Deployment Architectures of MQTT Brokers in Event-Driven Industrial Internet of Things

Amirali Amiri
Institute of Computer Engineering
TU Wien, Vienna, Austria
amirali.amiri@tuwien.ac.at

Valentin Just
Institute of Computer Engineering
TU Wien, Vienna, Austria
valentin.just@tuwien.ac.at

Gernot Steindl
Institute of Computer Engineering
TU Wien, Vienna, Austria
gernot.steindl@tuwien.ac.at

Stefan Nastic
Distributed Systems Group
TU Wien, Vienna, Austria
snastic@dsg.tuwien.ac.at

Wolfgang Kastner
Institute of Computer Engineering
TU Wien, Vienna, Austria
wolfgang.kastner@tuwien.ac.at

Ian Gorton
Khoury College of Computer Sciences
Northeastern University, Seattle, USA
i.gorton@northeastern.edu

IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society | 978-1-6654-6454-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/IECON55916.2024.10905285

***Abstract*—The Industrial Internet of Things (IIoT) involves various standards, protocols, and tools, requiring extensive expertise for system design. The traditional automation pyramid limits scalability due to tightly-coupled components. Integrating IIoT data in the cloud through event-driven communication, like MQTT brokers, provides loose coupling. This integration also aids in resource monitoring and planning, enhancing IIoT application performance. Despite many IIoT architecture studies, there is a lack of empirical comparisons of MQTT broker deployment strategies. This paper examines an edge-cloud aggregation scenario, where IIoT device requests are aggregated at the edge and fog services before being transmitted to the cloud. We study four MQTT deployment architectures and compare the results empirically. We use an Arduino Opta as an IIoT device. Also, we use software-based load generation to support replicability of our experiment and reproducibility of our results. Findings indicate that a central broker on a dedicated virtual machine gives 13.8% improvements of the mean response time compared to the shared deployment of MQTT broker and device gateways. Our results offer insights into effective deployment strategies.**

***Index Terms*—Event-Driven IIoT, Empirical Data, MQTT Deployment Architectures**

I. Introduction

The Industrial Internet of Things (IIoT) includes a wide range of standards, protocols, and tools. Architects need in-depth expertise to design such comprehensive systems. The traditional automation pyramid [9] can limit scalability due to its tightly coupled and linearly-integrated components. Real-time requirements are crucial in the domain of Operational Technology (OT). However, when IIoT devices are integrated with Information Technology (IT), event-driven communication [3] can be advantageous by providing loose coupling, such as using MQTT brokers [10]. The event-driven IIoT provides many benefits, e.g., allowing easier integration of devices.

The IT/OT convergence can also help in resource monitoring and planning, improving the performance of IIoT applications. There are many studies on IIoT architectures, e.g., [5], [13], [16]. Regarding the broker-deployment architecture, different options are available, e.g., centralized, clustered, or multiple edge brokers. To the best of our knowledge, there is a lack of empirical evidence in the literature comparing different deployment architectures of MQTT brokers. This is specially important when a system is under stress with high call frequency. When multiple sites transmit information, e.g., IIoT device data, to the cloud for processing, the resource usage of brokers becomes increasingly important. Thus, we set out to answer the research questions:

RQ1: *What is the impact of MQTT deployment architectures, i.e., centralized or distributed, on the response time in an application of IT/OT convergence?*

RQ2: *How do the deployment architectures compare, when we take resource usage into account?*

The contributions are as follows: We performed multiple experiment runs using an edge-cloud aggregation scenario. In our experiment, requests from an IIoT device and a load generator were aggregated on multiple levels at the edge and fog services. The aggregated results were sent to the cloud service for logging. Results show that deploying the central broker on an exclusive Virtual Machine (VM) improves the mean response time by 13.8% compared to the shared deployment of MQTT and gateways, adding minimum additional resources.

The structure of the paper is as follows: Section II provides the related studies. Section III outlines the studied deployment architectures. Section IV presents the details of our experiment. Section V discusses the results and the threats to the validity of our research. Section VI concludes the paper.

