A Server Select Method for distributed robotic system based on ROS

YIFAN LIU[†1] MIDORI SUGAYA[†1]

Abstract: Currently, multiple robots are expecting to provide useful services corporately at the event of disaster and medical treatment, nursing care, home security etc.. Autonomic robots such as searching and following people should have the functionalities of collecting data from its physical environment and analyzing to share the information with each other. And the robots should be connected to the internet and share the calculate resources through a large number of distributed servers for the purpose to offer better services. However, when it comes to the internet for the robots, the latency of the internet can not be ignored. In this paper, we described and implemented a system with server selection depends on network latency based on ROS with several distributed servers. An evaluation had been done by comparing with and without the server selection. Finally, the issues may occur while scaling the system and solution had been discussed in this paper.

Keywords: multiple robots, ROS, network latency, server select

1. Introduction

Multiple robots' services are required for rescue, nursing, security and etc. These robots working together is more efficient than the robots that work separately. These cooperated works with information exchange could avoid repeated works while searching people. Moreover, it saves time to search survivals and is able to save more people. In other field such as nursing robot, security robot, there are also such demands existed for multiple robots to collect data so that we can know abnormal situation such as when nursing robots detect falling down of person and invasion detected by security robot in time.

And recently, since the technologies such as artificial intelligence, big data is becoming more and more popular, bringing them to the robots' world could make a huge difference. Also, the application of simultaneous localization and mapping (SLAM [1]) is needed for the robots' navigation. However, all of these technologies require a large amount of calculation. Due to the resources of the robots such as central processor unit, memory and etc. is limited, these kinds of complex calculation may not be possible to be finished on the robot itself. Under this situation, we consider that connect the robot with a more powerful server or even a large number of servers becomes necessary.

To achieve the goal to provide services with multiple robots that can share information among each other and share the calculation resource through a number of distributed servers to do their work more efficiently, we consider that a software framework for building this system is required to satisfy the requirements. The one of the most widely used software framework for robotic systems is ROS (Robot Operating System) [2,3], and it provides a structured communications layer above the host operating systems. It also provides libraries and tools to help software developers create robot applications. The ROS node makes no assumptions about where in the network it runs, allowing computation to be relocated at run-time to match the available resources, which makes it suitable for distribute robotic system.

However, while developing this system and bringing it to the internet, some issue occurs because the latency of the internet can not be ignored. Due to the latency of the internet, some tasks for the robots such as image process could cause failures because of the robots could not get the result from servers as soon as required. Although it is expected that only the ROS will be satisfied for the requirements of variety of services, the result turns out negative because of the network latency. For the reason that there are several servers in the system, we consider a method to select the appropriate server to finish the task depend on network latency.

In this paper, in section 2, we propose a system, and describe the specific system implementation. Section 3 shows the evaluation process and result. In section 4, we discussed about the issues of this system and the possible solution. Eventually, we made a conclusion about this paper.

2. Proposal

2.1 Issues and proposal

As described in the introduction, there are the problems of network latency, and because of the latency, some tasks which requires the result immediately could cause failures of tasks. Since the robots are moving around and the server is placed in different location, the network latency between robots and servers is changing. Once the robot is moving far from the calculate server which means the latency is becoming high, failure of tasks would occur. To solve this problem, we designed and developed a system to dynamically selects server that has the minimal latency between robot and server.

As figure 1 shows, first, all the network latency between robots and servers is measured every second and send to the manage server. Then the latency will be sorted to find out the least network latency between robots and servers and this server will be selected. While the robot is moving, the calculation will be switched to another server that has the least network latency. In this figure, the robot 1 selected the server 1, the robot 2 selected the server 2 and etc.

