International Journal of Information and Communication Sciences
2019; 4(4): 70-78
http://www.sciencepublishinggroup.com/j/ijics
doi: 10.11648/j.ijics.20190404.11
ISSN: 2575-1700 (Print); ISSN: 2575-1719 (Online)

A Benchmarking Tool for Elastic MQTT Brokers in IoT Applications

Linh Manh Pham[1, *], Truong-Thang Nguyen[2], Manh-Dong Tran[2]

[1]Faculty of Information Technology, VNU University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam

[2]Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam

Email address:
linhmp@vnu.edu.vn (Linh M. P.), ntthang@ioit.ac.vn (Truong-Thang N.), dongtm@ioit.ac.vn (Manh-Dong T.)

[*]Corresponding author

To cite this article:
Linh Manh Pham, Truong-Thang Nguyen, Manh-Dong Tran. A Benchmarking Tool for Elastic MQTT Brokers in IoT Applications. *International Journal of Information and Communication Sciences*. Vol. 4, No. 4, 2019, pp. 70-78. doi: 10.11648/j.ijics.20190404.11

Received: October 16, 2019; **Accepted**: November 13, 2019; **Published**: November 22, 2019

Abstract: Cloud computing is an evolution in IT consumption and delivery which makes available self-management on the Internet with a flexible, pay-as-you-go business model. Within the context of Internet of Things, the MQTT (Message Queuing Telemetry Transport) protocol that is implemented broadly by the applications of "Publish-Subscribe" paradigm has a vital role. However, MQTT brokers are saturated easily if they have to cope with huge and speedy data generated by IoT "chatty" devices. With capability of provisioning/deprovisioning granular virtual resources, Cloud computing empowered MQTT brokers by enabling its elasticity feature. Elasticity helps the brokers deal with a very large variety of data integrated into the IoT every single day. However, there was lack of sturdy benchmarking tools that judge all the aspects of MQTT brokers in order to advocate correct elastic decision-making. This article focuses on the work of benchmarking MQTT by introducing a new developed tool called MQTTBrokerBench. With this tool, users not only can benchmark MQTT brokers but also can specify saturation points where the IoT load makes the brokers be saturated. Those saturation points can be used to set thresholds for elastic decision-making. Furthermore, the article also demonstrates the results acquired by this tool through the experiments on Windows Azure Cloud Platform.

Keywords: Benchmarking, MQTT, Cloud Computing, Internet of Things

1. Introduction

The Internet of Things (IoT) is currently changing into a reality, with the supply of low cost buried communication devices like RFID tags, wireless sensors, mobiles, etc. IoT offers new opportunities for firms from fashionable economic models (e.g. pay-as-you-go style) improving the standard of service (QoS) provided to their customers (individuals or companies) and meet their legal or written agreement obligations. It established itself as the new economical and productive tool (i.e. just-in-time) for e-agile businesses and organizations. The communication paradigm principally employed in the IoT infrastructure is that the "Publish-Subscribe" one (PubSub in brief) that disseminates mensuration messages collected by the device nodes and delivered to intensive applications [1]. Message Queuing Telemetry Transport (MQTT) protocol is that the most well-liked implementation of PubSub model in context of IoT normally and in context of Machine to Machine (M2M) specially [2]. With MQTT the messages are sent and oriented by topics, not by servers (a.k.a. brokers). A challenge of the IoT and MQTT is that the ability to supply a service with capability of dynamic resource management (the resources are going to be (de)provisioned supported actual demand). Thus, it is necessary to define the concept of "elastic topic" that may handle connections optimally. To outline the threshold, by that a broker will support in point of fact, it is necessary to realize the MQTT tests of performance.

In this paper, we have a tendency to create following contributions: (1) propose MQTTBrokerBench, a benchmarking tool which allows realizing numerous MQTT tests of performance; (2) offer an approach to analyse the results statistically by examination the definitions of MQTT in theory and to reveal scaling thresholds. The remainder of the paper is organized as follows. Section 2 presents the

general context where our proposal locates. The architecture and components of our developed tool are elaborated in Section 3. Section 4 describes substantiating experiments and discusses results from different scenarios that correspond to practice in IoT context. Once highlighting numerous related works in Section 5, we have a tendency to conclude and present future work in Section 6.

2. Context

2.1. Cloud Computing

Cloud computing is a term pertaining to a network model within which a program or application runs on one or several connected servers, rather than on a local computing device like a PC, tablet or smartphone. As the ancient client-server mode, a user connects to a server to perform a task. In additional detail, the "Cloud computing" refers to a machine or cluster of machines, ordinarily called servers, connected through a network like the Internet, an Intranet, a local area network (LAN) or wide area network (WAN). All users who have permission to access the server will use its computing power to run an application, store information or perform alternative computing tasks. Therefore, rather than employing a notebook computer to run the application, the individual can now run the application from anyplace within the world.

The distinction of "Cloud computing" is that the calculating process can be operated on one or multiple computers at the same time, using the ideas of virtualization. With virtualization, one or a lot of physical servers may be designed and divided into many virtually independent servers, referred to as virtual machines (VMs). All physical servers work independently and appear to the user as one physical machine. The VMs do not physically exist and can be moved or scaled up/down without affecting the end user. The information resources have become more and more granular, that provides benefits for the end users and the operators, including the demand for self-management, broad access through multiple devices, pooling of resources, speedy elasticity and the ability to service mensuration.