II. Related Work

Sarasola et al. [13] argue that industrial applications often handle high-frequency data, traditionally processed directly at the source or via a commanding Programmable Logic Controller (PLC). With Industry 4.0 [7] technologies, raw data can now be transmitted in near-real time to edge, fog or cloud services for processing, requiring efficient communication protocols. The authors indicate that MQTT excels in edge and fog computing. Tan et al. [16] introduce a unified IIoT cloud platform to address interoperability issues. Their solution includes a configurable edge connector for streaming telemetry data and a multi-platform analytic dashboard, eliminating the need to develop new platforms from scratch. This platform enables secure, cloud-based sensor data access, supporting the digital transformation of factories.

Ferencz et al. [5] study the integration of IIoT devices in the cloud from an architectural point of view. They examine deployment on major cloud platforms and provide an open-source implementation of their concepts. The authors also consider security issues of IT/OT convergence. However, unlike our study, this paper does not provide empirical data to compare different architectures. Kurdi et al. [8] propose a multi-tier MQTT architecture with fog computing, deploying an authentication manager at each broker. Additionally, a lightweight mutual authentication scheme using hash functions and XOR operations is introduced. Benchmark results show this approach reduces storage and communication overheads by 89% and 23%, respectively, while enhancing resistance to cyberattacks and scalability.

Guillén et al. [6] assess the performance of an open-source IoT platform with a three-layer architecture. It includes layers for data acquisition and automation, device communication, and application management using container technology. The experiment uses MQTT for data transmission and database services. Evaluation involves virtual processes, and the key performance indicators focus on network bandwidth and container resource efficiency. Ramzey et al. [12] introduce an edge computing framework for IIoT with flowcharts and system architecture, validated through experiments showing a significant 12.14% improvement in runtime efficiency. The framework reduces costs, supports decentralization, and ensures reliability, making it suitable for automatic monitoring and control in oil field operations. Swamy et al. [15] explore the impact of IoT, addressing key design aspects like energy efficiency, scalability, and security. The authors cover IoT architectures, communication standards, computing paradigms (edge, fog, cloud), and highlight security challenges and real-time suitability. They focus on IoT in general and, unlike our study, does not provide empirical data focused on IIoT domain.

III. MQTT Deployment Architectures

We study four deployment architectures of MQTT brokers. We follow an example of an aggregation scenario from IIoT devices to the cloud. The aggregation scenario can be used, e.g., to reduce the number of requests sent to the cloud to improve resource usage. Moreover, the aggregated data can be used at the cloud for statistics, prediction models or as training data for machine-learning techniques. We study different sites with high and low frequency of incoming requests. Two high-load site are aggregated at the edge. This aggregation can be performed, e.g., to reduce the frequency of calls sent to the upper-level services, i.e., fog and cloud services. The low-load site is directly sent to the fog service to aggregate the data.

We study four architectures, in which the deployment of MQTT and gateways differ from one another. Fig. 1 shows the *central-shared* architecture. There is a central broker that