†1 Shibaura Institute of Technology

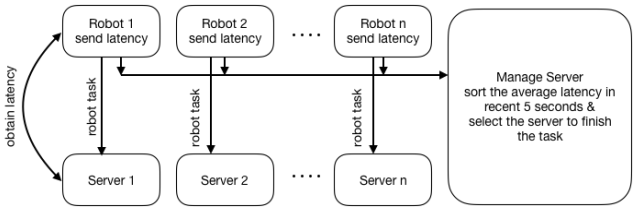

Figure 1 server select system

The server select method is based on the recent 5 seconds' latency that measured in the real time. The latency that measured indicates the network status between servers and robots. As the basic system, we compared the latency among every servers and find out the least latency, finally select this server for the specific robot's task. While comparing the latency, for example, the average latency in the recent 5 seconds between one robot and several servers will be sorted by a sort algorithm named Timesort. Timsort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data [4]. The reason why we choose this algorithm is that this algorithm will also be efficient even if the sorted object becomes large, so is will not be an issues while scaling this system.

2.2 Monitor and analyze the latency

In order to obtain the latency between each robot and server, there are two communication ways that is possible to choose in robot operating system, one is called the publish/subscribe model and another is called request/reply interaction.

The publish/subscribe model provides a flexible communication paradigm, which the sender of messages called publishier do not program the messages to be sent directly to specific receivers called subscriber, but instead published message into classes without knowledge of which publishers. On the contrary, its many-to-many one-way transport is not appropriate for RPC request/reply interactions, which are often required in a distributed system. Request/reply is realized via a ROS service, which is defined by a pair of messages: one for the request and one for the reply. And in ROS, the pub/sub model is an asynchronous communication way and the ROS service is a synchronous communication way. All of them are based on TCP/IP communication.

Under this situation, if the network latency is required, it has to be round trip latency since it is so difficult to synchronize two machine's clock. As the pub/sub model is a one-way transport, we have to choose to use ROS service as the communication way to obtain the latency.

As Figure 2 shows, we choose to put the ROS service server (/latency_RxS1, latency_RxS2 …) on every server except the manage server to reply the request from robots. And the latency is the average latency in recent 5 seconds.

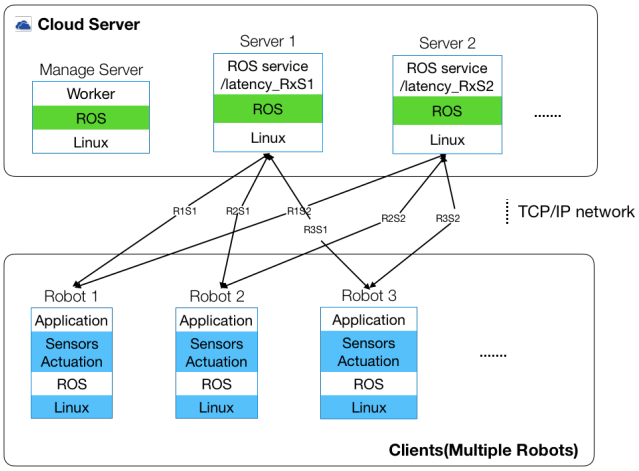

Figure 2 the way to get latency via service

2.3 Manage server

After getting latency, every robot will publish its latency between server to topics. For example, as Figure 3 shows, the robot 1 will publish its latency between every server every second to topics (/latency_R1S1 which means the latency between Robot 1 and Server 1 & /latency_R1S2).

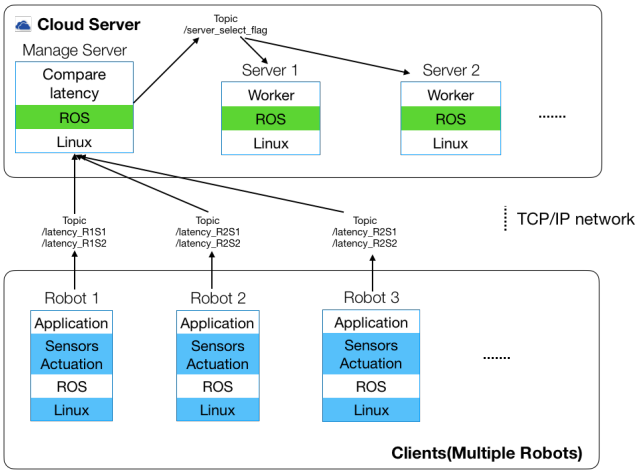

Figure 3 publish latency to topics & manage server subscribe

The manage server will subscribe every topic that the robots publish and calculate the nearest five seconds' average latency and publish the shortest latency's server number to a topic named /server_select_flag. And all the calculation server (Server 1, 2, …) will subscribe to this topic as the figure shows. While a task is given to robots, according to the /server_select_flag, the task will be done by the server chosen for the purpose to get a better response performance.

