A Large-scale Distribution and Deployment of Robot Task Based on MQTT Protocol and ROS

Jing WEI*, Dianxi SHI, Bingzheng YAN
National Laboratory for Parallel and Distributed Processing, School of Computer,
National University of Defense Technology
Changsha, Hunan, China 410073
E-mail: {dubiousjing, bingzhengyan93}@gmail.com;
E-mail: dxshi@nudt.edu.cn
+* Corresponding author

Yerong HU
Laboratory of Science and Technology on Integrated Logistics Support, School of Mechatronics Engineering and Automation
National University of Defense Technology
Changsha, Hunan, China 410073
E-mail: yeronghu@foxmail.com

***Abstract*—The beginning and development of modern intelligent robot makes robot task become more and more complicated. Task distribution and deployment for large-scale robots has become a major problem. Traditional distribution mechanisms are mainly Polling, SMS Push and IP Push, but for the task distribution of robotic devices, these mechanisms cannot fully meet the system's requirements. In this paper, we propose a "Push-Pull" distribution mechanism, and make a comparative experiment with the "Direct-Push" distribution mechanism based on MQTT protocol. Meanwhile, we construct a large-scale distribution and deployment system of robot task, which implements the distribution and deployment of large text tasks promptly and can guarantee the quality of task distribution. The experimental results show that the task package distributed by our system can run successfully on the robot operating system (ROS).**

Keywords-MQTT; task distribution and deployment; large-scale robot; ROS

I. INTRODUCTION

The development of robots has been focused by academia and industry. The world's first industrial robot was born in the late 1950s [1], and henceforth the robot history really began. In recent years, artificial intelligence, deep learning and other technical developments continue to heat up, so that robots have ushered in a new revolution. Intelligent Robot [2] is a machine system that fully simulates people in terms of perception, thinking and effect. Compared with industrial robots, intelligent robots have functions of perception, recognition, judgment and planning. With the growing application of intelligent robots, people expect them to complete more and more complex task. Modern intelligent robots can not only complete their own tasks, but also collaborative work. Due to the limited computing capability of a single robot, a large unmanned system often requires a certain number of robots to cooperate with each other to complete the task [3]. If we need to copy each task program to respective robot, it will consume a lot of resources and time. If the task can be actively distributed to the robot, the large-scale robot's co-operation efficiency will be increased.

Method for obtaining information of mobile terminal mainly contains "client pull" and "server push". "Client pull" can obtain information through the polling strategy, that is, to send a request to the server to check whether there is an update to achieve the effect of pushing, but it has poor real-time performance. "Server Push" means that the server pushes the information to the client actively [4]. This method has the advantages of better real-time performance and higher efficiency. At present, the mainstream message push protocol includes XMPP and MQTT [5]. XMPP protocol is mature and powerful, but the protocol is more complex and redundant, it also cost more data flow, electricity and has high hardware deploying cost. MQTT protocol is an open, lightweight protocol that is particularly suited for embedded devices and mobile devices with low bandwidth, unstable and costly network, and limited processor and memory resources [6].

Thus, this paper constructs a robot task distribution system in large scale. The system is based on MQTT protocol to achieve the task distribution, and built on the currently mainstream robot operating system (ROS). According to the network environment of the robot and the requirement of real-time performance and low energy consumption, this paper puts forward the distribution mechanism of "Direct Push" and "Push-Pull", which can improve the timeliness and stability of the system effectively. Finally, the distribution efficiency of the two task distribution mechanisms is compared in the same network environment.

The remainder of the paper is organized as follows: Section 2 shows the research work related to this paper, and Section 3 elaborates the overall architecture of the system. In section 4, we introduce the principle of "Direct Push" and "Push-Pull" distribution mechanism. Section 5 is the implementation of task distribution system. The sixth part elaborates the system test method and the result. Finally, we make a conclusion and define the future development direction of the system in section 7.

II. RELATED WORK

A. Polling Mechanism

Polling mechanism adopts the traditional pulling technology, where the client actively sends request to the server for a new task or update in a certain frequency cycle. This method allows the client to determine the pull frequency, and periodically connect to the network to achieve the push effect. If the server does not have new content for a long time, the client terminal will still send an access request and an update query to the server cyclically. This mechanism is simple and controllable, and has low cost of hardware deployment. However, if the time interval between two requests is too short, the client's power consumption and network traffic will be greater. Meanwhile, if the time interval is too long, it will make the push meaningless. And thus this mechanism is only applicable to applications with low real-time demanding.

