Self-Adaptive Intelligent Deployment of Message Brokers: An Empirical Study on IoT Performance

Amirali Amiri
Automation Systems Group
TU Wien, Vienna, Austria
amirali.amiri@tuwien.ac.at

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

Bahman Javadi
School of Computer, Data and Mathematical Sciences
Western Sydney University, Sydney, Australia
b.javadi@westernsydney.edu.au

Wolfgang Kastner
Automation Systems Group
TU Wien, Vienna, Austria
wolfgang.kastner@tuwien.ac.at

***Abstract*—The Internet of Things (IoT) encompasses diverse characteristics, such as varying load frequencies and performance requirements. Designing static IoT systems to cover such varying loads is challenging. Self-adaptive systems, enhanced by artificial intelligence, can offer better performance by responding dynamically to changing conditions. Empirical research is essential to validate such systems. In this paper, we contribute by designing experiments to assess the performance of multiple IoT architectures under various load frequencies. Using the empirical data collected, we train an artificial neural network to predict response times for untested frequencies, identifying optimal scenarios for self-adaptive architecture transitions. We present a dataset of 2,641,008 points regarding the response times of requests for several industrial IoT deployment architectures. We perform an extensive systematic evaluation of 4,374 cases indicating 29.6% improvements in terms of reducing mean response times. Additionally, we provide prototypical tool support for practical implementations and to make our approach easy-to-use.**

***Index Terms*—Self-Adaptive Systems, Deep Neural Networks, Empirical Data**

I. Introduction

The Internet of Things (IoT) has very different characteristics, e.g., load frequencies and performance. Very high incoming load is found in IoT sensor readings from the domain of Industrial IoT (IIoT). We typically face sites of many IIoT devices with aggregated incoming requests, e.g., in smart factories. Moreover, the real-time requirements play a vital role in the domain of operational technology [16]. The necessity to hold deadlines typically requires high-frequency of communication. The performance, in terms of response times of requests with different load frequencies, is an important and defining characteristics of the IoT systems.

It is hard and ineffective to design a static IoT application to cover various load frequencies. A self-adaptive system [4] and artificial intelligence can prove beneficial to serve the dynamic behavior of IoT applications to give low response times. The complex landscape of standards, device types, and IIoT architectures, e.g., [7], [18], makes it hard to design such systems without deep expertise of the field. An optimization based on the deployment of device gateways, either on exclusive or shared resources with respect to the MQTT brokers [12] is desired. The goal is to improve performance in terms of response times, when using event-driven communication [5]. Thus, we set out to answer the following research questions:

RQ1: *What are the impacts of IoT deployment architectures with several load frequencies on the response time of requests?*

RQ2: *How can intelligent IoT systems self-adapt to optimize the performance?*

The contributions of the paper are as follows. Firstly, we ran multiple experiments on a private cloud infrastructure to understand the performances of IoT architectures for multiple load profiles. Secondly, we used our empirical data to train an Artificial Neural Network (ANN) [17]. The ANN predicts the response time of requests for load frequencies that were not included in our experiment data. This prediction gives up precise load profiles where a self-adaptivity between different studied architecture proves beneficial in terms of performance. Our ANN indicates an adaptation threshold of 339 r/s.

A third contribution of the paper is a dataset of 2,641,008 points, reporting measured data of our experiment w.r.t. response times for different IIoT deployment architectures. As our fourth contribution, we propose a tool support to make the use of our concepts easier, as well as generate the optimized code. We performed an extensive systematic evaluation of 4,374 cases, and found out that our approach provides 29.6% performance improvements on average with exclusive deployment of the message broker.

The structure of the paper is as follow. Section II gives the background of our study. Section III provides the approach details. Section IV presents our supporting tool. Section V evaluates our method. Section VI studies the threats to the validity of our work. Finally, Section VII concludes the paper.

