A Distributed MQTT Broker System for Location-based IoT Applications

Ryo Kawaguchi and Masaki Bandai
Graduate School of Science and Technology, Sophia University
Yombancho, Chiyoda, Tokyo, 102-0081 Japan
Email: r-kawaguchi-m4w@eagle.sophia.ac.jp

***Abstract*—Message queue telemetry transport (MQTT) is well known as an application layer communication protocol for the Internet of Things (IoT). Since basic MQTT defines one broker in a system, the number of publishers and subscribers is limited. Therefore, there are MQTT implementations that perform load balancing via cooperation with multiple brokers. However, when subscriptions frequently change for mobile clients, as is the case when handling location-dependent data, or when the number of clients is large, information sharing between brokers increases. In addition, with existing implementations, different kinds of broker cannot cooperate. In this paper, we propose a distributed broker system for large-scale location-based IoT services. The proposed method introduces a topic structure suitable for handling location-dependent data. In addition, distributed brokers and gateways are introduced to reduce broker load and to support heterogeneous brokers in a system. A prototype implementation and a theoretical evaluation are used to show the effectiveness of the proposed system.**

I. Introduction

With the spread of the Internet of Things (IoT), data collected from a massive number of IoT devices are expected to be used for sophisticated services, such as smart cities, disaster detection, and traffic management [1]. In particular, it is important to collect location-dependent data [2]. For example, data acquired by environmental sensors, such as temperature, humidity, and carbon monoxide concentration, are more useful if the sensor location is known. There are two major requirements for large-scale location-based IoT services: scalability and heterogeneity.

For scalability, reducing the number of messages in a network is important. Message queue telemetry transport (MQTT) [3] is a promising application protocol for IoT. MQTT is a simple and lightweight messaging protocol that supports topic-based publish/subscribe-type communication. In MQTT, publishers send messages such as sensing data to subscribers via a broker. Each message has a topic, which is used by a subscriber to subscribe to a broker. When a broker receives a message from a publisher, it forwards it only to subscribers of the message topic. Since basic MQTT defines one broker in a system, the number of publishers and subscribers is limited. Some researchers have introduced multiple brokers and broker load balancing to realize large-scale systems.

JoramMQ [4] introduces cluster brokers and distributed brokers. Cluster brokers broadcast all published messages to the other brokers when they receive a message. Cluster brokers can receive all messages in the system. Distributed brokers are arranged based on the hierarchy of topics. They transfer messages between high- and low-order brokers. HiveMQ [5] is also a cluster-based system. In HiveMQ, brokers share subscription information with other brokers. Brokers forward received messages only to brokers that need it. Mosquitto [6] is a well-known implementation of an MQTT broker. It uses a mechanism called a bridge to transfer specific topics to specific brokers. A broker implementation for an edge environment has been presented [7][9].

All of the above implementations allow sharing information among brokers or nodes. Therefore, when subscriptions frequently change for mobile clients, as is the case when handling location-dependent data, or when the number of clients is large, information sharing between brokers increases.

In addition, location-dependent data are typically used where the data were occurred. A publish/subscribe system has thus been developed for efficiently using a network considering of data locality [8]. Also, to reduce message latency, one study measured the proximity in the network and route messages to the appropriate broker [9].

For heterogeneity, a mechanism that allows heterogeneous brokers to communicate is necessary. The appropriate broker implementation may depend on the edge environment [7]. However, with existing implementations, different kinds of broker cannot cooperate. The study introduces intermediate nodes between the broker and the client. Publish message and subscription information are shared between nodes. This allows heterogeneous brokers to cooperate.

In this paper, we propose a distributed broker system for location-based IoT services. The proposed system is specialized for handling location-dependent data. In the proposed method, a topic structure suitable for handling location-dependent data is used. In addition, distributed brokers and gateways that connect clients and the system are introduced to reduce broker load and to support heterogeneous brokers in a system. We implement a prototype of the proposed system on Amazon Web Services (AWS) and local machines. Moreover, we theoretically compare the number of messages among brokers to confirm the effectiveness of the proposed system.

