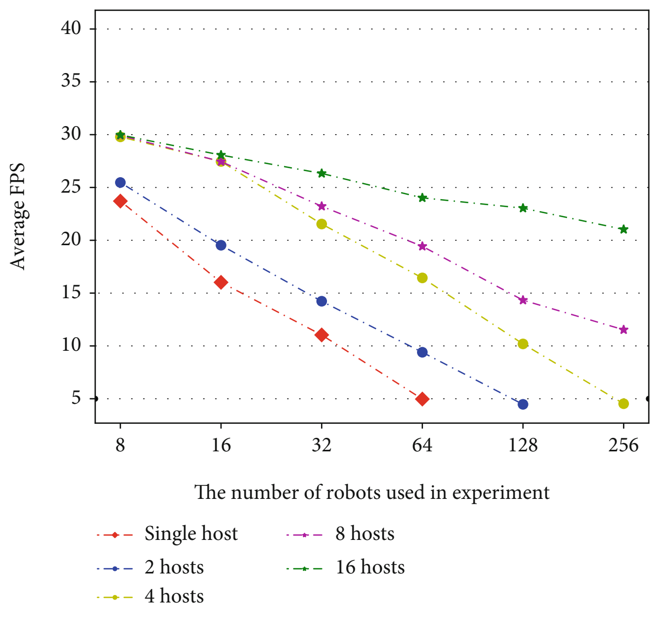
(d) Results of Cloudroid Swarm

Figure 12: Result of the evaluation of Collvoid: (a–d) show the motion trajectories of the four robots with the Local Native, Cloudroid, Rapyuta, and Cloudroid Swarm setups, respectively.

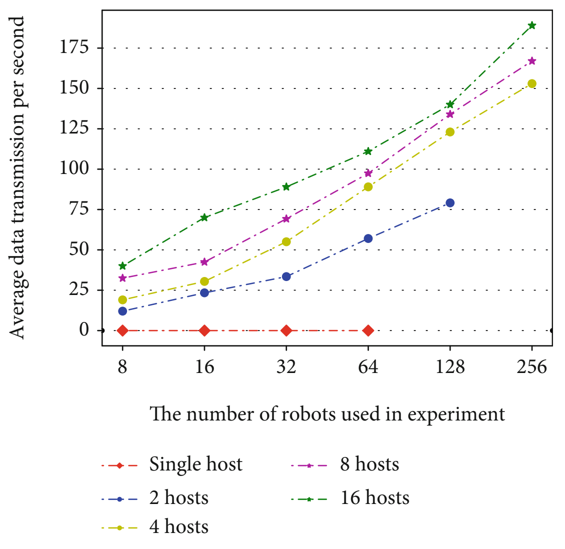
factor in the accuracy of map merging, which directly becomes an important factor in the speed and accuracy of exploration. In order to evaluate the performances of the multirobot system quantitatively, we use a ROS stage simulator similar to [28]. In addition, AAU uses another communication mechanism for its local architecture; specifically, robots that are not sufficiently close to each other but want to exchange data can use the third robot as a relaying router for message forwarding. Although this would be an effective optimization approach for local communication, in the setup of Cloudroid Swarm, it becomes unnecessary and can even increase the latency. To accommodate this situation, we configured the internal ad hoc communication of each node to be migrated to the cloud side such that it is transferred directly inside the cloud to boost performance.

The performance of multirobot exploration tasks can be measured as the time used to expand the entire area of an explored place. The communication efficiency has a strong influence on the overlapping area, which in turn affects the coverage speed of the entire robot group. Based on this fact, we measure the total exploration progress of the four robots during the task, as shown in Figure 9. To eliminate the effect of differences in the size of the total area, we normalize the size as a ratio of the total size. In the initial phase of the task, the size of the overlapping area between robots is very small, and the progress in this phase increases rapidly. However, when the overlapping begins to increase, the message transmission path optimized by Cloudroid Swarm begins to increase and becomes advantageous relative to the other three setups. Note that although the additional communication path suppresses data exchange, it still outperforms the local native setup because the computation is offloaded in Rapyuta; i.e., inequality (3) can be satisfied in this configuration. Additionally, the capability of our method on AAU is also demonstrated in the exploration overlapping in Table 2, where Cloudroid Swarm has the smallest value. This is because the transmission of messages between robots is more efficient so that 5% overlapping is enough to match and joint the local maps of every robot to construct a global map.