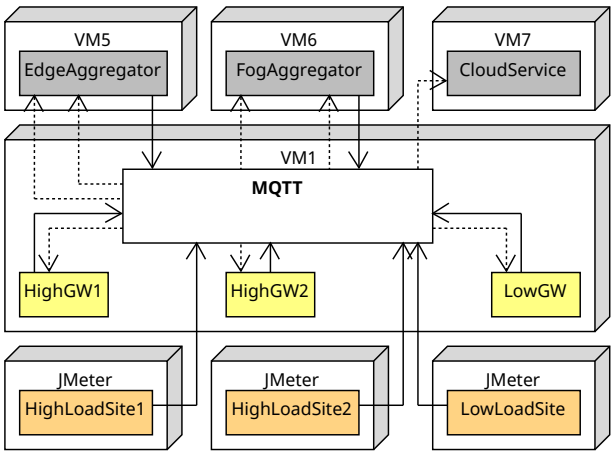


Fig. 1: The *Shared* Architecture

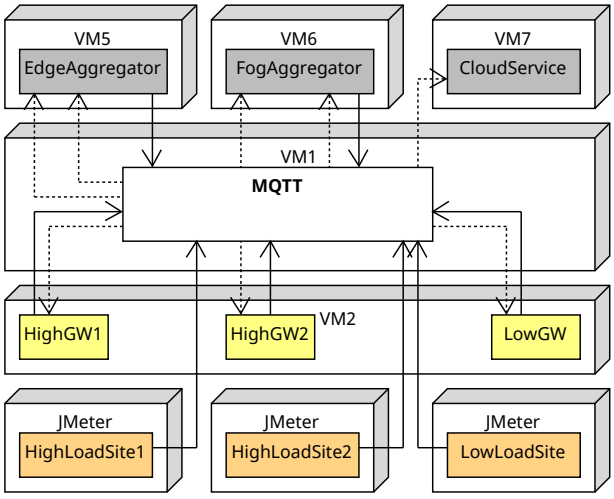


Fig. 2: The *Shared-Gateway* Architecture

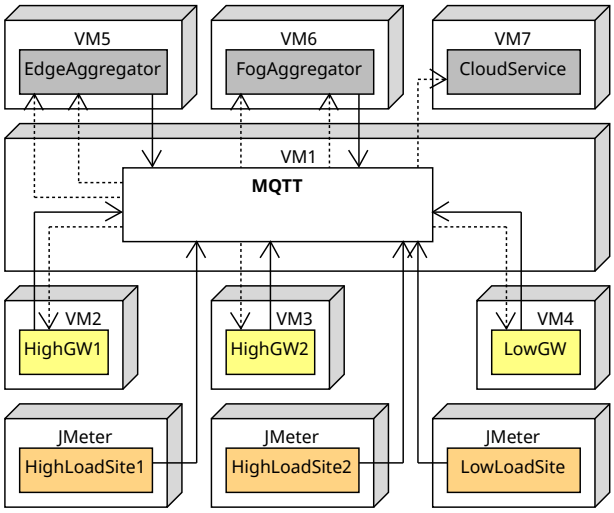


Fig. 3: The *Exclusive* Architecture

II. State of the Art

Background. In [2], we studied different deployment architectures of the event-driven IIoT systems [5]. We ran multiple experiments using an aggregation scenario with an IIoT data connected to the cloud to reduce the number of requests and optimize resource usage. Aggregated data can be used for statistics, prediction models, or machine learning. We analyzed sites with varying request frequencies: High-traffic sites were aggregated at the edge to minimize calls, while low traffic sent directly to the fog service.

In this paper, we build on top of our previous work and study three IIoT deployment architectures. In the studied architectures, the deployment of the IIoT device gateways differ from one another. In all of these system architectures, a central MQTT broker [12] processes the incoming load. Fig. 1 illustrates the *shared* architecture, where an MQTT broker is hosted on the same Virtual Machine (VM) as the device gateways. We study two more architectures: The *shared-gateway* architecture deploys the central MQTT broker on an exclusive VM but keeps the IIoT device gateways on a shared one as presented by Fig. 2. The *exclusive* architecture, uses a central broker but deploys the gateways on exclusive VMs, shown by Fig. 3. In the online artifact of our study[1], the figures with topic names are available. The code and data of our experiment are included in the online artifact (see Table I).

Related Work. Qui et al. [14] argue that robust topology is vital for IoT connectivity and longevity. They introduce ROMEM, an evolutionary, self-adaptive method that enhances exploration and speeds convergence through distributed training. ROMEM outperforms existing neural learning methods in robustness and efficiency. In contrast, our study focuses on optimizing IIoT request response times by adapting deployment architecture. Alfonso et al. [1] propose a self-adaptive, multi-layered IoT architecture with edge and fog computing to enhance QoS. Their model-based approach uses a Domain-Specific Language (DSL) for automated deployment, demonstrated in underground mining. Unlike our study, they provide no empirical performance data for intelligent IoT systems.

Jo et al. [9] propose a self-adaptive framework for cyber-physical systems, minimizing delay by adjusting sensing and analysis periods based on network and CPU workloads. While both our work and theirs use a publish-subscribe pattern, their focus is on sensing adjustments, whereas ours adapts IIoT architecture to handle load frequency. Provoost et al. [13] describe self-adaptive IoT systems that use feedback loops to maintain reliable wireless communication despite environmental challenges. They propose a joint-learning pattern for distributed adaptation, validated in two case studies, though lacking empirical evidence. Similarly, our study addresses uncertainties by adapting to varying load frequencies.