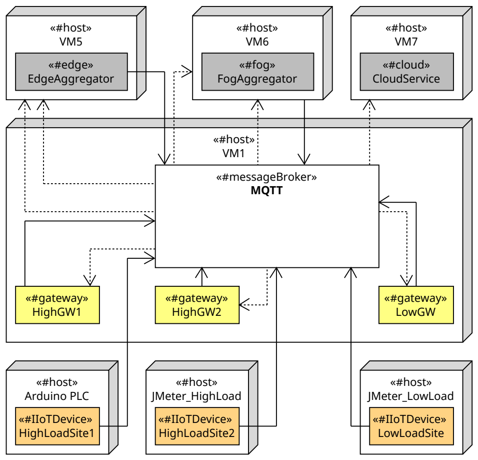


Fig. 1: *Central-Shared Architecture*
MQTT and Gateways on a Shared VM

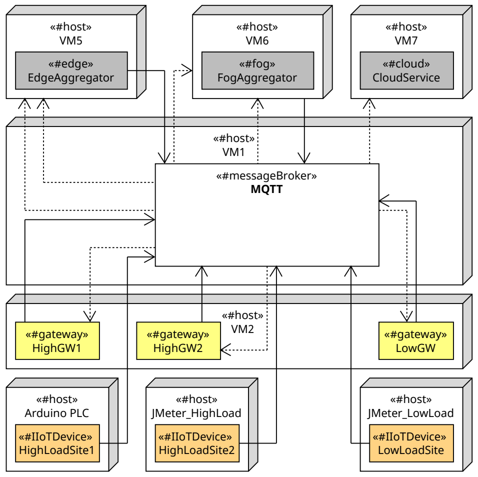


Fig. 2: *Central-SharedGW Architecture*
MQTT on one VM, Gateways on a Shared VM

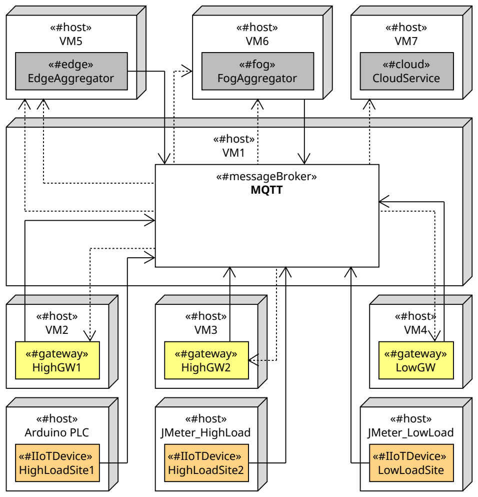


Fig. 3: *Central-Exclusive Architecture*
MQTT and Gateways on Exclusive VMs

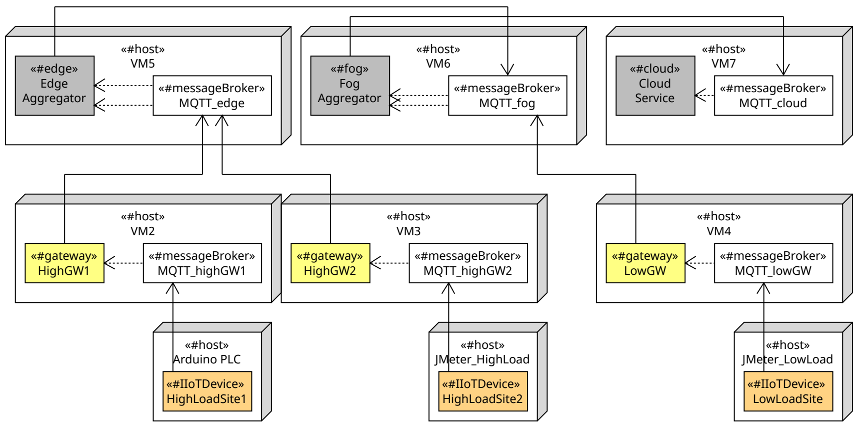


Fig. 4: *Distributed Architecture* Multiple MQTTs and Gateways on Exclusive VMs

is placed on the same VM as the device gateways. In the figure, connected and dashed arrows indicate publish and subscribe, respectively. We have removed the topic names from the figures to improve readability. In the online artifact of our study[1], we have provided the figures with topic names, through which services exchange data. The *central-sharedGW* architecture deploys the central MQTT broker on an exclusive VM but keeps the gateways on a shared one as presented by Fig. 2. The *central-exclusive* architecture, uses a central broker but deploys the gateways on exclusive VMs, given by Fig. 3. Finally, the *distributed* architecture deploys one MQTT broker on each exclusive VM for gateways, edge, fog, and cloud services, presented by Fig. 4. The details of the load generation is provided in the next section. Our empirical data indicate that clustering MQTTs is not needed in our experiment cases (see Section V: Discussion).

IV. Experiment Details

In this section, we introduce the experiment details.