II. Proposed method

In this paper, we propose a distributed MQTT broker system for large-scale location-based IoT applications. The proposed method has three elements:

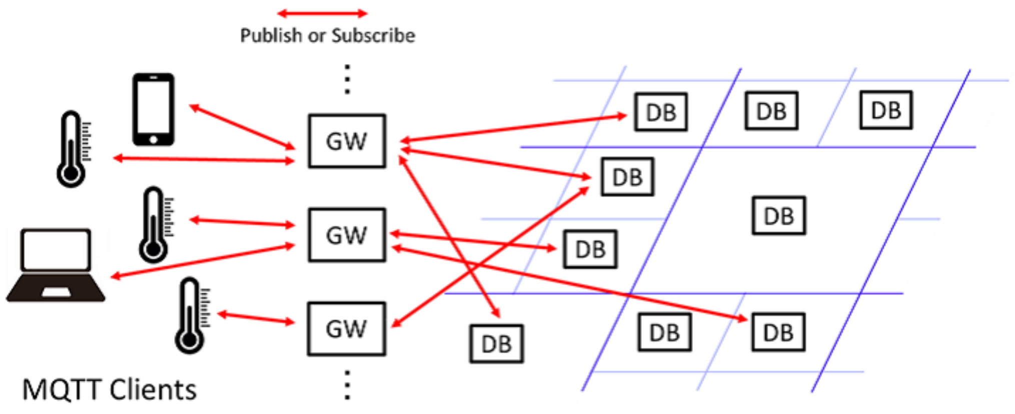

Fig. 1. System architecture of proposed system

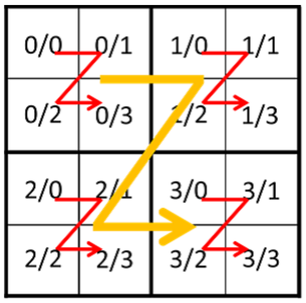

Fig. 2. Geographic representation in proposed system ($k = 2$)

- Topics with embedded geographic information
- Geographically distributed brokers (DBs)
- Gateways connecting DBs to clients

Figure 1 shows the system architecture of the proposed system. The system consists of MQTT clients, DBs, and gateways. Clients are publishers and subscribers of location-dependent data. Geographically placed DBs are in charge of topics in a nearby area. Gateways work as proxy brokers between clients and DBs.

A. Topic notation

The topics for MQTT have a hierarchical structure, where levels are indicated by "/". In the proposed system, the topic notation is as follows: *(geographical representation based on Z-ordering)/(unlimited part)*

Z-ordering [10] can map two-dimensional geographic information, eg. (x, y), to one-dimensional information, which is used to divide a square area hierarchically. In Z-ordering, the target square area is initially divided into four areas. Each divided area is named 0 to 3 in order of the drawing Z of the alphabet. By repeating this operation recursively k times, a square area is divided into 2^{2k} areas.

Figure 2 shows the geographic representation based on Z-ordering at $k = 2$. In the proposed system, we separate k numbers given by Z-ordering with "/", which indicates geographic representation, e.g., "1/0/temperature" for $k = 2$. Using this format, subscribers can easily specify location-based information. Moreover, the proposed system can use the wildcard "+" when specifying a topic, as well as normal MQTT. For instance, "1/+/temperature" is equivalent to "1/0/temperature, 1/1/temperature, 1/2/temperature and 1/3/temperature". Clients specify a message using a topic that includes hierarchical geographic information.