Sarasola et al. [16] emphasize MQTT's effectiveness in transmitting high-frequency data in industrial applications, showing it outperforms protocols like OPC UA and Apache Kafka. While their study compares communication protocols,

[1]https://zenodo.org/uploads/14181394 DOI:10.5281/zenodo.14181394

TABLE I: Experiment Data with the Low Load of 5 r/s for All Cases

Architecture	High Load (r/s)	First Quartile (ms)	Median (ms)	Third Quartile (ms)	95th Percentile (ms)	Mean (ms)	Standard Deviation
Shared	**25**	31.725	42.839	54.429	70.563	42.731	16.793
	50	22.817	28.654	37.778	88.012	35.182	20.356
	75	16.045	20.200	25.955	85.634	27.527	21.916
	100	16.033	19.451	23.915	35.052	22.330	14.248
	125	15.717	17.972	21.580	63.405	22.462	16.644
	150	13.246	15.592	18.733	75.532	20.829	19.041
	175	14.036	15.957	18.526	66.561	20.353	16.988
	200	12.387	14.19	16.549	36.427	16.697	10.460
	225	12.238	13.690	15.603	21.128	14.932	6.432
	250	12.594	14.116	16.300	40.757	17.929	23.412
	275	12.308	13.665	15.519	37.648	16.896	13.839
	300	13.318	14.682	16.513	37.593	17.868	13.917
	325	13.114	14.565	16.531	16.531	19.476	18.164
	350	14.893	16.909	20.813	80.293	25.525	28.658
	375	16.764	18.998	23.696	92.177	28.493	27.779
	400	191.733	594.737	977.180	1406.268	624.659	458.075
Shared-Gateway	**25**	26.441	43.317	54.801	71.731	41.075	19.583
	50	22.819	28.882	38.121	84.532	35.106	20.224
	75	19.617	24.215	30.475	62.078	27.942	13.804
	100	17.126	20.932	25.962	65.057	25.192	15.799
	125	14.957	17.916	21.685	39.596	19.905	8.596
	150	13.539	16.035	19.419	44.574	18.891	11.320
	175	12.145	14.229	16.891	23.549	15.595	6.892
	200	14.065	15.846	18.180	23.397	16.802	4.812
	225	13.506	15.014	17.057	22.080	17.489	12.636
	250	12.885	14.405	16.609	27.091	17.370	13.374
	275	12.532	13.866	15.806	26.940	16.531	11.914
	300	11.571	12.712	14.412	24.614	14.829	9.220
	325	10.620	11.808	13.581	23.726	14.109	10.667
	350	12.521	13.478	14.839	46.756	17.176	15.095
	375	12.432	13.411	14.778	38.590	16.074	11.000
	400	11.983	13.048	14.397	50.824	17.217	17.007
Exclusive	**25**	43.862	56.864	73.347	91.419	58.683	19.492
	50	26.203	33.069	42.862	42.862	41.373	24.523
	75	21.083	25.485	31.326	68.492	29.745	15.913
	100	18.196	21.734	26.285	66.504	25.493	15.838
	125	17.209	20.029	23.460	29.340	21.036	6.720
	150	17.093	19.516	22.630	64.946	23.523	15.635
	175	15.444	17.453	20.064	37.994	20.880	14.795
	200	13.226	15.075	17.595	53.874	18.923	16.066
	225	11.766	13.193	15.127	19.870	14.337	5.517
	250	11.684	13.041	14.887	18.405	13.927	5.062
	275	11.777	13.037	14.799	20.588	14.501	7.024
	300	10.808	11.959	13.575	20.835	13.489	6.895
	325	12.332	13.421	14.984	23.574	15.197	8.066
	350	11.322	12.325	13.659	26.966	14.844	11.552
	375	11.042	12.022	13.340	34.550	15.097	12.645
	400	12.092	13.131	14.444	37.538	15.766	10.222

ours presents broader empirical evaluations in IIoT. Ferencz et al. [7] explore IIoT cloud integration and security but lack empirical architecture comparisons. Kurdi et al. [10] propose a multi-tier MQTT architecture with lightweight authentication, reducing overhead. Unlike these studies, our research focuses on how IIoT gateway deployment affects response times, showing a central broker better handles high-frequency load.

Ramzey et al. [15] propose an edge-computing framework for IIoT, improving runtime efficiency by 12.14% and supporting decentralization and reliability. While both studies present empirical evidence for edge computing in IIoT, their focus is on reducing downtime, whereas ours targets improving response times. Guillén et al. [8] evaluate an open-source IoT platform with a three-layer architecture using MQTT, focusing on network bandwidth and container efficiency. Unlike our study, this paper lacks extensive IIoT-specific empirical data.

III. Approach Details

Experiment Goal. We aim to empirically measure the impact of different deployment architectures reported in Section II on the response time of requests 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 the *shared* architecture, where the MQTT broker [12] and the device gateways are all on a shared VM (see Fig. 1). The *shared-gateway* and *exclusive* architectures provide more resources, i.e., VMs, to avoid the overloading of the infrastructure with the high frequency of incoming load. We send requests with various frequencies to device gateways that forward the requests to edge and fog services. These services calculate an average of the response time of two incoming requests. A cloud component finally records the response times. We compare the response time of requests in multiple IoT deployment architectures.