1) Goals: We aim to empirically measure the impact of different MQTT deployment on the response time of request in an edge-cloud aggregation scenario. We take resource usage into consideration, and aim to find the architecture that lowers average response times with lower additional resource usage. To do so, we start with a shared architecture, where the MQTT broker and the device gateways are all on a shared VM. Each further architecture provides more resources, i.e., VMs, to avoid the overloading of the infrastructure with the high frequency of incoming load. We send requests with different frequencies to gateways, which forward the requests to edge and fog services. These services calculate an average of two incoming requests. A cloud component finally records the request response times. We compare the response time of requests in different deployment architectures of brokers.

2) Technical Details: We used 7 VMs to deploy the MQTT brokers and the services. Each VM had 2 vCPUs and 8 GB of system memory, equivalent to the Google Cloud Platform E2 machine[2]. We used VerneMQ[3] for broker implementation as it is open-source and supports clustering of brokers. The gateway, edge, fog and cloud services are implemented in Python using the Eclipse MQTT Paho library[4].

3) Load Generation: We used an Arduino Opta[5] PLC to produce MQTT version 3.1 messages. Moreover, to support replicability and reproducibility, we also generate load using Apache JMeter[6] with the MQTT plugin[7]. The software-based load generation allows researchers to reproduce our experiment, and stress the system without the need for an Arduino Opta hardware. We ran JMeter on a MacBook Air with Apple M2 chip and 24 GB of system memory.

4) Experimental Cases: For the frequency of incoming requests, we chose the maximum of 500 requests per second (r/s) and studied different portions of it. Please note that our bound of 500 r/s is very high load. Assume a home-automation scenario, where we read the temperature sensor values. In this example, 4 r/s is already very high since temperature values do not change so fast. However, we are considering a site, where many IIoT devices are sending requests to device gateways so an aggregate of 500 r/s can be imagined. Note that related studies use a higher bound of 100 r/s, e.g., [4], [14], as well as our own previous work, e.g., [1], [2]. Because in this paper, we study high loads that overload the infrastructure, we use 100 r/s as our *lowest* bound.

We also consider two low loads to be complete. These are 5 r/s and 10 r/s. We have the following eight cases:

$$\begin{aligned}(\,HighLoad,\, LowLoad\,)\ r/s \in & \\ &\{(100,5),(200,5),(400,5),(500,5)\}\\ &\{(100,10),(200,10),(400,10),(500,10)\}\end{aligned} \tag{1}$$

We consider four architecture configuration, i.e., *central-shared*, *central-sharedGW*, *central-exclusive*, and *distributed*.

[1]The code and data of our experiment is available at https://zenodo.org/uploads/13710120 doi:10.5281/zenodo.13710120

[2]https://cloud.google.com/compute/docs/general-purpose-machines
[3]https://docs.vernemq.com/
[4]https://eclipse.dev/paho/index.php
[5]https://store.arduino.cc/pages/opta
[6]https://jmeter.apache.org/
[7]https://github.com/emqx/mqtt-jmeter

Overall, we have 32 cases.

5) Data Set Preparation: For each experimental case, we instantiated the architectures and ran the experiment exactly ten minutes (excluding setup time) repeatedly to mitigate the effect of noise on the results. We report the first quartile, median, third quartile, 95th percentile, mean, and the standard deviation of the data for each case.

6) Methodological Principles of Reproducibility: We followed the eight principles of reproducibility introduced in [11]. *Repeated experiments:* We ran multiple cases, each repeated for 600 seconds. *Workload and configuration coverage:* we covered 32 experimental cases. *Experimental setup description:* see this section. *Open access artifact:* The code and the results of our experiment are anonymously downloadable in our online artifact[1]. *Probabilistic result description of measured performance:* see Section V. *Statistical evaluation:* see Section V. *Measurement units:* we reported all units. *Cost:* we did not use a public cloud settings; see this section for VM and service configurations.