6.3. Evaluation Case 3: Multi-robot Collision Avoidance. In this section, we present our investigation of the performance



(a) Average FPS



(b) Data transmission rate (Mbps) in the cloud side

Figure 13: The scalability metrics in different configurations.

of Cloudroid Swarm on a multirobot application with more complex architecture with real-world robots. This task requires each robot in the group to navigate to the specified target position while avoiding the obstacles in the maps and their robot peers. The application we use in the experiment is *Collvoid* [29], which is a multirobot collision avoidance system based on the velocity obstacle paradigm [30]. The architecture of Collvoid is based on the original local wireless network, and the pipeline of its algorithm is as follows:

(i) Each robot receives information about the odometry of the wheels and uses a laser scanner for a more precise localization procedure

(ii) The position information is then broadcast over the wireless network, and the peers receive these messages and integrate them to detect both the obstacles and their peers

(iii) Based on self-estimated localization and obstacle detection, the robot performs motion planning for further navigation and guides the wheels for movement with velocity commands

The modules and message flows are shown in Figure 10. From the perspective of computation, the most computationally intensive operator is AMCL localization. When this operator is offloaded to the cloud for acceleration using computation offloading in Cloudroid and Rapyuta, the localization messages are shared locally using the wireless network, whereas in Cloudroid Swarm, it occurs in the cloud links. The real-world experiment is conducted in a closed indoor environment. Depicted in Figure 11, the four robots form a square, and each one navigated to the position occupied by its peer diagonally across while running Collvoid to avoid colliding with its peers. The inputs are directly from the turtlebots' laser scanning, containing more noise. So more efficient communication is required to complete the cooperative task.

In the evaluation, two crucial metrics need to be evaluated to compare the performance of Collvoid. The first is to determine whether robots can complete the traveling tasks and reach the specified target location, and the other is the smoothness of the traveled trajectory. A smoother trajectory indicates more flexible control of the robot. As we observe in Figure 12, all four robots can reach their specified target in Figure 12(d). On the other hand, in the other three setups, because the message transmission is too inefficient to carry precise information of obstacles and locations, one or more of the robots failed to reach their target. However, the various obstacle messages lower the yellow robot's smoothness relative to others, especially using our framework. This is because with cooperation offloading, robots receive more messages, thus behaving more conservatively to avoid collisions. Since we do not focus on the algorithm itself, this result is in line with our expectations.

6.4. Scalability Evaluation. In this section, we evaluate our framework with a large group of robots to show the scalability of Cloudroid Swarm. However, the scalability validation of most existing multirobot applications, such as multirobot SLAM, is still limited to a small group of robots, which is hardly directly scaled to dozens of robots. We improve ORB-SLAM [31] for better scalability and propose a scalable and real-time multirobot visual SLAM framework [32]. This framework can effectively divide and schedule SLAM task inside a cluster, with the group-based parallelism and the map point multicut algorithm. The framework adopts a switchable messaging pattern to meet different transmission scenarios to reduce the data sharing latency between different hosts. And the map data consistency is improved by the designed linage feedback and timestamp versioning mechanisms.

Because we do not have the condition for the experiment to exploit hundreds of real-world robots for evaluation, during the simulation evaluation, we choose Docker container to emulate the physical robots instead. Each container is configured as one Xeon E5-2682 CPU and 2 GB RAM, and the network bandwidth is limited to 50 Mbps, which is the ceiling rate of the wireless and 4G cellular network devices on robots. The used public data set is the same as in Section 6.1. In order to adapt the data set to large-scale robot swarms, we divide each image data sequence into multiple pieces, making every segment have a length of 20 s.

Since our method is fully distributed, the scalability of it is demonstrated from two aspects, the number of robots which our framework can support for performing cooperative SLAM in real-time and the number of computation hosts in the cloud side our method can extend to. In this situation, we choose the number of computation hosts varied from 1, 2, 4, 8, to 16, and the different numbers of simulated front-end robots ranged from 16, 32, 64, 128, to 256, to deep insight into the effects on each combination.

The results are depicted in Figure 13, where the average metrics, including FPS, data transmission rates, and group sizes, are shown to characterize the performance of our method. Although our method also encounters FPS decreasing when the number of robots is increasing, FPS also has to get promotions when the number of hosts increases. Especially in the case of 16 hosts, even for 256 robots, it has retained the rate to more than 20 FPS, which is enough for the real-time requirements of SLAM applications. The data transmission in the most remarkable case is 188 Mbps (256 robots in 16 hosts), which is exceeding the local communication capabilities of the mobile robot, but still much less than 1 Gbps network bandwidth in the cloud side. With more hosts (from 4 to 8, 16) deployed in the cloud side, the data transmission does not show a significant increase, indicating that our framework can be effectively scaled out in the cluster.