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. We used VerneMQ for broker implementation as it is open-source. The gateway, edge, fog and cloud services are implemented in Python using the Eclipse MQTT Paho library. We used the Docker technology for containerization. To support replicability and reproducibility, we generate load using Apache JMeter with the MQTT plugin generating MQTT version 3.1 messages. The software-based load generation allows researchers to reproduce our experiment. We used a MacBook Air with Apple M2 chip and 24 GB of memory.

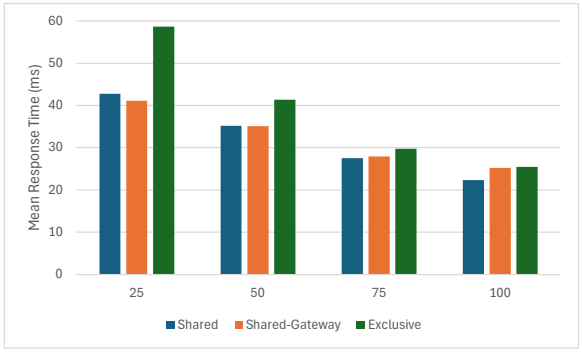
Experiment Cases. For the frequency of incoming requests, we chose the maximum of 400 requests per second (r/s) for the high load and studied different portions of it. We start with 25 r/s and increase the load in increments of 25 r/s. Therefore, we study 16 levels of incoming load. Please note that our bound of 400 r/s is very high load. There are industrial IoT scenarios, where a site of many devices send data to be processed and an aggregate of 400 r/s can be imagined. Moreover, the necessity to hold deadlines in IIoT real-time applications requires high-frequency communication [16]. Note that related studies use a higher bound of 100 r/s (see, e.g., [3]). In this paper, we study the overloading of the infrastructure so we consider higher loads.

As it can be seen in Fig. 1 to 3, we have two high-load sites and one low-load site. The studied high-load frequencies (25 to 400 r/s) are produced by both high-load sites. The low-load site sends requests with the frequency of 5 r/s to be complete. We consider three architectures, i.e., *shared*, *shared-gateway*, and *exclusive*. Overall, we have 48 cases. For each 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.

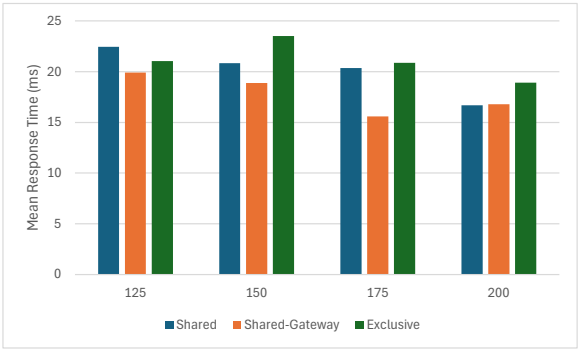
Statistics of the Empirical Data. Our empirical data are presented in Table I and visualized by Fig. 4. We observe that increasing the high load from 25 to 100 r/s always decreases the mean response time of requests for all architectures. Note that we take an average response time of two requests at 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 times. When we increase the frequency from 25 to 100 r/s, the wait time for the second request to arrive decreases. This results in decreasing the mean of the data. As shown by Fig. 4a, in these cases the more expensive *exclusive* architecture deteriorates the performance. The communication overhead between different VMs results in increasing the mean response times.

Between the loads of 100 and 200 r/s, we still see the same trend of mean time decreasing. However, we have slight increase of average response time in some cases. For example, going from 100 to 125 r/s using the *shared* architecture increases the mean response time from 22.330 to 22.462 r/s. These slight increases are most probably because of the noise in the data as we are considering the mean of all measured experiment data. However, this is not always the case with incoming load frequencies between 200 and 400 r/s. The reason is most probably because the system under consideration starts to become overloaded with these frequencies, and the processing times increase. For example, the *shared* architecture with the high frequency of 400 r/s becomes overloaded and processes the requests very slowly with the mean response time of 624.659 r/s as can be seen in Fig. 4d. That is why we fixed our high bound for the incoming load frequencies at 400 r/s. We can observe in Fig. 4c and 4d that *exclusive* architecture improves the performance compared to the other architectures with these higher load frequencies.