2.2. Internet of Things

The Internet of Things may be a network of networks that, with standardized systems and wireless electronic identification, identifies and communicates digitally with physical objects so as to live and exchange information between the physical and virtual worlds. IoT refers to single objects and their virtual representations in an Internet-like structure [3]. The new construct of IoT is a necessary foundation for the vision of a better planet (Smarter Planet - IBM) [4]. What is more, considering the looks of various devices referred to as "smart", the net is absolute to evolve IoT within the close to future. Let imagine some exceptional things we are able to do with IoT in next ten or twenty years:

1. A doctor will examine a patient in an exceedingly distant town and see the state of health in real time still as numerous data like vital sign or vital sign.
2. An energy company could monitor oil and gas pipelines placed many miles away and bring to a halt the flow if issues are detected.
3. An owner will see his house on an Internet page, via devices like a security alarm, a heat, and more.

IoT is even on the far side these examples. Devices act not solely with the users however additionally with one another, that might become a long type of central system.

2.3. Publish-Subscribe Paradigm

The communication paradigm "Publish-Subscribe" may be a model within which the message senders, referred to as Publishers, do not transmit their messages on to specific receivers, referred to as Subscribers [1, 5]. Instead, the Publishers send messages on the specified topics while not information of the existence or location of Subscribers. The Subscribers, in turn, receive messages from topics that interest them regardless the Publishers who sent them.

2.4. MQTT

Telemetry technology permits activity or dominant remote things. Moreover, today, enhancements within the measure technology enable inter-connected sensors and devices to be monitored at totally different locations still as scale back the price of making applications that may run on the good devices. People, firms and governments are turning to good appliances and measure technology to act a lot of showing intelligence with the planet.

MQTT provides measure technology to satisfy the challenges related to data exchange of the net stakeholders nowadays. MQTT is a very straightforward and light-weight electronic messaging protocol. Its PubSub design is intended to be simple to implement, with many thousands of remote clients could also be supported by one server. These options create the MQTT ideal to be used in strained environments, wherever information measure is low or high-latency, with remote devices that will have restricted memory or process capabilities. The construct of electronic messaging in MQTT communication rather more versatile than request/response attributable to biface asynchronous "push" communication, and therefore the decoupling of Publishers and Subscribers. This approach additionally makes the MQTT protocol notably appropriate to the association between machines (M2M), that is a necessary side of the emergence of the IoT construct.

MQTT has an open specification [2]. There are over forty totally different client and server implementations of this specification like Mosquitto, ActiveMQ, Apollo, Jboss, Joram, Carpet, RabbitMQ, HiveMQ. During this paper, we have a tendency to target MQTT protocol standardized by OASIS (Organization for the Advancement of Structured Information Standards).

2.5. MQTT with Elasticity

Elasticity is one in all outstanding action that Cloud computing brings to our world. Elasticity is capability to regulate granularly the virtual computing resources (i.e.

scaling out/in or up/down) to fluctuation of atmosphere in brief periods of your time. Nowadays, IoT applications sometimes get large information from numerous sources. Additionally, some applications need this information to be processed in real time to adapt the changes of giant data flows like text messages, social media posts, exchange feeds. Like said, the foremost fashionable communication technique in context of IoT is MQTT that advantages abundant from advantages of Cloud computing like elasticity. So as to try to to therefore, it is essential to possess the benchmarking tools to fret MQTT brokers that reveals the ultimate saturation points. These points are going to be valuable inputs for Cloud elastic engines to grant correct scaling selections like the engine proposed by Pham et al. [6]. Such a benchmarking tool, MQTTBrokerBench, is delineated in details through ensuing section.

3. MQTTBrokerBench

MQTTBrokerBench may be a distributed MQTT check platform for IoT applications, that follows main principles of the component-based design [7]. It is designed to recycle totally different COTS (component-off-the-shelf) parts and mix them flexibly. Therefore, the core of MQTTBrokerBench is unbroken light-weight with simplified APIs in operation as docks to port the plug-ins specific to the COTS parts. Figure 1 shows overall software package design of MQTTBrokerBench, that consists of many modules.

3.1. Test Performance Platform API

The "Test Performance Platform API" module is enforced in Java for generating and injecting load profiles on numerous systems and activity performance. It absolutely was originally designed for checking MQTT brokers in IoT applications however has been swollen to alternative test functions. We have a tendency to developed a plug-in for this module to plug to our Apache JMeter system, an open supply testing software package that supports portability naturally [8]. JMeter provides the flexibility to perform check of high load, and check of performance with differing kinds of services/protocols (Web - hypertext transfer protocol, HTTPS, SOAP, FTP, info via JDBC, Message Oriented Middleware via JMS, etc.). It permits the construct of multi-threading and may be accustomed accelerate the tests. JMeter's competitors are CLIF Server [9] and Gatling [10]. However, all of them lack a Load gadget for MQTT protocol and it is wherever we have a tendency to contribute our work.

So far, all developers were to jot down their own code snippets to check MQTT as needed. It is waste of your time for a developer who desires to review MQTT however cannot notice a tool to check his code utterly. If the developer does not notice any snippets that may facilitate him for the check, he has got to do the code himself. If he found any codes, extracted codes could not be essentially all-mains with reference to languages, systems operation and therefore the developer can still have to be compelled to rewrite their code. Alternative developers also will be needed to hold out identical add the long run. continuation identical work, for several folks, may be a waste of your time and energy. On the opposite hand, it is dangerous for a beginner to jot down his own code to check a protocol. And even for the developers, it is not attention-grabbing to put in all the mandatory environments, so that they will write some lines of codes solely to check MQTT. Currently, all API libraries for MQTT are in Java, therefore this creates a heavy downside for developers, who do not know Java however need to grasp MQTT. With a graphical tool, the testing for MQTT can abundant easier and easier. The user will try and perceive the MQTT protocol, then use it merely. The importance of MQTT protocol within the world of the IoT is the maximum amount as hypertext transfer protocol in the world of internet. Indeed, we have got already had sturdy tools for testing hypertext transfer protocol, what we have a tendency to miss may be a comprehensive tool for testing MQTT.