7. Discussion

Our approach has some limitations. Considering the limited computing capabilities and real-time requirements of mobile robots, we simplify the cooperation offloading problem by assuming that either all tasks are performed locally or all of them are offloaded to servers. However, it is common that each robot can determine whether to offload or not in the computation offloading scenes. We believe that deep learning and reinforcement learning will play important roles in generating cooperation offloading decisions for each individual.

Security is one of the critical issues in mobile cloud computing [33] and mobile edge computing [21]. Since our framework is based on ROS, every user connecting to the ROS master could leak sensitive information (such as data from sensors or cameras) or even send commands to move robots, creating privacy and safety risk. The problem becomes serious if we extend ROS to the public Internet. To alleviate this problem, we restrict only authorized users to access the platform. Some more advanced encryption algorithms should be introduced to safeguard the robots during the cooperation offloading process to deal with security threats (e.g., snooping and alteration).

MEC servers are much closer to mobile devices and thus have lower latency, while MCC servers can provide flexible and scalable computing capability to support complicated applications [34]. For simplicity, we do not explicitly differentiate between edge servers and remote cloud servers in our formulations. However, the distinguishing characteristics of edge computing include its dense geographical distribution, support for mobility, and proximity to end users [35]. Loghin et al. [36] demonstrate that MEC is more effective than MCC when the task has a higher input-to-output ratio and lower computation-to-communication ratio for uploading and processing the input on the cloud. Though we conduct our experiments with MCC until now, we believe it is easy to extend cooperation offloading to MEC according to the situation.

8. Conclusion

In this paper, we study the cloud-based offloading problem in multirobot cooperative scenes and propose an approach named cooperation offloading for robot swarms performing a cooperative task. We analyze the time cost and then propose offloading decisions by formalizing a general model for this problem. To apply this concept in a practical situation, we propose a set of network components and develop an algorithm on both the local and global levels to optimize the network links. Next, we implement Cloudroid Swarm and use three representative multirobot applications to validate the framework in a constrained network environment. The results show the efficiency of our approach, which enhances the communication performance more than twice and the task performance more than four times compared to the setup without offloading or with well-known computation offloading frameworks. Finally, we verify the feasibility of our framework in the real-world environment and scalability with hundred-level robot swarms.

Data Availability

The input public data set used by four robots in case 1, Cooperative SLAM, is available at https://vision.in.tum.de/data/datasets/rgbd-dataset/download.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is partially supported by the major Science and Technology Innovation 2030 "New Generation Artificial Intelligence" project 2020AAA0104803.

References

[1] A. U. R. Khan, M. Othman, A. N. Khan, J. Shuja, and S. Mustafa, "Computation offloading cost estimation in mobile cloud application models," *Wireless Personal Communications*, vol. 97, no. 3, pp. 4897–4920, 2017.

[2] B. Ding, J. Xu, H. Wang, H. Zhang, H. Liu, and D. Feng, “Invited paper: distributed computing in cyber-physical intelligence: robotic perception as an example,” in *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 1–17, San Francisco, CA, USA, 2019.

[3] G. Mohanarajah, V. Usenko, M. Singh, R. D’Andrea, and M. Waibel, “Cloud-based collaborative 3d mapping in real-time with low-cost robots,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 423–431, 2015.

[4] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, “A fog robotics approach to deep robot learning: application to object recognition and grasp planning in surface decluttering,” in *International Conference on Robotics and Automation (ICRA)*, pp. 4559–4566, Montreal, QC, Canada, 2019.

[5] Y. Li, H. Wang, B. Ding, and W. Zhou, “Robocloud: augmenting robotic visions for open environment modeling using internet knowledge,” *Science China Information Sciences*, vol. 61, no. 5, article 050102, 2018.

[6] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, “QoS-aware cooperative computation offloading for robot swarms in cloud robotics,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 4027–4041, 2019.

[7] Y. Zhai, B. Ding, P. Zhang et al., “Cooperative offloading for multiple robot applications,” in *IEEE International Conference on Joint Cloud Computing*, pp. 63–70, Oxford, UK, 2020.