B. *MQTT (Message Queuing Telemetry Transport)*

MQTT is an open, simple, lightweight pub/sub protocol developed by IBM [6]. As early as 1999, Dr. Andy Stanford-Clark of IBM and Dr. Arlen Nipper of Arcom Company invented MQTT technology. MQTT protocol is particularly suitable for robot device with low bandwidth, poor network environment, high network cost and limited processor and memory resources. The protocol uses a small transmission, where the power consumption is small and the network flow is greatly reduced. It has now expanded a number of MQTT server programs that send relevant messages to MQTT through PHP, JAVA, Python, C, C# and other system languages. Three roles are defined in the MQTT protocol: message publisher, message broker, and message subscriber [7]. The client, as a message subscriber, first subscribes to the relevant task topic to the message broker server, and when the client of the message publisher publishes a message to a certain topic on the message broker server, the message broker server will take the initiative to push the message to the client that has subscribed the topic. Fig. 1 is the message push schematic diagram based on the MQTT protocol.

The MQTT protocol has the following features:

- It uses asynchronous pub/sub message mode, providing one-to-N message distribution and decoupling the application program;
- MQTT protocol provides network connections based on TCP/IP, so it's more reliable in the anti-control device than the CoAP and other UDP-based protocol;
- MQTT protocol has a very small communication overhead, with the shortest message only containing two bytes. Protocol exchange minimization can reduce the transmission traffic of the network;
- It supports three different quality of services [8]. QoS0, namely "up to one time", where the message is completely dependent on the underlying TCP/IP network, will result in the message lost or repetition. This level applies to situations where the network state is poor and the loss of a single data does not affect the overall results, such as the use of sensors to collect environmental data. QoS1 is "at least once". It ensures the message's arrival, but may leads to message duplication for general network status. QoS2 is "only once" to ensure that the terminal only receives the message once. It is used when the loss and duplication of the message will cause the wrong results, such as billing system;
- Message transmission for load content masking;
- The protocol supports the "last will" mechanism. After the client connection is disconnected due to non-normal reasons such as poor network status, other client users who may be interested in the user's state are notified in the form of a published topic according to the user's set of will mechanism.

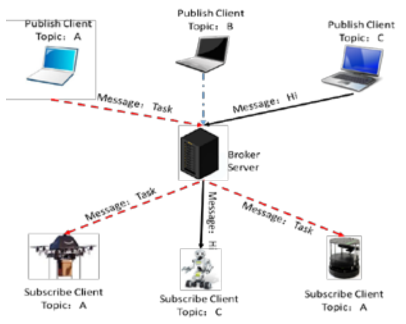


Figure 1. Schematic diagram of message push based on MQTT

C. *Robot Operating System (ROS)*

The open source robot operating system ROS, released by Willow Garage in 2010, is actually a robot software development platform [9]. ROS's pub/sub mechanism has greatly improved code reuse rate, and it provides service similar to the operating system which greatly reduces the cost of the robot in software development. Compared to other common robotic operating systems, ROS has the characteristics of point-to-point design, multi-language support, simplicity and integration, as well as rich, free and source-open toolkit [10], and is favored by many developers and robotics companies. It is also widely used in the academic field, such as SLAM, three-dimensional reconstruction and other aspects. ROS provides a software development platform with unified interface for the robotic programming that allows developers and researchers to focus on the functional development of application, without having to care about the specific implementation of underlying hardware drivers and control technology.

III. SYSTEM ARCHITECTURE

Robot task distribution system in large scale is divided

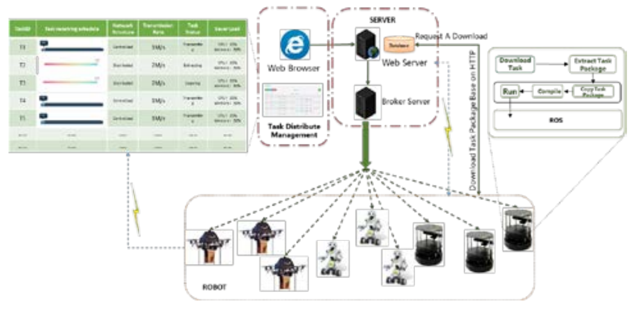


Figure 2. System architecture for large-scale robot task distribution and deployment

into four modules of task distribution, task reception, task execution and task distribution management. Fig. 2 shows the overall architecture of the system.