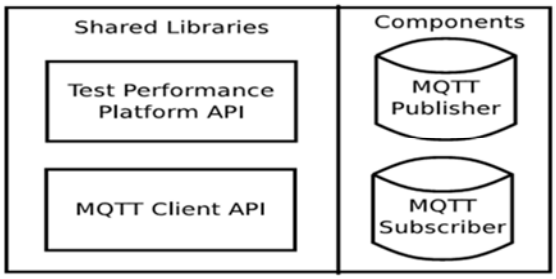

Figure 1. *Overall software package design of MQTTBrokerBench.*

3.2. MQTT Client API for Publishers and Subscribers

There are several check automation tools, however most of them are for the practical check. To boot, they are ineffective to emulate the mandatory traffic for the strain check. The "MQTT Client API" module is developed as a tool to simulate MQTT clients. We have a tendency to additionally developed interface for the client tool supported graphical user interface and CLI provided by JMeter. As a result of all the applications do not seem to be identical, every application should have tailor-made check eventualities to live actual real-world traffic performance [11]. We are able to use the graphical user interface to develop check cases and use CLI to perform the particular tests. Moreover, MQTTBrokerBench additionally provides a spread of graphical analysis performance reports. On the opposite hand, JMeter incorporates a Java API that is simple to develop for graphical user interface and every one alternative necessary parts. Therefore, we have a tendency to might target the looks of MQTT while not losing an excessive amount of time to make a brand new design graphical user interface or samplers for an unwell-known check tool. The MQTT client module includes 2 components: a Publisher that represents a client who publishes information to topics within the brokers, and a Subscriber that represents a client who receives information from broker in step with specific topics that he has signed.

With MQTT publisher, we are able to notice several eventualities of check because of its numerous choices, so all aspects of the protocol may be evaluated. The Publisher

involves four sub-modules: association data, Encoding, Option, and Content. The association data module permits us to enter the client id and a group of topics that we wish to publish to. It additionally allows the authentication, the clean session (server do not bear in mind its client once disconnection) and 2 totally different ways to publish: Random or spherical Robin. By Random strategy, the client publishes information to random topic at intervals the set of entered topics, otherwise, by spherical Robin strategy, the client publishes information to the topics in equal parts and in circular order. The encryption module permits us to send messages in type of non-coding, binary, base64, binhex and plain text with totally different codecs. The choice module allows capability to send preserved or not retained messages, to feature a timestamp or variety sequence to message. It permits to manage 3 levels of QoS: at the most once (0), a minimum of once (1), specifically once (2). The Content module provides four forms of messages to be published: a text, a generated price at intervals an optioned vary, a hard and fast price, or a random computer memory unit array. These four forms of message adapt to numerous eventualities of check, from the easy check (send a text) to the difficult one, in which, the messages represent the device information to be sent in little volume however in large amount. The MQTT Subscriber includes 2 sub-modules: association data and possibility which give identical properties for subscriber's and publisher's connections. Mqtt-client FuseSource API is additionally used since it provides American sign language a pair of Apache software package License API and it will pay attention of automatic reconnection to MQTT brokers and restore client sessions if network outages occur [12].

4. Evaluation

We conducted an entire performance check with multiple eventualities to validate the practicableness of MQTTBrokerBench.

4.1. Testbed

The IoT information are synthesized by configuring the Content module of MQTT publishers. Within the experiments, a MQTT publisher plays the role of the IoT entry between sensors and IoT applications implementing on the Cloud. The information browsing the gateways are delivered to the MQTT brokers.

The tests are deployed on Window Azure cloud platform (32 central processor licenses). The Azure Virtual Machines Service provides on-demand, scalable IT resources. An Azure VM may be a server within the cloud that we have a tendency to piece and maintain in step with our desires. It provides us the pliability of virtualization while not the price of buying and maintaining instrumentality to host it. With a VM in Azure, we are able to deploy the accessible versions of Windows Server or distributions of Linux operating system by selecting the preconfigured images. We have a tendency to can also load a Virtual magnetic disc (VHD) that contains a software package Server then use it to make VMs. Window Azure Platform additionally provides the potential of making and connecting multiple VMs in order that we are able to balance traffic between them. It uses each automatic and manual thanks to produce, manage and delete VMs. We are able to use the online portal (Azure Management Portal), Windows PowerShell or API for IaaS management. With them, we are able to delete and recreate VMs as again and again as necessary. For every VM, Ubuntu 16.04 LTS is employed as software package. Virtual Network Windows Azure provides us the flexibility to expand and handle the deployments as a natural extension of our network on web site. In our experiments, the VMs are deployed in an exceedingly ten.0.0.0/24 virtual network.

An important step within the benchmarking is to outline metrics. This can be done through a tool referred to as Sysstat [13]. It allows assembling information on system activities and treating them. The measured metrics are central processor load, idle CPU, Memory, I/O activity, Network statistics.

We used Mosquitto as MQTT message broker [14]. Mosquitto is free message broker software package (BSD license) that implements the MQTT protocol 3.1. Mosquitto provides a basic configuration, therefore, once putting in, it should be designed specifically for the corresponding options of what we wish to check. In our expertise, a haul sometimes arises once the tests are administered with nice variety of clients (e.g. thousands). Linux does not enable to exceed a most of outlined connections, therefore the tests do not provide a correct result. This comes from the very fact that Linux is designed by default to avoid a "Denial-of-service attack" (DoS). Therefore, we have a tendency to should increase the most variety of open file to 200000 in given that a socket association corresponds to one open file.

Another downside could seem once playing the check with various connections. There is a corresponding object in Java code (Fuse supply MQTT client API) to handle every association. Therefore, if there are thousands of connections, there will be thousands of created objects, in consequence, the JVM does not have enough Heap memory, it causes the breakdown of machine. Therefore, increasing the Heap memory by a configuration in MQTTBrokerBench is critical.