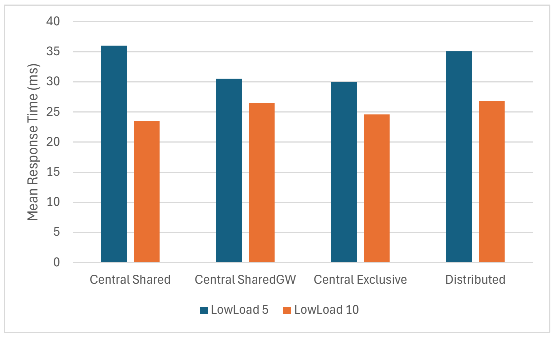
V. Results and Discussion

We report the results and discuss the main findings.

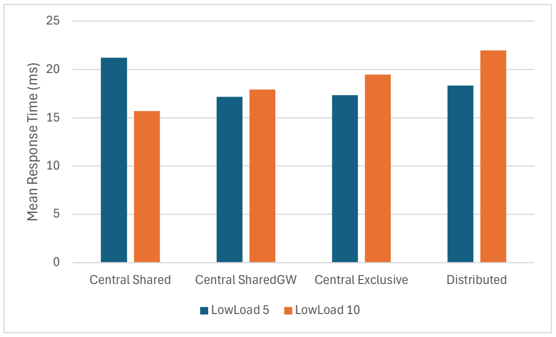
1) Results: The results of our experiment is presented in Table I and visualized in Fig. 5 w.r.t. the mean of the data. We observe that increasing the high load from 100 r/s to 200 r/s always decreases the mean response time of requests. Note that we take an average response time of two requests at the cloud, and fog services in our aggregation scenario. When a request arrives, the MQTT stores it and waits for a second request to take an average of response time. When we increase the frequency from 100 r/s to 200 r/s, the wait time for the second request to arrive decreases. This results in decreasing the mean of the data. However, this is not always the case with 400 r/s and 500 r/s. This is because the system under consideration starts to become overloaded with these frequencies, and the processing time increases.

Comparing the mean to the quartiles of the data, we see that in all cases, the mean is higher than the median. This indicates that many of the measurements appear in the upper half of the data, i.e., taking longer to be processed. This is most probably because when more requests arrive at a service, the processing times also increase, e.g., because of queuing. That is the reason, we also reported the 95th percentile of the data to inform on the few requests that take the longest in each experiment case. Note that we have not taken out any noise data and report the statistics exactly as they are recorded.

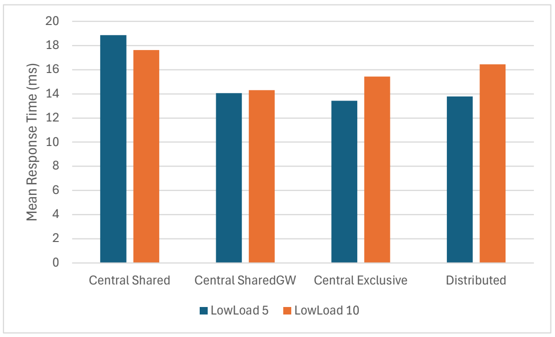
Comparing the mean response times of architectures visualized by Fig. 5, we can see that the *central-sharedGW* architecture gives comparable results to the *distributed* architecture. Remember that we put the gateways on a shared VM separated from the MQTT broker (see Fig. 2). We add only one VM to the system and already get comparable results to the more expensive *distributed* architecture. Interestingly, in all cases, the *central-exclusive* architecture gives better performance than distributed MQTT brokers. In the exclusive case, we have one VM for each gateway and the central broker