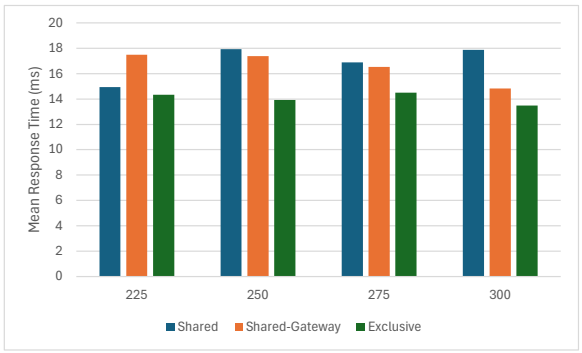
Comparing the mean to the quartiles of the data, we see that in almost all cases (except for 25 r/s using *shared* and *shared-gateway* architectures), the mean is higher than the median. This indicates that many of the measurements appear in the upper half of the data, i.e., taking longer to be processed. This is most probably because when more requests arrive at a service, the processing times also increase, e.g., because



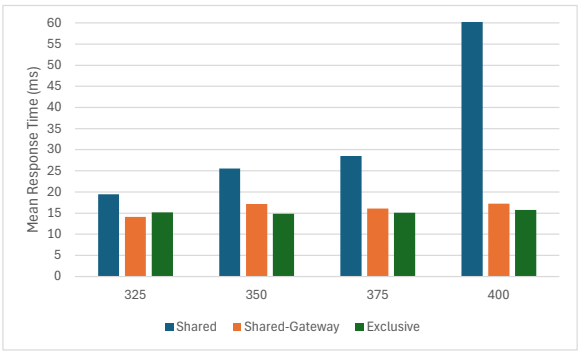
(a) High-Load Frequencies between 25 and 100 r/s



(b) High-Load Frequencies between 125 and 200 r/s



(c) High-Load Frequencies between 225 and 300 r/s



(d) High-Load Frequencies between 325 and 400 r/s (Only shown up to 60 ms)

Fig. 4: Plots of all Experimental Cases Regarding the Mean of the Data

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. We have not taken out any noise from the data and report the statistics exactly as they are recorded. We use an ANN to precisely find the boundary frequencies, with which the performance can be optimized.

Artificial Neural Network. We used the TensorFlow and the Keras sequential model to create a deep ANN [11]. A sequential model has exactly one input and one output. We prepared a Comma Separated Values (CSV) file with the high-load values and the number of added VMs compared to the *shared* architecture. The number of added VMs for the *shared*, *shared-gateway*, and *exclusive* architectures are 0, 1, and 3, respectively (see Fig. 1 to 3). For the output, we prepared another CSV file with the response time of requests for 10 minutes of experiment for each case. Our sequential model has 5 densely-connected layers, each with 40 neurons. We use the standard rectifier-linear-unit activation function, the mean squared error loss function, and the Adam optimizer [6] with a learning rate of 0.0001 for training. We used these values akin to the related works (see, e.g., [3]). One contribution is a data set with 2,641,008 points that we used for the training of the deep neural network, available in our online artifact[1].

Self-Adaptation. As mentioned before, the *shared* architecture can perform well on low incoming loads. This architecture has the lowest amount of resources in terms of VMs, which allows us to save costs. On the other hand, the *exclusive* architecture can yields benefits in terms of mean response times as statistically reported in Section III. Therefore, we can adapt the system at runtime according to the incoming load. We use the predictions of the ANN to define the exact load thresholds for this adaptation. Our ANN indicates that the *shared* architecture becomes overloaded and starts deteriorating performance starting with the frequency threshold of 339 r/s. Therefore, we make the adaptation strategy to add VM resources to the system and change the deployment architecture if the load frequency goes higher than this adaptation threshold. Conversely, when the frequency of incoming load goes below 339 r/s, we can remove resources and change to the *shared* architecture.

To choose which architecture (*shared-gateway* or *exclusive*) is a better-fitting candidate to change to with frequencies above the adaptation threshold, we consider the performance improvements and the added resources of both architectures. The *shared-gateway* architecture adds one VM resource to the system, on which the IIoT device gateways are deployed (see Fig. 2). The more expensive *exclusive* architecture adds three VM resources to the system to deploy each gateway exclusively on one VM. In cases, where the maximum performance improvement is the goal, the *exclsive* architecture