B. Distributed brokers

Based on a representation of Z-ordering, one DB is placed in each square area constituting the first l-th ($1 \leq l \leq k$) digit. l is defined as the broker level. A DB is in charge of topics of all levels above l. The broker level for all areas at level k is not necessarily the same. A large l is set for a DB if the area is expected to have a large number of clients (e.g., an urban area) to reduce the number of topics that a given broker is responsible for. This allows the load balancing of brokers. The closer that a DB is to its area of responsibility, the lower the latency of the topics within that area. However, DBs are not necessarily placed geographically near their areas of responsibility. In addition, each DB operates independently; i.e., there is no information sharing among DBs.

C. Gateways

Gateways are logical brokers that connect a client and a DB. Gateways work as proxy brokers. Clients regard a gateway as a broker. Gateways can uniquely determine the brokers they are responsible for from the topic name included in publish or subscribe messages. Gateways publish/subscribe to a DB on behalf of clients. With this mechanism, no information shearing is necessary between DBs. Since gateways are ordinary MQTT clients, they operate within the default range of the protocol. Therefore, there is no need to unify a specific MQTT implementation in a system.

Gateways only use the features of an ordinary MQTT client. They look like brokers from the perspective clients, and like normal MQTT clients from the perspective of DBs. Therefore, the proposed system is independent of the broker implementation, and can thus support heterogeneous brokers in a system.

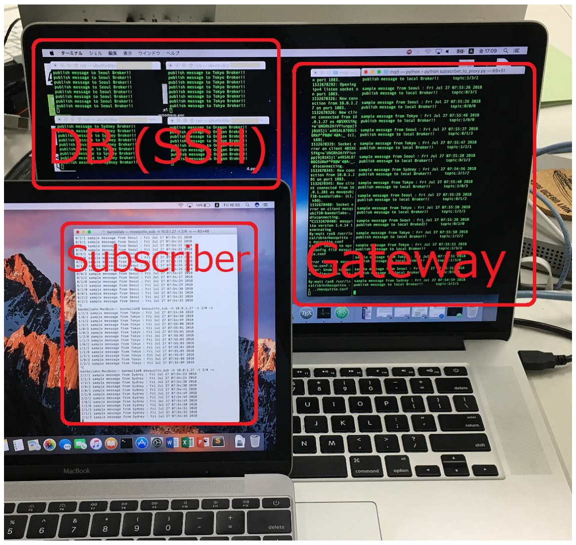

Fig. 3. Photograph of implementation

III. Prototype implementation

We implemented a prototype of the proposed system. Four DBs were implemented on an AWS EC2 t2.micro instance. Mosquitto was used for four regions: Tokyo, Seoul, Sydney, and Oregon. A gateway was implemented on a MacBook Pro (CPU: Intel Core i7, 2.6 GHz; RAM: 16 GB) in Tokyo. In the prototype, $k = 3$ and $l = 1$. The DB of the Seoul region was responsible for the topic 0. A table of the IP addresses and port numbers of the DBs was registered in the gateway in advance. The gateway used Mosquitto for client connections. Client were described using the Paho MQTT Python client library for publishing/subscribing to a DB. The Mosquitto clients were subscribers to the gateway on the MacBook Pro.

Figure 3 shows a photograph of the implementation. The computer in the back is the MacBook Pro working as a gateway. The four terminal windows in the upper-left region of the screen of the MacBook Pro are connected to instances with DBs over secure shell (SSH). We prepared a virtual publisher and published to each DB. The two terminal windows on the right side show the gateway running. The computer in the front shows a terminal window running an ordinary MQTT client subscribed to the gateway.

We confirmed that the prototype works properly. The MacBook client subscribed to gateways for topics such as 1/# or 2/#. “#” is a wildcard that can be used in MQTT topics. If the subscriber specifies “#”, it will be subscribed to all topics at the lower levels. The messages from the publishers on AWS then reached the subscriber, as shown in Figure 3.