Among them, the task distribution module mainly uses the "Push-Pull" distribution mechanism, composed of two servers, namely the Web server and MQTT protocol message broker server. The Web server, as the message publisher, is responsible for publishing the robotic task package information to the message broker server, while the message broker server as the middleware is responsible for pushing the task package information to the robot's client. The client on the task receiving module is responsible for receiving the task package information distributed by the server and analyzing the received task package information to download the package. Task execution module builds the ROS work environment on the robot, and when the task package is fully received, this module is responsible for compiling and executing the task package in the ROS environment. The task distribution management module is used to monitor the reception of the robotic task package, and that whether the task packages are correctly extracted and copied, as well as the network and server condition.

IV. "Direct-Push" and "Push-Pull" Distribution Mechanism

A. *"Direct-Push" Distribution Mechanism*

The traditional polling mechanism cost much power and flow of the terminal, and it is easy to cause unnecessary waste of resources of the server [11]. The push mechanism based on MQTT protocol can solve these problems well. The principle of "Direct-Push" distribution mechanism is that the message publisher packs the task first, and then publishes the task package to the corresponding task package topic of the message broker sever based on MQTT, and finally the broker server pushes the task package to the robot who has subscribed to the corresponding topic. Fig. 3 shows the schematic diagram of "Direct-Push" distribution mechanism.

The robot distribution process based on "Direct-Push" distribution mechanism is as follows:

- The message broker server first calls the system initialization module to start the service function of the message broker server. The system message publisher and the robot device as the message subscriber are connected to the message broker server through the initialization module of the message broker server.
- The robot devices need to provide their own MAC address initiatively to complete the registration and obtain the registration account. Then pub/sub their own topic after successfully connecting to the MQTT message broker server. The subscription table management module of the broker server records the corresponding subscription topic and the account identification of the unmanned equipment that subscribes to the topic.
- The system message publisher will first push the task package, which is packed into the form of message, to MQTT broker server. The broker server stores the message into the system reception queue according to the priority. Then the message processing module gets the message out of the queue and pushes the message to the corresponding massage reception queue according to the subscription information of robot device recorded in subscription table management module.
- The message listener of the MQTT message broker server is triggered to warn the message subscriber to receive the message according to the message subscriber account.
- Message subscriber (robot device) will read the message from the respective message reception queue when listening to a new message in the message queue.

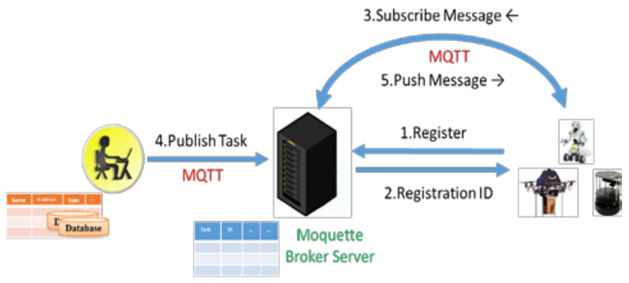


Figure 3. Schematic diagram of "Direct-Push" distribution mechanism

B. *"Push-Pull" Distribution Mechanism*

Although the "Direct-Push" distribution mechanism has a great advantage in the message push field, MQTT protocol does not allow message fragmentation due to the design of the protocol itself which hampers the transmission of large messages in constrained environments [12]. In this paper, we propose a distribution mechanism named "Push-Pull" for robot task distribution in large scale. In this mechanism, the task package link is pushed to the robot device by the "Direct-Push" distribution mechanism, and then, the robot device uses the traditional pull mechanism to request actively for downloading the task package according to the push link. The distribution mechanism, which combines MQTT and HTTP, has the advantages of both MQTT and HTTP, that is, it has the advantage of timeliness, efficiency of MQTT and stability of HTTP, and overcomes the respective technical limitations of MQTT and HTTP. So it can ensure the quality and efficiency of large task package distribution. Fig. 4 shows the working process based on "Push-Pull" distribution mechanism based on MQTT and HTTP.