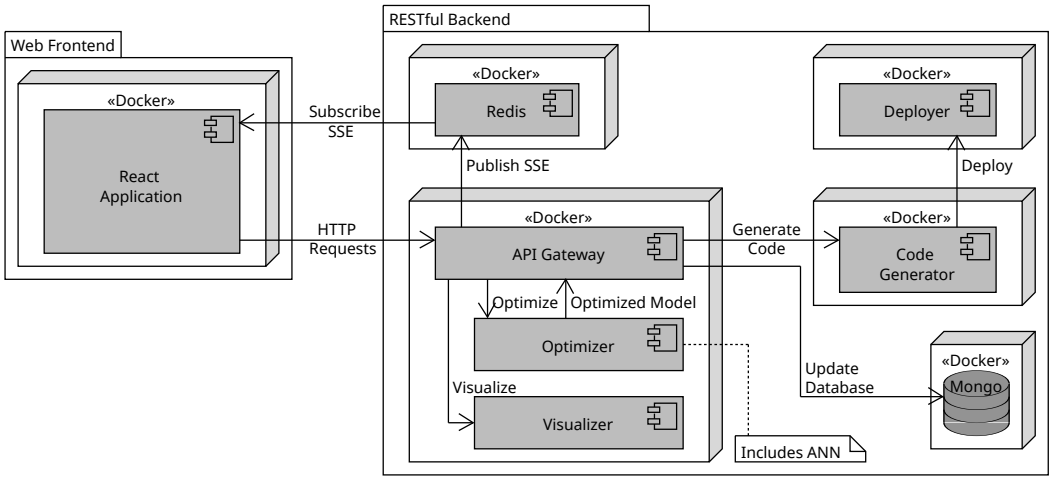


Fig. 5: Tool Architecture Overview

can be chosen. However, we argue that the *shared-gateway* architecture reduces the mean response times significantly with minimum additional resources, i.e., one (see Fig. 2).

IV. Tool Support

We provide a prototypical tool to demonstrate our approach, available in the online artifact of our study[1]. We follow a simplified syntax of Systems Modeling Language (SysML) 2.0[2] to make the learning effort of our tool minimal. Our prototypical tool supports SysML 2.0 *Parts*, *Connections* and *Metadata*[3]. Parts are any defined entity of the system, i.e., hosts, hardwares, softwares, configurations, artifacts, etc. Connections model the link between different system parts. Metadata are used to tag system parts and connections. These tags are used for artifact generation, e.g., system code.

Fig. 5 shows the architecture of our supporting tool. The frontend is implemented in React[4] and the backend is developed as a RESTful API[5] in Python. We implemented the Model-View-Controller pattern [**?**] with Server-Sent-Events (SSE) [**?**] to updated the views in the frontend. The publish-subscribe pattern [**?**] is implemented using Redis[6] for the SSE, where the Application Programming Interface (API) gateway in the backend publishes data and the frontend subscribes to different topics for rendering. This implementation ensures a realtime rendering of the database in the frontend. Visualizations are created using PlantUML[7]. The database is implemented using MongoDB[8]. We have tested the tool extensively and provided frontend and backend error handling.

The optimizer monitors the incoming data and uses our ANN predictions on load thresholds to make a decision on the adapted architecture. We follow the well-studied Monitor, Analyse, Plan, Execute, Knowledge (MAPE-K) loops [4] for the adaptation. The optimizer performs the *monitor* and *plan* phases. Having decided on the optimized model, the code generator creates the updated code and passes it to the deployer performing the *execute* phase. We log the response times and the improvement of the performance in terms of mean response time over static architectures (see Section V) accounting to the *knowledge* phase.

We use the Docker technology[9] to generate a containerized application. When an adaptation is triggered and the optimized code is generated, it takes almost three seconds until the new containers are up and running. This is important information that we consider when evaluating our approach.

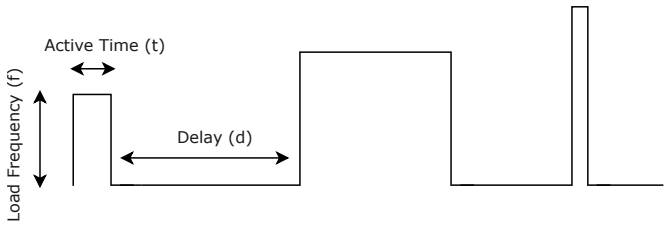
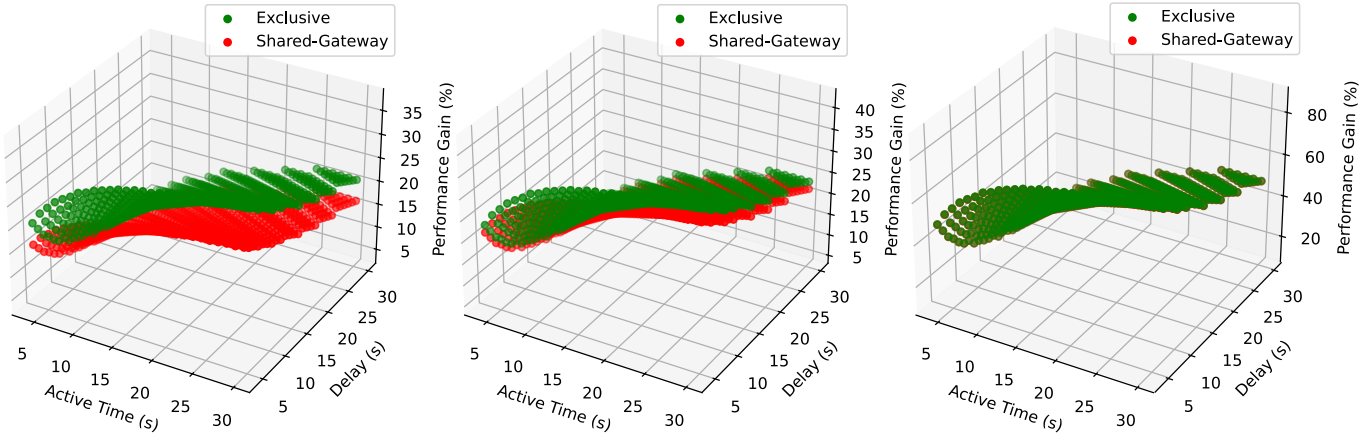