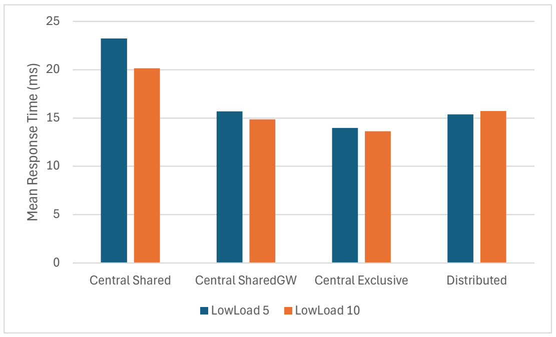
(a) High-Load Frequency of 100 r/s



(b) High-Load Frequency of 200 r/s



(c) High-Load Frequency of 400 r/s



(d) High-Load Frequency of 500 r/s

Fig. 5: Plots of all Experimental Cases Regarding the Mean of the Data

TABLE I: Results of the Experiment on Deployment Architectures of MQTT Brokers in an IIoT Application

Architecture	**High Load (r/s)**	**Low Load (r/s)**		**First Quartile (ms)**	**Median (ms)**	**Third Quartile (ms)**	**95th Percentile (ms)**	**Mean (ms)**	**Standard Deviation**
Central Shared	**100**	**5**		28.198	29.262	32.631	83.706	36.022	17.874
		10		20.334	21.513	23.686	39.453	23.538	7.477
	200	**5**		13.183	15.515	19.292	87.818	21.206	20.009
		10		11.838	12.685	14.021	37.508	15.699	8.897
	400	**5**		14.679	16.203	18.177	41.340	18.850	10.788
		10		13.354	15.565	18.541	34.283	17.624	9.053
	500	**5**		17.913	19.631	22.066	51.104	23.253	13.706
		10		16.280	17.851	20.1515	35.784	20.155	9.474
Central SharedGW	**100**	**5**		24.838	25.812	27.182	70.882	30.540	14.639
		10		22.531	23.604	25.257	51.691	26.552	9.517
	200	**5**		11.911	15.218	19.884	27.195	17.167	8.664
		10		14.624	17.383	19.033	29.734	17.929	5.845
	400	**5**		10.564	11.551	13.174	24.615	14.050	10.449
		10		11.651	12.385	13.393	29.811	14.302	7.483
	500	**5**		12.489	13.386	14.597	29.207	15.681	9.759
		10		10.943	12.191	15.822	29.815	14.863	8.143
Central Exclusive	**100**	**5**		27.599	28.738	30.028	38.338	29.965	13.456
		10		19.205	20.523	23.895	52.443	24.608	10.559
	200	**5**		12.783	15.491	19.823	26.1845	17.323	7.816
		10		12.782	16.033	20.722	41.422	19.456	10.054
	400	**5**		10.116	11.331	13.135	21.571	13.443	9.355
		10		11.584	12.684	15.143	36.430	15.440	8.514
	500	**5**		10.994	11.905	13.101	26.070	13.966	9.382
		10		10.5 00	11.365	12.975	27.213	13.616	7.801
Distributed	**100**	**5**		24.041	25.654	28.178	97.072	35.107	23.535
		10		22.462	24.647	27.568	52.513	26.789	9.881
	200	**5**		13.756	17.123	21.475	28.965	18.327	6.149
		10		15.299	17.926	23.065	44.876	21.953	11.016
	400	**5**		11.675	12.805	14.287	17.502	13.781	5.830
		10		13.192	15.145	18.855	23.095	16.455	5.216
	500	**5**		12.727	13.711	14.909	18.556	15.373	8.824
		10		12.812	14.422	15.928	22.480	15.740	6.977

(see Fig. 3). In the *distributed* architecture, we have multiple MQTT brokers, each on the same VM as gateways, edge, fog, and cloud services (see Fig. 4). We quantify the differences.