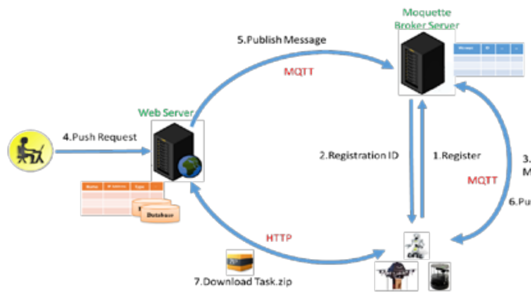


Figure 4. Schematic diagram of "Push-Pull" distribution mechanism

The robot distribution process based on "Push-Pull" distribution mechanism is as follows:

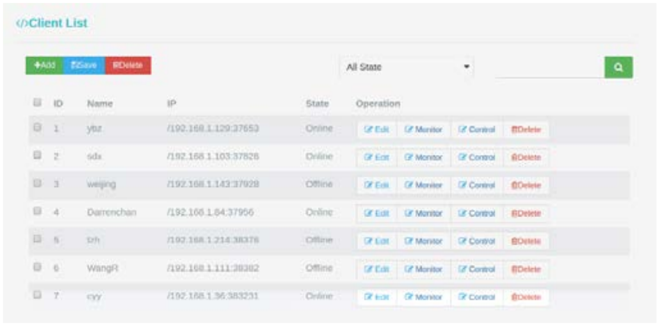
- The system task publisher first sends the push request to the Web server. Then the Web server, as the message publisher, pack the task package link into message format, and push the packed message to the robot device through "Direct-Push" distribution mechanism.
- The message subscriber (robot device) will take the initiative to send query request to the Web server and download the task package after receiving task package link.
- After completing downloading the task package, the robot will take the initiative to send a response to the Web server to inform the system task publisher of the task package's current status.

V. System Implementation

A. *Server Implementation*

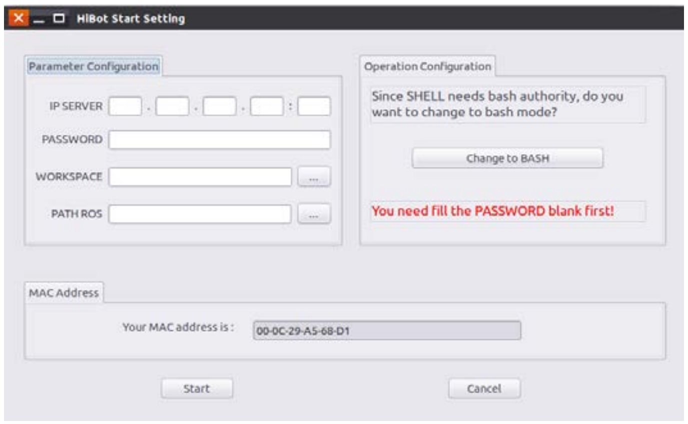
The server of the whole distribution system is composed of web server and push server. Because the Java language has the advantages of cross platform and good running efficiency [13], the whole task distribution system is written in JAVA language. Moquette [14], which supports websocket, SSL and other functions, is a Java MQTT broker based on an event model with Netty. In this paper, we use Moquette as a push proxy server, customize in its Moquette-0.8 version, and introduce a database to store the task package. Moquette supports three quality of service of the MQTT protocol: QoS0, QoS1 and QoS2, and the quality of service QoS1 is applicable to the general situation of the network, to meet the system requirements to ensure the timely and accurate delivery of task packages to the client.

Web server adopts Jersey framework and MVC coding style, making the server software architecture more concise and hierarchical. Jersey framework is the purest REST service development framework in Java domain [15]. Compared with the traditional Web Service, Java Restful Web Service has the advantages of less resource consumption, simple implementation and high efficiency. Web server also added a friendly task distribution management module, and the visual interface can view the state of the robot task package. The MAC address of each robotic device is the robot's unique identifier in the system. After selecting the robot that needs to be distributed the tasks in the interface, the background will push the task link to the robotic device and monitor the transmission progress of the task package in real time. Fig. 5 shows task distribution management interface in the web server.