4.2. Scenarios

As mentioned, there is presently no open supply tool that measures MQTT brokers for IoT applications in cloud atmosphere. Therefore, we have a tendency to produce numerous check eventualities running on MQTTBrokerBench to validate the practicality of this tool. For the experiments, there are ninety generated check plans representing a minimum of 67.5 hours of experimentation on Azure cloud. Following three check eventualities represent the specialised forms of MQTT protocol.

4.2.1. Multi Publishers - Zero Subscriber

This situation implements the case of the many of Publishers, while not Subscriber to consume messages. It analyses the price of publication used for the second situation in 4.2.2 (see Figure 2).

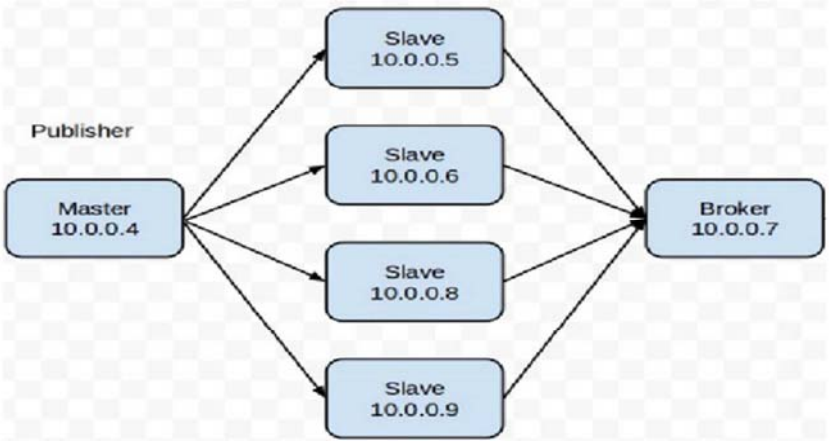

Figure 2. *Situation Multi Publishers - Zero Subscriber.*

4.2.2. Multi Publishers - Single Subscriber

This situation implements several of Publishers with one Subscriber. This corresponds to the case of employing a terribly sizable amount of sensors or sources of crowdsourcing (e.g. smart-phone) that measures are sent back to a central IT system (See Figure 3).

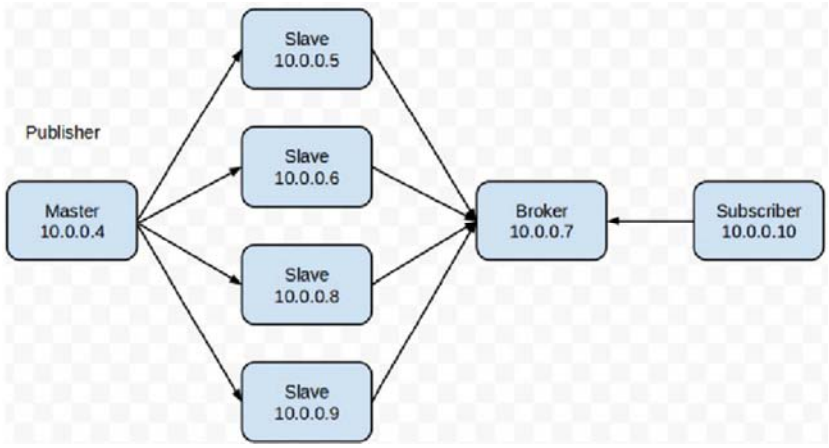

Figure 3. *Situation Multi Publishers - Single Subscriber.*

4.2.3. Some Publishers - Multi Subscribers

This situation uses a small number of Publishers and a large number of Subscribers. This corresponds to the case of using period of time notification of an oversized variety of smart-phones transmitted by a central IT system (See Figure 4).

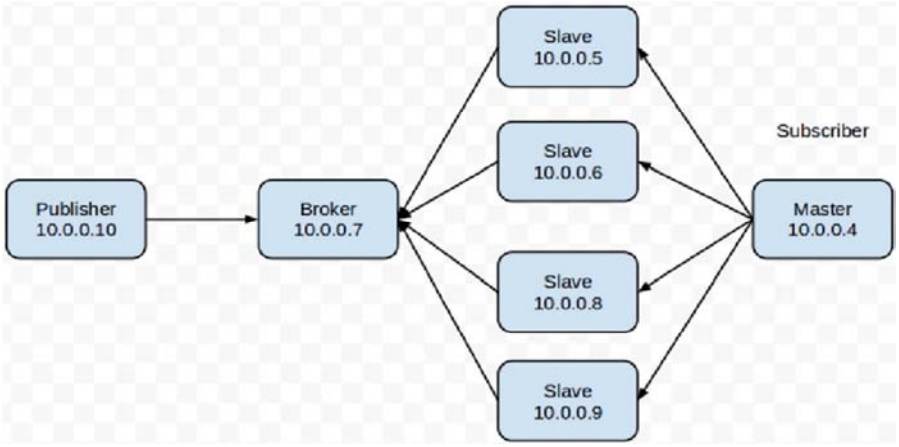

Figure 4. *Situation Some Publishers - Multi Subscribers.*

For the experiments, MQTTBrokerBench is employed in non-GUI mode. To achieve an oversized variety of clients in parallel, MQTTBrokerBench is additionally employed in a distributed manner, within which one master machine controls four slave machines (which are MQTTBrokerBench brokers). The test result is statistically recovered by MQTTBrokerBench summariser. The bash scripts are put in to live the metrics of all machines within the check situation. They are attack every machine to handle Sysstat remotely via SSH.