In addition, due to mediation by the gateway, there is message transmission delay. To quantify this delay, the time it took for a retain message to arrive after a client subscribed to its topic was measured with and without a gateway. Retain is a flag in MQTT to tell the broker to retain only the latest message of a specific topic. The client was implemented on Ubuntu 14.04 LTS (CPU: Intel Core i7, 3.4 GHz; RAM: 16 GB) in our laboratory. The measurement client used the Paho MQTT Python client library. A DB was placed in the Tokyo region. The gateway was implemented on the MacBook Pro. With the use of a gateway, the average of 100 delay measurements was 113.0 ms. Without the gateway, the average of 100 delay measurements was 56.9 ms. Therefore, the gateway added a delay of 56.1 ms.

IV. Performance Evaluation

We evaluated the scalability of the proposed system. The purposes of the proposed attempts to reduce the broker load in cases where a massive number of clients are connected or when frequent subscription changes occur in location-based IoT applications. Therefore, we focus on published messages from publishers and subscription change messages from subscribers and brokers. We calculated the number of transmitted and received messages by brokers . The proposed system is compared with the following two systems:

System 1) Broadcast published messages to all brokers
System 2) Multicast published messages to brokers that needs the topic

We assume that the number of hierarchies in Z-ordering is k. In each divided area, there are N_P publishers and N_S subscribers. Each publisher publishes a message every T_P. Each subscriber sends a subscription change message every T_S. A subscriber subscribes to a topic from another area with probability p.

A. System 1

In System 1, a broker broadcasts published messages to all brokers. JoramMQ and the PF algorithm (used between intermediate nodes) [7] are classified as System 1. A broker is placed in each divided area, and there are 4^k brokers in the system. Each client publishes or subscribes to the distributed broker in charge of the area in which the client exists. The number of published messages received by each broker per unit time and the number of messages due to a subscription change received are:

$$\mathit{Publish} : \frac{4^k \times N_P}{T_P}, \qquad \mathit{Subscribe} : \frac{N_S}{T_S} \tag{1}$$

In addition, the number of messages that the broker transmits per unit time is:

$$\mathit{Publish} : \frac{N_S}{T_P} + \frac{N_P}{T_P} \times (4^k - 1), \quad \mathit{Subscribe} : 0 \tag{2}$$

B. System 2

In System 2, a broker multicasts published messages to brokers that need the topic. HiveMQ is classified as System 2. Information can be shared among brokers via one-hop communication. There are 4^k brokers, and each client publishes or subscribes to the distributed broker in charge of the area in which the client exists. The number of published messages received by each broker per unit time and the number of messages due to a subscription change received are:

$$Publish : \frac{N_P}{T_P} + \frac{pN_S}{T_P},\ Subscribe : \frac{N_S}{T_S} + \frac{pN_S}{T_S} \times \frac{4^k - 1}{4^k} \tag{3}$$

In addition, the number of messages that the broker transmits per unit time is:

$$Publish : \frac{(1-p)N_S}{T_P} + \frac{pN_S}{T_P} \times \frac{4^k - 1}{4^k}, Subscribe : \frac{pN_S}{T_S} \tag{4}$$

C. Proposed system

In the proposed system, there are 4^k brokers. Each client publishes and subscribes to its nearest gateway. The number of published messages received by each broker per unit time and the number of messages due to a subscription change received are:

$$Publish : \frac{N_P}{T_P},\ Subscribe : \frac{N_S(1-p)}{T_S} + \frac{pN_S}{T_S} \times \frac{4^k - 1}{4^k} \tag{5}$$

In addition, the number of messages that the broker transmits per unit time is:

$$Publish : \frac{(1-p)N_S}{T_P} + \frac{pN_S}{T_P} \times \frac{4^k - 1}{4^k},\ Subscribe : 0 \tag{6}$$

D. Numerical results

We evaluated the performance of the proposed system using the expression derived in the previous section. We assume that $k = 4$, $T_P = 10$ seconds, $T_S = 30$ seconds, and $p = 0.2$. We evaluated the following three cases:

Case A) $N_P = 10000$, $N_S = 1000$
Case B) $N_P = 10000$, $N_S = 10000$
Case C) $N_P = 1000$, $N_S = 10000$

Table I shows the number of messages received by each broker per unit time. In System 1, since a broker receives all messages in the system, an enormous number of messages accumulate. Compared with System 2, the proposed system has almost the same performance when the number of subscribers is small (Case A). When the number of subscribers is large (Cases B and C), the number of received messages in the proposed system is greatly reduced.