2.4 Example

For example, like Figure 4 shows, while an image process task is given to robot 1, it will publish a topic named /task_image. According to the latency, the selected server will do the image process. As the robot is moving around, maybe the next second, the robot is nearer to another server (the latency between another server becomes less), the calculation task will be switch to

another server. And the processed image will be send back to robot through another topic named /processed_image as the task is finished.

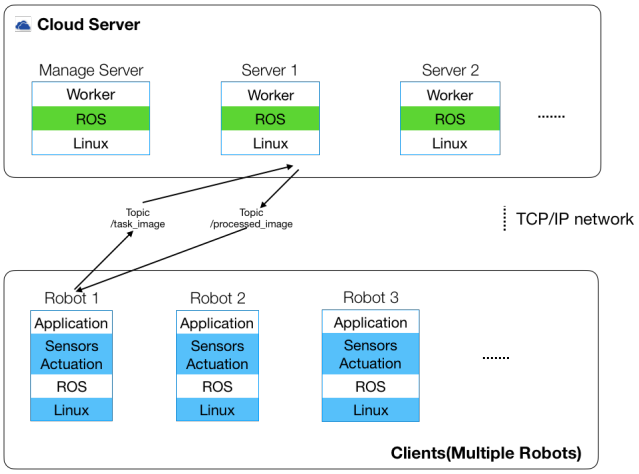

Figure 4 example of task processing

3. Evaluation

3.1 Evaluation environment

We utilized 6 Raspberry Pi 3 Model B V1.2 that the hardware information is included in Table 5 and a 2.4 GHz wlan router.

SoC	Broadcom BCM2837 900 MHz
Processor	ARM Cortex-A53, 4 cores processor
GPU	Broadcom VideoCore IV
Memory	1 GB LPDDR2

Table 5 Raspberry Pi 3 model B

For the evaluation, we utilized 1 as the manage server, 2 as the robots and another 3 as the calculation server. And all the machines are connected to a wireless router which support IEEE802.11a/b/g/n/ac wireless network protocol with a max speed at 866Mbps.

For the software part, we had installed Raspbian Jessie (Operating System) and ROS indigo 14.04 on all the hardware.

3.2 Evaluation process

In the evaluation part, we simulated that the task for the robot is to sort a 1000 length list. We utilized a Int32MultiArray as the ROS message type among servers and robots.

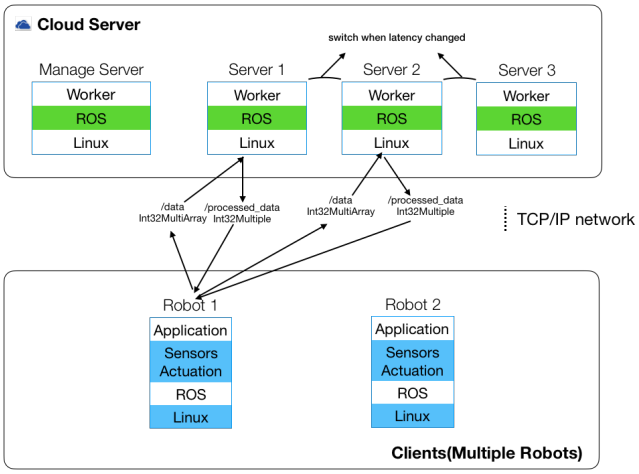

Figure 6 evaluation

We executed two experiments, and I will describe about this two experiment in the following. In the first one, we only use the server 1 for calculation without server switch. First, we generate a 1000 length random list on the robot side and then publish it with a time stamp to a topic named /data with a Int32MultiArray ROS message type. The server will subscribe to this topic, sort it and publish it with the original time stamp to a topic named /processd_data. On the robot side, it subscribes this topic and pull out the time stamp. The time we measured is from sending the message to receive the result as Figure 7 shows. The second experiment is the same with the first, but with a server select method depends on network latency. The calculation process was done 10~70 times for each experiment.