For each situation, a corresponding script is formed to come up with all the check set up. For instance, with the situation Multi Publishers - Single Subscriber, the script generated thirty check plans. The generation depends on variety and sort of parameters that have an effect on the brokers (QoS, Retained, Clean session, etc.). For instance, if testing with three QoS levels, the amount of check plans can increase three times compared to testing with one QoS level. Scripts are created in step with every situation. They are at the master machine (10.0.0.4) then the opposite VMs are controlled by SSH session. Logs, results, statistics, metrics are collected centrally and are retrieved for additional analysis.

The identity of clients (IDs) from publishers and subscribers collaborating within the experiments are generated indiscriminately with the weather of a group of 62 characters and numbers. With the IDs of length n, we are able to produce 62n cases. If we have a tendency to run the ID generation a hundred thousand times consecutively, the chance of getting 2 identical IDs is:

$$1 - (1 - 1/62^n)\,(1 - 2/62^n)\,\ldots\,(1 - 99999/62^n)$$

We can see that the chance to possess 2 identical IDs in tests with high variety of connections is sort of zero, which is able to not influence the check.

4.3. Result and Discussion

In this section, we have a tendency to discuss in details regarding results of 2 eventualities Multi Publishers - Zero Subscriber and Multi Publishers - Single Subscriber. The remaining situation is omitted attributable to restricted house.

4.3.1. Multi Publishers - Zero Subscriber

Figures 5 and 6 show an example during this situation with one master publisher (A3 instance, 4 cores, 1.6GHz, 7GB Ram), four slave publishers (also A3 instance, 4 cores, 1.6GHz, 7GB Ram), one target Mosquitto broker (A2 instance, 2 cores, 1.6GHz, 3.5GB Ram). Variety of threads in every slave are 2100. every thread performs five connections. Therefore, total variety of connections are 42000. The"2100.0.false" indication implies that variety of threads of every slave is 2100, QoS level is zero and false implies "Message no-Retained". In step with the graphs, we are able to conclude that with 3 levels of QoS:

1. The Mosquitto broker consumes the central processor in step with the order: QoS 0 > QoS 1 ≈ QoS 2.
2. The Mosquitto broker consumes the memory in step with the order: QoS 0 > QoS 1 > QoS 2.

This is explained by the definition of QoS in MQTT. With QoS zero the publishers publish their messages while not looking ahead to confirmation from the broker. The quantity of queued messages at the broker is therefore hugely fluctuated. The broker has got to method quickest and thus central processor load is largest. On the opposite hand, with QoS one and a pair of the graphs are a lot of stable. The quantity of queued messages is a lot more stable, therefore the result is more credible. We have a tendency to additionally get saturation points (where graphs get peak) and use them to specify thresholds utilized by the elastic engines for scaling selections.

4.3.2. Multi Publishers - Single Subscriber

In Figures 7 and 8, the "2100.0.falseSub.1.0.false" indicates variety of threads of every slave is 2100, QoS level is zero, false implies "Message no-Retained", Sub stands for Subscriber, QoS of the subscriber is zero and "Clean Session" is ready to false for the subscriber.

According to the graphs, it is found that within the situation Multi Publishers - Single Subscriber, the Mosquitto broker should method a lot of to send messages to the subscribers that has signed, therefore each metrics (CPU load and Memory) are consumed a lot of compared with the situation Multi Publishers - Zero Subscriber. In each eventuality, the QoS a pair of provides us the foremost reliable results and therefore the planned graphs therewith level of QoS are a lot of stable.

4.3.3. Discussion

In IoT atmosphere, handiness of services is that the key issue. For the environments using MQTT this implies that the publisher desires a decent and stable association to the broker. In alternative words, brokers should work all the time. During this context, how to avoid the waste of resources by making certain the performance and persistent of MQTT system? This leads the necessity of developing a brand new style of MQTT topic: "Elastic Topic". This style ought to enable us to unravel the matter of making certain the quantifiability of MQTT services to soak up an employment once the MQTT broker saturates or to get rid of brokers when the load decreases so as to avoid a waste of resources.

The horizontal elasticity may be employed in this context. Brokers are deployed by a PaaS (Platform as a Service) on the Cloud, and a VM operating sort of a load reconciliation device could distribute the MQTT connections from the employment. It will attempt to produce a brand new VM broker to handle connections once the present total employment exceeds a threshold. On the opposite hand, it will take away the superfluous VM brokers once the employment is small.

5. Related Work

The seamless and versatile network of everyday objects can become a crucial application field for Internet-based communication, in step with the vision of the net of Things. The performance of IoT systems may be an important issue,

particularly with period of time needs. For a few IoT systems using batteries like wireless device network, energy consumption may be an important concern. However, putting in a performance analysis atmosphere for IoT system is complicated and quite difficult to the developer. Thus, the necessity of a friendly and economical tool for benchmarking MQTT is crucial and below are some works on the point of our approach.