ID	Name	IP	State	Operation
1	ybz	/192.168.1.129:37653	Online	Edit Monitor Control Delete
2	sdx	/192.168.1.103:37826	Online	Edit Monitor Control Delete
3	weijing	/192.168.1.143:37928	Offline	Edit Monitor Control Delete
4	Darrenchan	/192.168.1.84:37956	Online	Edit Monitor Control Delete
5	trh	/192.168.1.214:38376	Offline	Edit Monitor Control Delete
6	WangR	/192.168.1.111:38382	Offline	Edit Monitor Control Delete
7	cyy	/192.168.1.36:383231	Online	Edit Monitor Control Delete

Figure 5. Task distribution management interface in the web server



Since SHELL needs bash authority, do you want to change to bash mode?

You need fill the PASSWORD blank first!

Your MAC address is : 00-0C-29-A5-68-D1

Figure 6. Client configuration interface

B. *Client Implementation*

The main task of the client is to receive the task package link pushed from server, then to initiate a query request to the web server and download the corresponding task packet. The task arrived at the terminal needs to be run on the ROS platform, so that some of the parameter information is required when the terminal is first added to the system. The system provides the parameter setting interface on the client. The client software will automatically identify the MAC address of the robot device terminal, and write the

Figure 7. Experimental verification scenario

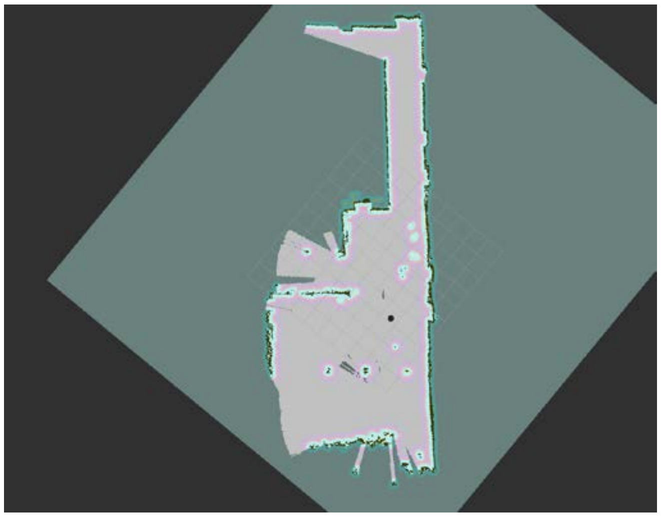


Figure 8. Experimental verification map

parameters submitted by the user into the property file. Fig. 6 is the client configuration interface. When the client receives a complete task package pushed from server, it will take the initiative to send the packet to the server in response to receiving complete task package, so that the server can monitor tasks to distribute packages to the terminal robotic device in real time.

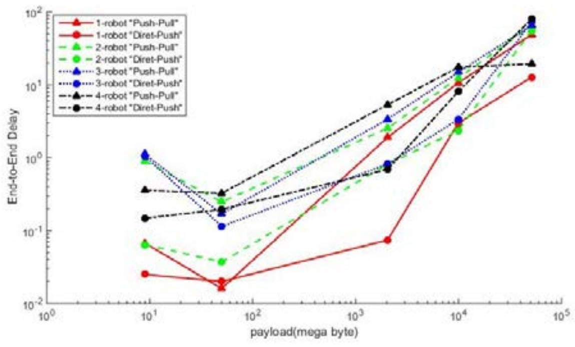
VI. EXPERIMENT AND TEST

A. Experiment

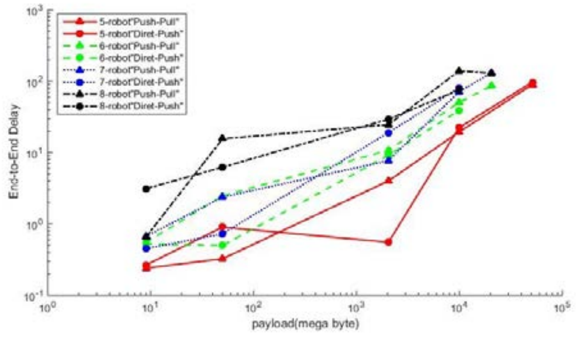
The deployment of the task package on the ROS platform involves four processes: extracting the task package, copying the task package to the ROS workspace, generating the shell script, and finally running the shell script. In order to verify the integrity and reliability of the system, we use the turtlebot2 robot and select Gmapping map building task package based on the FastSLAM algorithm to have an experiment on distribution and deployment of robot task in the laboratory corridor in this paper. Fig. 7 shows an experimental validation scenario that is similar to the "L" shape. Fig. 8 shows a map of the Gmapping map building task package that is distributed through the system in this scenario. The task package can be executed correctly and build a map which is more consistent with the experimental scene.