Table II shows the number of messages transmitted by each broker per unit time. The proposed method has the smallest number of transmitted messages, especially for a large number of subscribers.

These results show that the proposed method outperforms existing broker implementations, especially when the number of subscribers is large or frequent subscription changes occur. From Table I and Table II, in Case 2, the total number of transmitted and received messages in System 2 is 1765.5 (=300+399.7+999.2+66.6). That of the proposed method is 1432.3 (=100+333.1+999.2+0). The difference between them is 333.2 (=1765.5-1432.3). Therefore, in Case 2, the proposed method reduced the number of transmitted and received messages by 18.9% compared to System 2.

TABLE I
NUMBER OF MESSAGES RECEIVED BY EACH BROKER PER UNIT TIME

Case A	Publish	Subscribe
System 1	256000	33.3
System 2	1020	40.0
Proposed	1000	33.3

Case B	Publish	Subscribe
System 1	256000	333.3
System 2	1200	399.7
Proposed	1000	333.1

Case C	Publish	Subscribe
System 1	25600	333
System 2	300	399.7
Proposed	100	333.1

TABLE II
NUMBER OF MESSAGES TRANSMITTED BY EACH BROKER PER UNIT TIME

Case A	Publish	Subscribe
System 1	255100	0
System 2	99.9	6.6
Proposed	99.9	0

Case B	Publish	Subscribe
System 1	256000	0
System 2	999.2	66.6
Proposed	999.2	0

Case C	Publish	Subscribe
System 1	26500	0
System 2	999.2	66.6
Proposed	999.2	0

V. CONCLUSION

In this paper, we proposed a DB system for large-scale location-based IoT services. The proposed method introduces a topic structure suitable for handling location-dependent data. In addition, DBs and gateways are introduced to reduce broker load and to support heterogeneous brokers in a system. A prototype implementation and a theoretical evaluation confirmed the effectiveness of the proposed system.

ACKNOWLEDGMENT

This work was partly supported by JSPS KAKENHI grant number 15K00136.

REFERENCES

[1] X. Sheng, J. Tang, X. Xiao and G. Xue, "Sensing as a Service: Challenges, Solutions and Future Directions," *IEEE Sensors Journal*, vol. 13, no. 10, Oct. 2013,
[2] L. Sharma, A. Javali, R. Nyamangoudar, R. Priya, P. Mishra, S.K. Routray, "An Update on Location Based Services: Current State and Future Prospects," In Proc. *IEEE ICCMC2017*, 2017.
[3] OASIS,"MQTT version 3.1.1 OASIS Standard 29 October 2014," http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf
[4] JoramMQ, http://www.scalagent.com/IMG/pdf/JoramMQ_MQTT_white_paper.pdf
[5] HiveMQ, https://www.hivemq.com/
[6] Eclipse Mosquitto, https://mosquitto.org/
[7] R. Banno, J. Sun, M. Fujita, S. Takeuchi and K. Shudo, "Dissemination of Edge-Heavy Data on Heterogeneous MQTT Brokers," In Proc. *IEEE CloudNet2017*, 2017.
[8] Y. Teranishi, R. Banno, T. Akiyama, "Scalable and Locality-Aware Distributed Topic-based Pub/Sub Messaging for IoT," In Proc. *IEEE GLOBECOM2015*, 2015.
[9] T. Rausch, S. Nastic, S. Dustdar, "EMMA:Distributed QoS-Aware MQTT Middleware for Edge Computing Applications," In Proc. *IEEE IC2E2018*, 2018.
[10] G.M. Morton, "A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing," *Morton1966*, 1966.