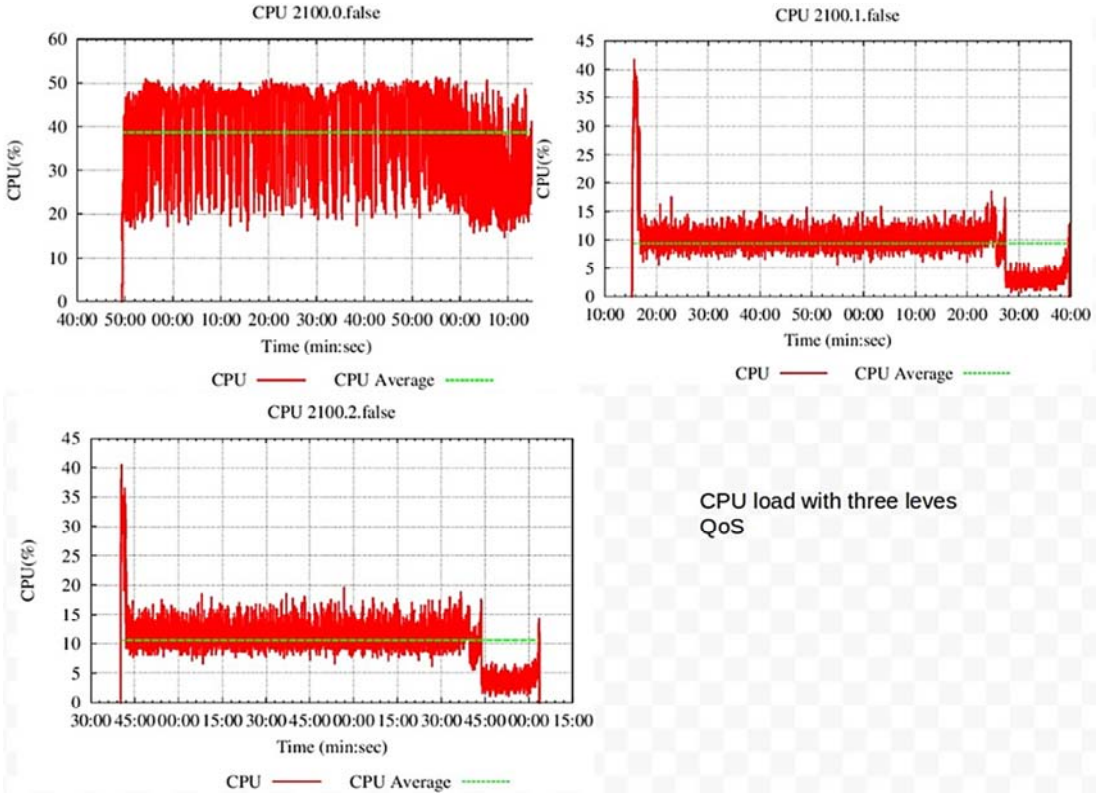

Figure 5. *CPU load of the broker in situation Multi Publishers - Zero Subscriber.*

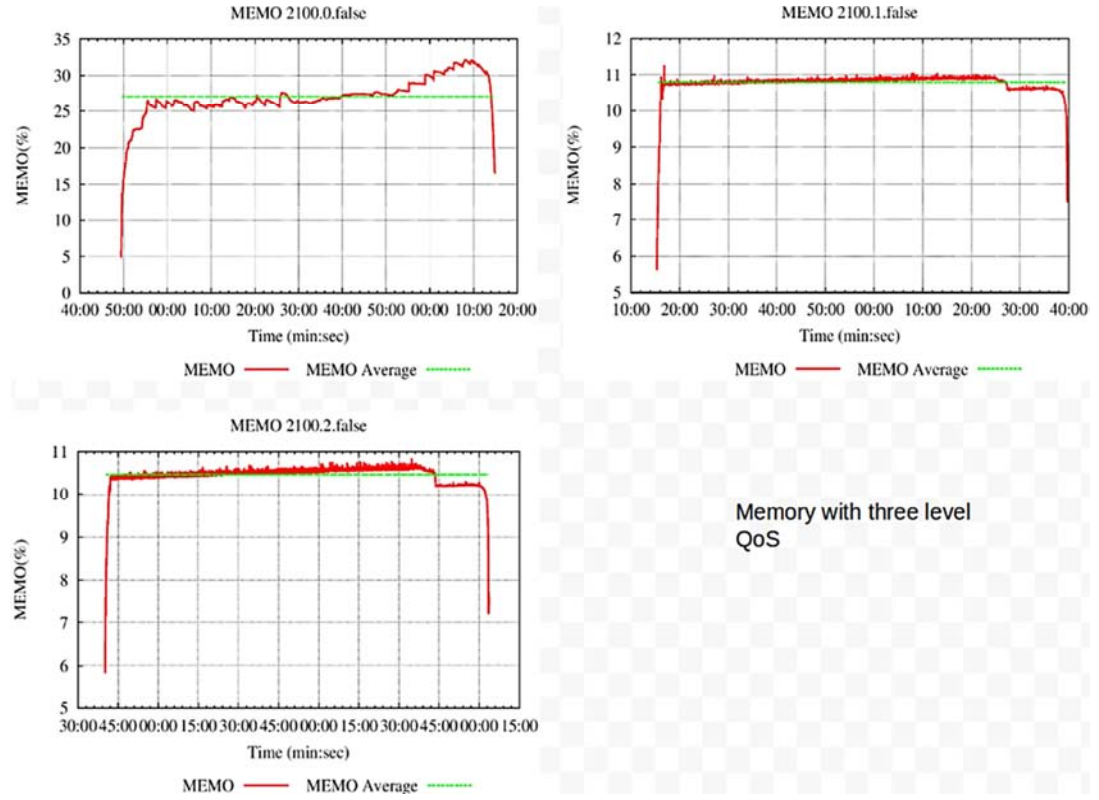

Figure 6. *Memory of the broker in situation Multi Publishers - Zero Subscriber.*