2) Answers to RQs: Let $\overline{\Delta}_{arch}$ be the average architecture differences (%), and $meanRT$ the mean response time.

$$\overline{\Delta}_{arch} = \frac{100\%}{n} \cdot \sum_{c \in Cases} \frac{meanRT_{shared} - meanRT_c}{meanRT_{shared}} \tag{2}$$

$meanRT_{shared}$ corresponds to the *central-shared* architecture. $Cases$ is the set of all frequencies, i.e., four levels of high load (100, 200, 400, and 500 r/s) and two levels of low load (5 and 10 r/s). So $n = 8$. To answer **RQ1**, we have:

$$\overline{\Delta}_{central-sharedGW} = 13.8\% \tag{3}$$

$$\overline{\Delta}_{central-exlusive} = 15.0\% \tag{4}$$

$$\overline{\Delta}_{distributed} = 6.5\% \tag{5}$$

Overall, we can infer that deploying the MQTT broker on one VM and giving resources exclusively to this broker results in lower response times, compared to the shared architecture. This can be achieved with the lowest additional cost of deploying the gateways on a separate shared VM in case of the *central-sharedGW* architecture. In case even lower response times are required, the *central-exclusive* provides even better performances. Note that in this experiment, we took adding resources into account. Clustering the MQTT brokers is also a possible approach but it is costly. As the 400 r/s and 500 r/s are very high for IIoT applications, and our data showed that the broker was not overloaded with lower frequencies, the *central-sharedGW* architecture lowers mean response times with minimum additional resources, answering **RQ2**.

3) Threats to Validity: We discuss the following threats.

Construct validity We studied deployment architectures of MQTT brokers, and measure the response time of requests in an aggregation scenario. We studied when MQTT brokers and device gateways use shared and exclusive resources. However, other measures can also improve the response time, e.g., clustering of multiple MQTT broker instances. In our experimental cases, clustering of MQTTs using the shared resources, i.e., *central-shared* and *distributed* architectures, deteriorated the measurements. These cases resulted in a longer response time. Adding another broker instance to share the underlying VM did not improve the results. For the architectures where MQTTs use an exclusive VM, i.e., *central-sharedGW* and *central-exclusive* architectures, the clustering was not needed. A central broker on an exclusive VM could handle the requests (see Section V: Discussion).

Internal validity This threat concerns factors that affect the independent variables with respect to causality. We collected empirical data to measure the impact of MQTT-deployment architectures on the response time of requests. However, we did so in limited experiment time. We avoided factors such as other load on the machines where the experiments ran. Nonetheless, more research observing real-world cloud-based systems would be needed to confirm that there are no other factors influencing the measurements.

External validity To increase internal validity, we decided not to run the experiment on a public cloud where, e.g., other load on the experiment machines might have had a significant impact on the results. As a consequence, there is the threat that generalization to a public cloud setting might be limited. Also, we covered multiple load levels and considered high- and low-load sites different for each experiment case. However, the load was still constant, i.e., not time-varying, and we did not consider bursty load. We plan to cover other load profiles for future work. A related threat is that we implemented all our services using Python and not as polyglot. We did so to have a comparable infrastructure and avoid technological impacts.

VI. Conclusions and Future Work

In this paper, we set out to answer what the impact of MQTT deployment architectures, i.e., centralized or distributed, is on the response time in an application of IT/OT convergence (**RQ1**), and how the deployment architectures compare, when we take resource usage into account (**RQ2**). We performed multiple experiment runs using an edge-cloud aggregation scenario. The data of an IIoT device and a load generator were fed to the system under test. We recorded the response time of requests with different call frequencies. Our study informs on effective deployment strategies, summarized as follows.

Comparing to the shared deployment of the broker and device gateways on a single VM (the *central-shared* architecture), our empirical data showcases that we can achieve an average of 13.8% improvements of mean response time over our experiment cases. This is given using a dedicated VM for the MQTT broker and a shared VM for device gateways (the *central-sharedGW* architecture). In case even lower response times are needed, deploying the MQTT broker and each gateway on an exclusive VM (the *central-exclusive* architecture) can give an average of 15.0% improvements compared to the shared deployment. Interestingly, deploying multiple brokers on the VM, where gateways reside (the *distributed* architecture) only improves the mean response on average 6.5% over our experiment cases. We conclude that the *central-sharedGW* architecture improves mean response times, while adding minimum additional resources.