Fig. 6: Bursty Load Profile

V. Evaluation

Load Profile. There are different load profiles considered for the IIoT systems. We can characterize these loads by the frequency of incoming load, as well as the active time. A high load is active for most of the time with a high frequency. Conversely, a low load is active with a low frequency of incoming requests. Since our approach has an adaptation threshold of 339 r/s, evaluation on these loads is straightforward: We gain performance improvements during high frequencies above the adaptation threshold. However, IIoT systems are also faced with bursty load, shown by Fig. 6. We characterize this load

[2]https://www.omg.org/spec/SysML/2.0/Beta1
[3]https://github.com/Systems-Modeling/SysML-v2-Release/
[4]https://reactjs.org
[5]https://restfulapi.net
[6]https://redis.io
[7]https://plantuml.com
[8]https://www.mongodb.com
[9]https://www.docker.com



(a) High-Load Frequency of 350 r/s (b) High-Load Frequency of 375 r/s (c) High-Load Frequency of 400 r/s

Fig. 7: Performance Gains of All Evaluation Cases Compared to the *Shared* Architecture

profile by a high-load frequency f for a short active time t followed by a delay d of low load or no incoming requests. Having this characterization, our adaptation strategy is as follows: During delay d, our system self-adapts to the *shared* architecture to save on resource usage. During active time t, our system updates the deployment architecture, either to *shared-gateway* or *exclusive*. We measure the performance improvements for both architectures compared to the baseline. A bursty load is repeated with different values for f, t, and d.

Systematic Analysis. We perform a systematic evaluation on our experiment cases. As given by Table I, we have three load frequencies above our adaptation threshold of 339 r/s. These are 350, 375, and 400 r/s. We consider these values for load frequency f. When an adaptation occurs, it takes around three seconds until the new containers are up and running using the Docker technology for containerization. Therefore, we only gain performance if the active period t of the bursty load is more than three seconds. To be complete, we consider active periods of up to thirty seconds, in increments of one second. So we have 27 levels for $4 \leq t \leq 30\,s$. Finally, for the delay d, we also consider values between three and thirty seconds, in increments of one second. So we have 27 levels for $4 \leq d \leq 30\,s$. During delay, the system adapts to the *shared* architecture, so the Docker uptime must be considered.

Overall, our systematic evaluation consists of an extensive 4,374 evaluation cases. These cases are three levels for f, 27 levels for t, 27 levels for d, and we report statistics for performance improvements of two architectures, i.e., *shared-gateway* and *exclusive* over our baseline. As in our experiment, we consider each evaluation case is running for ten minutes, i.e., 600 seconds. To calculate the performance gains, we first calculate fraction of active period over 600 seconds. Let st be the seconds of active time. For example, with $t = 30\,s$, and $d = 30\,s$, we have $st = 300\,s$ of active period. Let $meanRT_{arch}$ be the mean response time, and $Gain_{arch}$ the performance gain in terms of percentage improvements:

$$Gain_{arch} = 100\% \cdot st(\frac{meanRT_{shared} - meanRT_{arch}}{600 \cdot meanRT_{shared}}) \quad (1)$$

Fig. 7 visualizes our evaluation data. We can see that in all cases, both architectures improve the performances as expected. For the case of $f = 350\ r/s$, the improvement of the *exclusive* architecture is higher than the *shared-gateway* architecture. The average percentage gains for this case are 21.4% and 16.7%, respectively. As we increase the frequency, the gap between the two architecture shrinks. For the case of $f = 375\ r/s$, we have an average gain of 24.0% and 22.3%, and for the case of $f = 400\ r/s$, 49.8% and 49.7%. On average over all cases, we have 31.7% and 29.6% for the *exclusive* and *shared-gateway* architectures, respectively.

Therefore, we can conclude that the *shared-gateway* architecture with the lowest added VM resources, i.e., one, gives on average 29.6% performance improvement. This architecture can be chosen as the best-fitting candidate for the self-adaptation with the adaptation threshold of $f = 339\ r/s$.

VI. Threats to Validity

As in all empirical research, there are a number of threats to the validity of our study. We discuss based on the four threat types by Wohlin et al. [**?**].