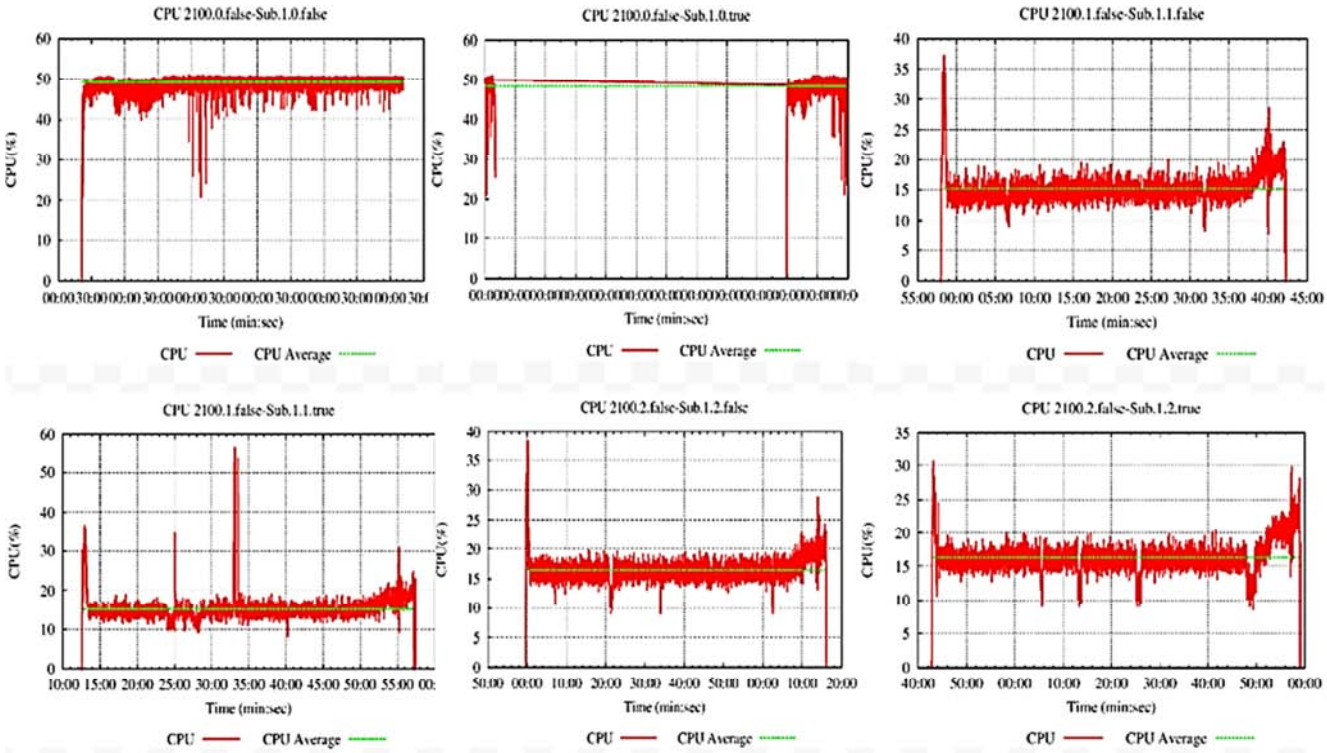

Figure 7. *CPU load of the broker in situation Multi Publishers - Single Subscriber.*

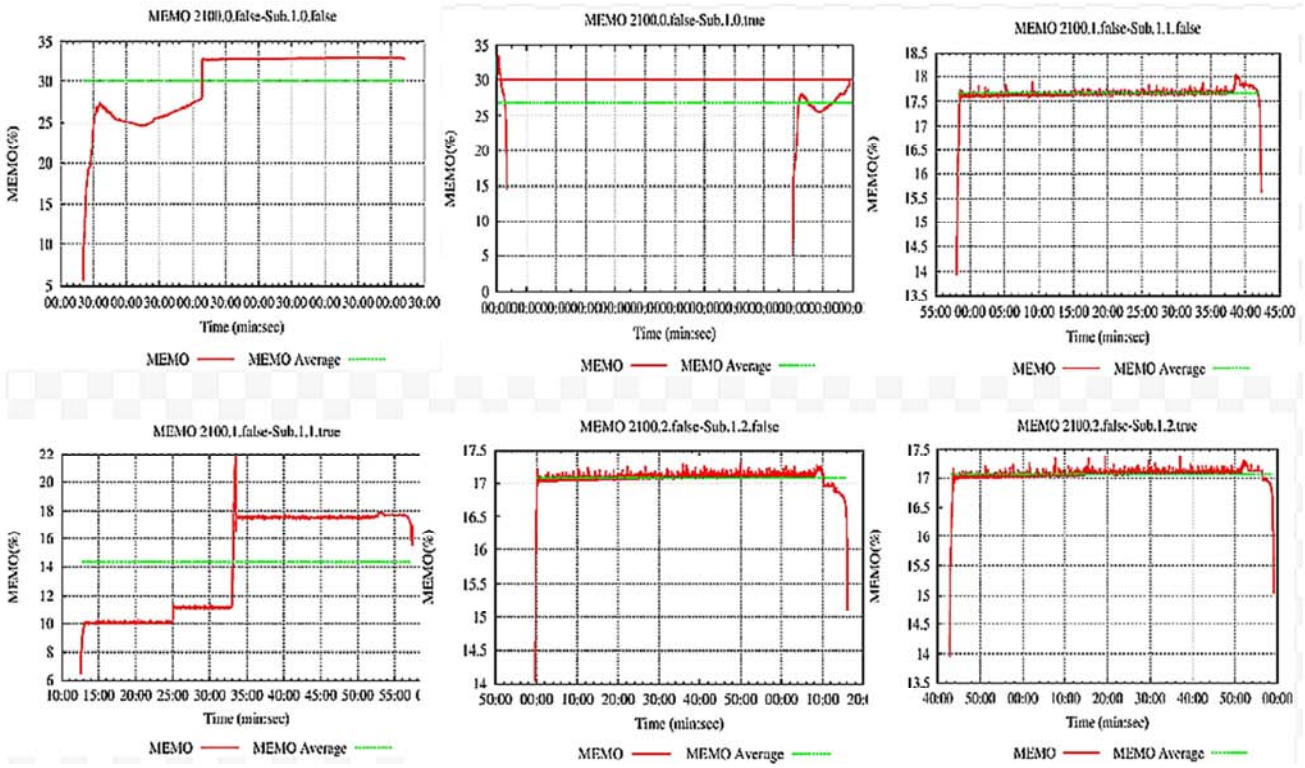

Figure 8. *Memory of the broker in situation Multi Publishers - Single Subscriber.*

Durkop has conducted experiments to check 3 IoT communication protocols - CoAP, MQTT and OPC UA - with relation to their transport mechanisms to evaluate the transmission times and analysing potentials for optimisation [15]. By employing a common middleware, a pursuit team in National University of Singapore has performed experiments to review the performance of MQTT and CoAP in terms of end-to-end delay and information measure consumption [16]. There do not seem to be several analysis firms like Ekito and Scalagent did MQTT benchmarking [17, 18]. However, all of

them used their own codes and self-developed tools to try to to those works.