B. Test Method

In this paper, the test metrics are transmission load, message arrival rate and message delay time. Network

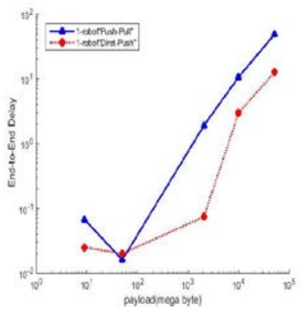


(a) 1 to 4 robots

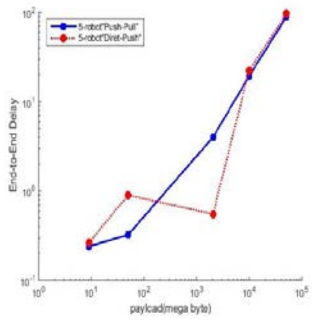


(b) 5 to 8 robots

Figure 9. Relationship between transmission load and end to end delay (QoS=1)

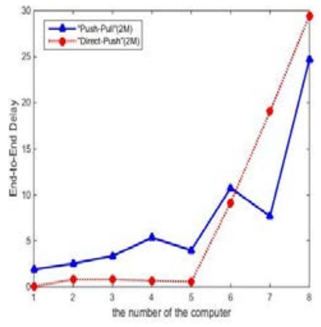


(a) Number of robots is 1

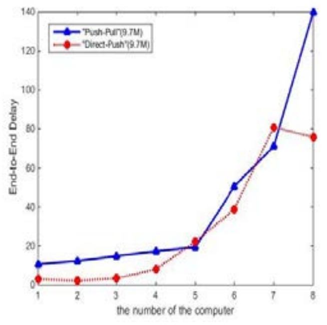


(b) Number of robots is 5

Figure 10. Relationship between transmission load and end to end delay (the number of robots is 1 and 5)



(a) Payload=2M



(b) Payload=9.7M

Figure 11. Relationship between the number of robot devices and transmission delay (Payload=2M and 9.7M)

capture software Wireshark can realize two kinds of distribution mechanism of delay test [16]. This test is carried out in the no-load state, that is, all devices are connected to the same local area network, in the same network environment to distribute the same content. Furthermore, in order to show the efficiency of two kinds of distribution mechanism more clearly, we use the logarithm of the parameter value as the coordinate value.

C. Test Results and Analysis

Fig. 9 is under the circumstance of QoS1 service quality level, we test the end to end delay of the two distribution mechanisms when the transmission load is increasing on 1 to 5 robotic devices. It can be seen from the figure, when the package reaches 100M, the client that uses the "Direct-Push" distribution mechanism has lost connection with only 1 robotic device. When the terminal equipment is increased to 3 units, with the increase of the transmission load, the "Push-Pull" distribution mechanism gradually shows its advantages, and the delay is less than the "Direct-Push" distribution mechanism. When the number of terminals reached 5 units, "Push-Pull" distribution mechanism is getting better.

Fig. 10 is the relationship between the size of the transmission load and the end-to-end delay when the task is distributed to the 5 robots. The figure shows that, with the increase of the size of the transmission load, the "Direct-Push" distribution mechanism appears unstable phenomenon, and the distribution efficiency of "Push-Pull" distribution mechanism is significantly higher than that of the "Direct-Push" distribution mechanism.

The task distribution test in large scale refers to the change of the end to end transmission delay with the increase of the number of robotic devices in the case of the same transmission load. Fig. 11 shows the relationship between the number of robots and the transmission delay of the two distribution mechanisms. As the figure shows, with the increase of the number of robotic devices, "Direct-Push" distribution mechanism also appears unstable phenomenon, resulting in client disconnection or transmission task package file in longer time. While the "Push-Pull" distribution mechanism can stably distribute task packages to the robot terminal, and with the increase of the number of robots, end-to-end transmission delay also begins approaching or even less than "Direct-Push" distribution mechanism.