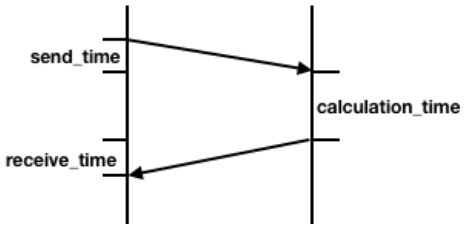

Figure 7 Measure Time

3.3 Evaluation result

As illustrated in Figure 8, the vertical axis shows the total time assumption with a unit second and the horizontal axis shows times of calculation. The total time have a significant decrease while using our server select method, the result turns about 10% less latency than without the system. The evaluation result shows that the server select method depends on latency that we developed works and such a method to solve the network latency is feasible.

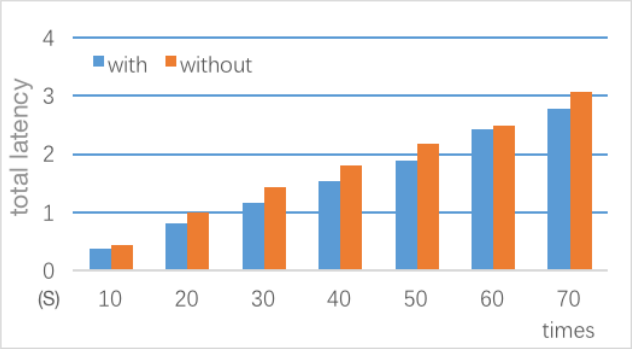

Figure 8 evaluation result

4. Discussion

The current system proves that it is feasible to solve the issue that we face in our system. However, there would be some issues occur while scaling the system.

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. In the current stage, there are only 3 servers in a narrow range included in the system. However, the real system should be scalable, thousands of servers in a wide range should be able to be added to the distributed robotic system.

Currently, in order to get the latency among servers and robots, there is a manage server exists. However, while scaling the system, the workload of manage server could be heavy which will lead to a failure of server select and finally lead to the failure of robot's task.

For the scaling issues, we consider a solution to group the servers first and for each group, a function like the manage server should be required.

Also, there is another issue, if the latency between robot and manage server is high, this system could not work as we expected because it can not select server seasonable. In order to solve this issue, it is necessary to consider that every server could hold a function like the manage server, which can sort the latency among robots and servers and finally make a decision to finish the task on which server. Also, there is another solution, if the calculation resource of robot is sufficient for the manage server's work, it is better to let every robot have the ability to make decisions to finish its task on which server because the latency between robots and manage server is the least under this situation.

5. Conclusion

In this paper, a distributed robotic system with server select method based on ROS is proposed and realized. At first, we described the issues that we face while bring the distributed robotic system into a wide range network. Then, we described a way to solve this issues by a server select method depends on network latency. And we introduced a specific way to implement this system and an evaluation for this system had been done. Finally, we discussed about the issues that will occur in this system while scaling the system and the possible solution.

Reference

[1] H. Durrant-Whyte, and T. Bailey, "Simultaneous localization and mapping: part I", IEEE Robotics & Automation Magazine, Volume: 13, Issue: 2, June 2006.

[2] "About ROS". http://www.ros.org/about-ros/, (accessed 2017-11-16).

[3] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng, "ROS: an open-source Robot Operating System", ICRA Workshop on Open Source Software, 2009.

[4] "Timesort". https://en.wikipedia.org/wiki/Timsort,(accessed 2018-8-25).