For our future work, we plan to provide support to monitor the system at runtime, and automatically adapt the broker deployment to different load profiles. This decision is usually made using optimization techniques, e.g., multi-criteria optimization of resource usage and minimization of the response time of requests. Moreover, data density can be considered. For example, the system can automatically deploy MQTT brokers, where a high-frequency of data is being generated for a short period of time, i.e., bursty load.

References

[1] A. Amiri and U. Zdun. Tool support for the adaptation of quality of service trade-offs in service- and cloud-based dynamic routing architectures. In *17th European Conference on Software Architecture (ECSA)*, 2023.

[2] A. Amiri, U. Zdun, and A. van Hoorn. Modeling and empirical validation of reliability and performance trade-offs of dynamic routing in service- and cloud-based architectures. In *IEEE Transactions on Services Computing (TSC)*, 2021.

[3] H. Cabane and K. Farias. On the impact of event-driven architecture on performance: An exploratory study. *Future Generation Computer Systems*, 153:52–69, 2024.

[4] D. J. Dean, H. Nguyen, P. Wang, and X. Gu. Perfcompass: Toward runtime performance anomaly fault localization for infrastructure-as-a-service clouds. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

[5] K. Ferencz, J. Domokos, and L. Kovács. Cloud integration of industrial iot systems. architecture, security aspects and sample implementations. *Acta Polytechnica Hungarica*, 21(4), 2024.

[6] J. F. Guillén, J. M. Sigua, and J. C. Zambrano. Performance evaluation of an internet-of-things platform based on open-source. In J. P. Salgado-Guerrero, H. R. Vega-Carrillo, G. García-Fernández, and V. Robles-Bykbaev, editors, *Systems, Smart Technologies and Innovation for Society*, pages 281–290, Cham, 2024. Springer Nature Switzerland.

[7] R. Heidel. *Industrie 4.0: The reference architecture model RAMI 4.0 and the Industrie 4.0 component*. Beuth Verlag GmbH, 2019.

[8] H. Kurdi and V. Thayananthan. A multi-tier mqtt architecture with multiple brokers based on fog computing for securing industrial iot. *Applied Sciences*, 12(14), 2022.

[9] C. Lucizano, A. A. de Andrade, J. F. Blumetti Facó, and A. G. de Freitas. Revisiting the automation pyramid for the industry 4.0. In *2023 15th IEEE International Conference on Industry Applications (INDUSCON)*, pages 1195–1198, 2023.

[10] S. Mirampalli, R. Wankar, and S. N. Srirama. Evaluating nifi and mqtt based serverless data pipelines in fog computing environments. *Future Generation Computer Systems*, 150:341–353, 2024.

[11] A. V. Papadopoulos, L. Versluis, A. Bauer, N. Herbst, J. von Kistowski, A. Ali-Eldin, C. L. Abad, J. N. Amaral, P. Tuma, and A. Iosup. Methodological principles for reproducible performance evaluation in cloud computing. In *IEEE Transactions on Software Engineering*. IEEE, 2019.

[12] H. Ramzey, M. Badawy, M. Elhosseini, and A. A. Elbaset. I2ot-ec: A framework for smart real-time monitoring and controlling crude oil production exploiting iiot and edge computing. *Energies*, 16(4), 2023.

[13] T. F. D. B. Sarasola, A. García, and J. L. Ferrando. Iiot protocols for edge/fog and cloud computing in industrial ai: A high frequency perspective. *International Journal of Cloud Applications and Computing (IJCAC)*, 14(1):1–30, 2024.

[14] O. Sukwong, A. Sangpetch, and H. S. Kim. Sageshift: managing slas for highly consolidated cloud. In *2012 Proceedings IEEE INFOCOM*, pages 208–216, 2012.

[15] S. N. Swamy and S. R. Kota. An empirical study on system level aspects of internet of things (iot). *IEEE Access*, 8:188082–188134, 2020.

[16] S. Z. Tan and M. E. Labastida. Unified iiot cloud platform for smart factory. *Implementing Industry 4.0: The Model Factory as the Key Enabler for the Future of Manufacturing*, pages 55–78, 2021.