Construct Validity. We examined three IoT architectures and evaluated the response time of requests in an aggregation scenario. Specifically, we analyzed the performance of MQTT brokers and device gateways when utilizing either shared or exclusive resources. Although techniques like clustering multiple MQTT broker instances can improve response time, our experiments showed that clustering brokers under shared resources (the *shared* architecture) actually worsened performance, leading to longer response times. Adding another broker instance to the same VM did not improve the results. In architectures where MQTT brokers had exclusive VMs, i.e., the *shared-gateway* and *exclusive* architectures, clustering was unnecessary. A central broker on an exclusive VM could

manage the requests, with delays primarily caused by shared gateways.

Internal Validity. This threat pertains to factors that may impact the independent variables in relation to causality. We collected empirical data to evaluate the influence of IoT architectures on request response times, but the experiments were conducted within a limited timeframe. We took steps to exclude factors such as additional machine load during the experiments. However, further investigation in real-world cloud-based environments is required to confirm that no other factors affect the results.

External Validity. To enhance internal validity, we chose not to run the experiment on a public cloud, as external factors like additional machine load could have significantly influenced the outcomes. However, this presents a potential limitation in generalizing our results to public cloud environments. We plan to extend our work to public cloud settings in the future. Another related concern is that all services were implemented in Python rather than polyglot. This decision was made to maintain a consistent infrastructure and reduce the risk of technology-related influences on the results.

Conclusion Validity. As the statistical method to evaluate the accuracy of our model's predictions, we defined the performance gain based on the mean absolute percentage error [**?**] as it is widely used and offers good interpretability in our context. Additionally, we made our experiment and evaluation code, as well as data publicly available in our online artifact[1]. This is to support replicability of our experiment and reproducibility of our evaluation results.

VII. Conclusions

The performance, in terms of response times of requests with multiple load frequencies, is an important defining characteristics of the IoT systems. A self-adaptive system and artificial intelligence can prove beneficial to serve the dynamic behavior of IoT applications to give low response times. We set out to answer the research questions: What the impacts of IoT deployment architectures with several load frequencies are on the response time of requests (**RQ1**), and how intelligent IoT systems can self-adapt to optimize the performance (**RQ2**). For **RQ1**, we designed and ran multiple experiments covering different load profiles on three IoT architectures, i.e., *shared*, *shared-gateway*, and *exclusive*. For **RQ2**, we trained an artificial neural network [17] on the empirical data to find the incoming load threshold to self-adapt between architectures. We found out that the adaptation threshold for the load frequency is 339 r/s. Moreover, we provided a prototypical tool support to make our approach easy to use and reduce the complexities of designing IoT systems by generating the system code automatically.

On average, the *shared-gateway* architecture gives 29.6% performance improvements over the *shared* architecture. Given that the *shared-gateway* architecture adds the minimum amount of VMs, i.e., one (see Fig. 2), we can conclude that this architecture is the best-fitting candidate for the self-adaptation.

References

[1] I. Alfonso, K. Garcés, H. Castro, and J. Cabot. A model-based infrastructure for the specification and runtime execution of self-adaptive iot architectures. *Computing*, 105(9):1883–1906, 2023.

[2] A. Amiri, V. Just, G. Steindl, S. Nastic, W. Kastner, and I. Gorton. Deployment architectures of mqtt brokers in event-driven industrial internet of things. In *50th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2024.

[3] A. Amiri and U. Zdun. Smart and adaptive routing architecture: An internet-of-things traffic manager based on artificial neural networks. In *IEEE International Conference on Software Services Engineering*, 2023.

[4] R. Ben Halima, M. Hachicha, A. Jemal, and A. Hadj Kacem. Mape-k patterns for self-adaptation in cyber-physical systems. *The Journal of Supercomputing*, 79(5):4917–4943, 2023.

[5] 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.

[6] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. Symbolic discovery of optimization algorithms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2023.

[7] 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.

[8] 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.

[9] H.-C. Jo, H.-W. Jin, and J. Kim. Self-adaptive end-to-end resource management for real-time monitoring in cyber–physical systems. *Computer Networks*, 225:109669, 2023.

[10] 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.

[11] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. 14 - evolving deep neural networks. In R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing (Second Edition)*, pages 269–287. Academic Press, second edition edition, 2024.

[12] 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.

[13] M. Provoost, D. Weyns, D. Van Landuyt, S. Michiels, and T. Bureš. Joint learning: A pattern for reliable and efficient decision-making in self-adaptive internet of things. In *Proceedings of the 28th European Conference on Pattern Languages of Programs*, EuroPLoP '23, New York, NY, USA, 2024. Association for Computing Machinery.

[14] T. Qiu, X. Yang, N. Chen, S. Zhang, G. Min, and D. O. Wu. A self-adaptive robustness optimization method with evolutionary multi-agent for iot topology. *IEEE/ACM Transactions on Networking*, 32(2), 2024.

[15] 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.

[16] 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.

[17] V. Štuikys and R. Burbaitė. *Introduction to Artificial Neural Networks and Machine Learning*. Springer Nature Switzerland, Cham, 2024.

[18] 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.