There are some fashionable testing tools like JMeter, CLIF Server or Gatling. They are all designed for simple use, maintainability and high performance. Like MQTTBrokerBench, they are provided a platform which permit to appreciate load checking for a few forms of servers and permits to simulate the test of performance. Except for the message queuing protocols, all of them solely support JMS. Solely a tool for MQTT check existed on the net is SmartBear, however it is not open supply and is fixed just for some basic options of MQTT protocol [19]. To the most effective of our information, there is presently no open supply tool that measures MQTT brokers for IoT applications within the cloud atmosphere. By developing MQTTBrokerBench plug-in, we have a tendency to stuffed within the gap existing in IoT world a couple of versatile benchmarking tool for MQTT, the widely-used electronic messaging protocol. MQTTBrokerBench is developed from JMeter, therefore all JMeter users currently will expertise each JMS and MQTT with their acquainted tool. What is more, totally different analysis teams have a standard, friendly, versatile tool to guage utterly MQTT brokers. As consequence, it will be easier for these teams to check their analysis results with one another.

6. Conclusion

IoT and Cloud computing each are getting more and more fashionable and vital in the computer world. During this IT revolution, MQTT protocol is crucial as it helps to accelerate IoT technology that provides interconnected things that we have never seen. We contribute partly to this revolution by providing MQTTBrokerBench, a holistic tool to benchmark the MQTT protocol implemented on the state-of-the-art brokers. By using this tool, we simulated a use case of 42000 MQTT connections to a Mosquitto broker with only seven virtual machines on Window Azure Cloud Platform. We divided this use case into 2 scenarios with different number of publishers and subscribers. The result shows that even in the same scenario, the saturation points in various QoS adjustments are totally different. These sophisticated adjustments can be done easily using the friendly GUI of MQTTBrokerBench. Thus our proposed broker benchmark tool is completely efficient and suitable if we wish to benchmark the brokers and realize performance or stress tests in order to figure out saturation points for elasticity in which the number and the performance of brokers may be provided and adjusted as needed optimally.

Acknowledgements

This research is funded by Graduate University of Science and Technology and partly by Institute of Information Technology and National Key Laboratory of Networking and Multimedia, Vietnam Academy of Science and Technology under grant number GUST.STS.ĐT2019-TT02, CS19.19, and PTNTĐ19.02, respectively. This work also has been partly supported by Vietnam National University, Hanoi/ VNU University of Engineering and Technology under Project CN19.09.

References

[1] P. T. Eugster, P. A. Felber, R. Guerraoui and A. M. Kermarrec, The many faces of publish/subscribe, ACM Computing Survey, vol. 35, no. 2, pp. 114-131, Jun. 2003.

[2] MQTT, http://mqtt.org/, visited on June 2019.

[3] I. M. Llorente, Key challenges in Cloud Computing to Enable Future Internet of Things, Keynote speech, Jan 2012. [Online]. Available: http://fr.slideshare.net/llorente/challenges-incloud-computing-to-enable-future-internet-of-things-v03.

[4] Smarter Planet, http://www.ibm.com/smarterplanet/us/en/, visited on June 2019.

[5] Y. Zhao, K. Kim and N. Venkatasubramanian, DYNATOPS: a dynamic topic-based publish/subscribe architecture, ACM DEBS 2013, pp. 75-86.

[6] L. M. Pham and T. T. Nguyen, Flexible deployment of component-based distributed applications on the Cloud and beyond, KSII Transactions on Internet and Information Systems, 13, 3, (2019), 1141-1163.

[7] MQTTBrokerBench, https://github.com/fimocode/mqttbench, visited on June 2019.

[8] Apache JMeter, http://jmeter.apache.org/, visited on June 2019.

[9] CLIF Server, http://clif.ow2.org, visited on June 2019.

[10] Gatling, http://gatling.io, visited on June 2019.

[11] E. Cecchet, V. Udayabhanu, T. Wood and P. Shenoy, BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications, in usENIX WebApps, 2011, pp. 4-4.

[12] MQTT FuseSource, https://github.com/fusesource/mqtt-client, visited on June 2019.

[13] Sysstat, http://sebastien.godard.pagesperso-orange.fr/, visited on June 2019.

[14] Mosquitto, http://mosquitto.org/, visited on June 2019.

[15] L. Durkop, B. Czybik and J. Jasperneite, Performance evaluation of M2M protocols over cellular networks in a lab environment, ICIN, pp. 70, 75, 17-19 Feb. 2015.

[16] D. Thangavel, X. Ma, A. Valera, H. Tan and C.K.-Y. Tan, Performance evaluation of MQTT and CoAP via a common middleware, IEEE ISSNIP, pp. 1, 6, 21-24 April 2014.

[17] A. Giuliani, MQTT benchmarks with RabbitMQ & ActiveMQ, June 2014. [Online]. Available: http://www.ekito.fr/people/mqtt-benchmarks-rabbitmq-activemq/.

[18] Scalagent, Benchmark of MQTT servers, Jan 2015. [Online]. Available: http://www.scalagent.com/IMG/pdf/Benchmark MQTT servers-v1-1.pdf.

[19] SmartBear, MQTT Test Steps, [Online]. Available: https://smartbear.com/plugins/mqtt-teststeps-page/, visited on June 2019.