[8] 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, 1611 pages, 2013.

[9] I. Farris, T. Taleb, H. Flinck, and A. Iera, “Providing ultra-short latency to user-centric 5g applications at the mobile network edge,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, article e3169, 2018.

[10] Y. Chen, Z. Du, and M. García-Acosta, “Robot as a service in cloud computing,” in *Fifth IEEE International Symposium on Service Oriented System Engineering*, pp. 151–158, Nanjing, China, 2010.

[11] R. Arumugam, V. R. Enti, L. Bingbing et al., “DAvinCi: a cloud computing framework for service robots,” in *IEEE international conference on robotics and automation*, pp. 3084–3089, Anchorage, AK, USA, 2010.

[12] G. Mohanarajah, D. Hunziker, R. D’Andrea, and M. Waibel, “Rapyuta: a cloud robotics platform,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 481–493, 2015.

[13] B. Hu, H. Wang, P. Zhang, B. Ding, and H. Che, “Cloudroid: a cloud framework for transparent and QoS-aware robotic computation outsourcing,” in *IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 114–121, Honololu, HI, USA, 2017.

[14] H. Yu, Q. Wang, and S. Guo, “Energy-efficient task offloading and resource scheduling for mobile edge computing,” in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–4, Chongqing, China, 2018.

[15] S. Bi and Y. J. Zhang, “Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[16] X. Chen, “Decentralized computation offloading game for mobile cloud computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2014.

[17] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, “Risk-aware data offloading in multi-server multi-access edge computing environment,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1405–1418, 2020.

[18] C. Zhang, P. Patras, and H. Haddadi, “Deep learning in mobile and wireless networking: a survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[19] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, “Collaborate edge and cloud computing with distributed deep learning for smart city internet of things,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8099–8110, 2020.

[20] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin, “Learning for computation offloading in mobile edge computing,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.

[21] B. Huang, Y. Li, Z. Li et al., “Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing,” *Wireless Communications and Mobile Computing*, vol. 2019, Article ID 3816237, 20 pages, 2019.

[22] T. Arai, E. Pagello, and L. E. Parker, “Guest editorial advances in multirobot systems,” *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[23] E. Klavins, “*Communication complexity of multi-robot systems*,” in *Algorithmic Foundations of Robotics V*, 2004.

[24] R. Doriya, S. Mishra, and S. Gupta, “A brief survey and analysis of multi-robot communication and coordination,” in *International Conference on Computing, Communication & Automation*, pp. 1014–1021, Greater Noida, India, 2015.

[25] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, K. Nakamura, and J. Ogawa, “A study of robotic cooperation in cloud robotics: architecture and challenges,” *IEEE Access*, vol. 6, pp. 36662–36682, 2018.

[26] M. T. Lazaro, L. M. Paz, P. Pinies, J. A. Castellanos, and G. Grisetti, “Multi-robot slam using condensed measurements,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1069–1076, Tokyo, Japan, 2013.

[27] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, 2012.

[28] T. Andre, D. Neuhold, and C. Bettstetter, “Coordinated multi-robot exploration: out of the box packages for ROS,” in *Globecom Workshops (GC Wkshps)*, Austin, TX, USA, 2014.

[29] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen, “Collision avoidance under bounded localization uncertainty,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1192–1198, Vilamoura-Algarve, Portugal, 2012.

[30] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935, Pasadena, CA, USA, 2008.

[31] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: a versatile and accurate monocular slam system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[32] P. Zhang, H. Wang, B. Ding, and S. Shang, “Cloud-based framework for scalable and real-time multi-robot slam,” in *2018 IEEE International Conference on Web Services (ICWS)*, pp. 147–154, San Francisco, CA, USA, 2018.

[33] A. N. Khan, M. M. Kiah, S. U. Khan, and S. A. Madani, “Towards secure mobile cloud computing: a survey,” *Future*

Generation Computer Systems, vol. 29, no. 5, pp. 1278–1299, 2013.

[34] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: an energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2163–2176, 2021.

[35] E. Ahmed, A. Ahmed, I. Yaqoob et al., "Bringing computation closer toward the user network: is edge computing the solution?," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 138–144, 2017.

[36] D. Loghin, L. Ramapantulu, and Y. M. Teo, "Towards analyzing the performance of hybrid edge-cloud processing," in *2019 IEEE International Conference on Edge Computing (EDGE)*, pp. 87–94, Milan, Italy, 2019.