Through the analysis of the above data, we can see that of two kinds of task distribution mechanisms have different advantages in the different transmission environment. Compared with the traditional polling mechanism, the distribution mechanism based on the MQTT protocol can reduce the number of interactions between the client and the server, reduce the burden on the server side and shorten the response time of the server. Meanwhile, "Push-Pull" distribution mechanism compared to the "Direct-Push" distribution mechanism has higher stability, and with the increase of transmission load and robotic device, the transmission delay of the "Push-Pull" distribution mechanism is gradually showing the advantage.

VII. Conclusions and Future Work

The intelligent robots have huge development space because of the rapid development of the Internet of things, big data, and human-computer interaction. The task distribution system for large-scale robots can not only save the network traffic, but also ensure the timeliness and integrity of the robot device to obtain the task package. Traditional polling mechanism can easily cause network congestion and waste of system resources. However, the distribution mechanism based on MQTT protocol is not only lightweight and simple, but also can accurately distribute task packages according to the features of the robot. Also, it occupies less network resources and pushes timely and safely.

In this paper, we make an intensive study on the current push mechanism, and propose a "Push-Pull" distribution mechanism based on MQTT and HTTP. Then we make a comparative experiment with the "Direct-Push" distribution mechanism based on MQTT protocol. Finally, a task deployment system is designed for large-scale robot, which has the advantages of timelines and stability and especially suitable for the scenarios of simultaneous task distribution.

The system still needs to be improved. The main work of next step is to realize the rule-based adaptive distribution. Since the system needs to support large-scale task distribution, it is necessary to build up server cluster to support the access of more robots and add load balancing mechanism to enhance the reliability and stability of the server.

Acknowledgment

The authors would like to express their gratitude towards National Laboratory for Parallel and Distributed Processing for providing with the research resources.

References

[1] Buckingham R, Graham A, Arnarson H, et al. Industrial Robot: An International Journal[J]. Industrial Robot An International Journal, 1973, 28(6):498-503.

[2] Hasegawa H, Mizoguchi Y, Tadakuma K, et al. Development of intelligent robot hand using proximity, contact and slip sensing[C]// IEEE International Conference on Robotics and Automation. IEEE Xplore, 2010:777-784.

[3] Kehoe B, Patil S, Abbeel P, et al. A Survey of Research on Cloud Robotics and Automation[J]. IEEE Transactions on Automation Science & Engineering, 2015, 12(2):398-409.

[4] Jung H. Study on the MQTT protocol design for the application of the real-time HVAC System[J]. International Journal of Internet Broadcasting & Communication, 8.

[5] Sun Z J, Chan X F, Center N C, et al. Applied Research of XMPP Push Technology in Mobile OA[J]. Research & Exploration in Laboratory, 2015.

[6] Tang K, Wang Y, Liu H, et al. Design and Implementation of Push Notification System Based on the MQTT Protocol[J]. 2013, 92:116-119.

[7] IBM, Eurotech. MQTT V3.1 Protocol Specification.

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqttv3r1.html, 2010-08-24.

[8] Lee S, Kim H, Hong D K, et al. Correlation analysis of MQTT loss and delay according to QoS level[C]// International Conference on Information NETWORKING. 2013:714-717.

[9] Quigley M, Conley K, Gerkey B P, et al. ROS: an open-source Robot Operating System[C]// ICRA Workshop on Open Source Software. 2009.

[10] Álvaro GarcíaPiquer, Colomé J. Using Robotic Operating System (ROS) to control autonomous observatories[C]// SPIE Astronomical Telescopes + Instrumentation. 2016:99132V.

[11] Guo K, Liu Y, Ma J. AMPS: An Adaptive Message Push Strategy for Ubiquitous Terminals[J]. Computer Journal, 2015, 58(6).

[12] Caro N D, Colitti W, Steenhaut K, et al. Comparison of two lightweight protocols for smartphone-based sensing[C]// IEEE, 2013:1-6.

[13] Eckel B. Thinking in Java[M]. China Machine Press, 2007.

[14] Moquette[EB/OL].[2017-2-5]. https://github.com/andsel/moquette.

[15] Li H. RESTful Web service frameworks in Java[C]// IEEE International Conference on Signal Processing, Communications and Computing. IEEE, 2011:1-4.

[16] Wireshark.org, “Wireshark,” http://www.wireshark.org/, 2013.