Q-balance: An Approach for Balancing Data Imputation Tasks on Edge resources of a Smart Grid

Matheus T. M. Barbosa[1], Eric B. C. Barros[1], Vinícius F. . Mota[2], Dionisio Leite Filho [3], Damla Turgut[4], Maycon L. M. Peixoto[1]

[1]*Institute of Computing – Federal University of Bahia (UFBA) – Salvador/Brazil*
{matheus.thiago, eric.bernardes, maycon.leone}@ufba.br

[2]*Department of Computer Science – Federal University of Espírito Santo (UFES) – Vitória/Brazil*
vinicius.mota@inf.ufes.br

[3]*Faculty of Computing (Facom) – Federal University of Mato Grosso do Sul (UFMS) – Brazil*
dionisio.leite@ufms.br

[4]*Department of Computer Science – University of Central Florida (UCF) – Orlando/United States*
turgut@cs.ucf.edu

***Abstract*—Smart grids integrate intelligence, automation, and communication into the electrical grid infrastructure, primarily through the use of smart meters. These meters play a crucial role in collecting and transmitting data, either to the cloud, which may cause delays, or to the edge, where meters are closer to the data source. In this paper, we propose Q-Balance, a neural network-based solution for optimizing computational resources at the edge, thus minimizing service processing time. Q-Balance utilizes the Multi-Layer Perceptron (MLP) technique to estimate response times for requests processed by computational resources. Evaluation results demonstrate that Q-Balance can significantly reduce the average response time, achieving up to a 65% reduction compared to the Min-Load approach at the edge and up to 79% in the cloud.**

Keywords: Edge Computing, Neural Network, Smart Grid, Smart Meter;

I. Introduction

According to the US Department of Energy, global energy consumption has exhibited a consistent annual growth rate of 2.5% over the past two decades [1]. This notable escalation in energy consumption underscores the necessity of transitioning from conventional electrical grids to the Smart Grid paradigm. Smart grids seamlessly combine automation and bidirectional communication to facilitate the exchange of information concerning energy production, transmission, distribution, and consumption. By leveraging bidirectional communication, as well as employing sensors and actuators, smart grids can intelligently enhance the overall management of electrical grid resources [2] [3].

One essential device for monitoring data in a smart grid is the smart meter, used to gather real-time information such as the smart home electricity consumption. Real-time measuring has enabled significant advances in many fields, including energy disaggregation, energy consumption pattern analysis and prediction, demand response, and user segmentation [4]. Energy disaggregation, or non-intrusive load monitoring (NILM), is a technique employed to analyze and segregate aggregate electricity consumption data into specific information regarding the usage of individual appliances, eliminating the necessity for additional per-appliance measurements. NILM can be used by Home Appliances Usage Scheduling Techniques (HAUST). HAUST intelligently shifts or schedules the operation of home appliances to times when energy rates are lower or renewable production is higher, setting the demand response to avoid periods of peak consumption [5]. HAUST can also lead to reduced billing and increased efficiency of electricity consumption.

Despite the substantial contributions of smart meters to smart grid applications, they are susceptible to reliability issues, prolonged delays, and failures in data collection, leading to the occurrence of missing or erroneous data [6]. The presence of missing data poses a challenge to maintaining an accurate energy usage model, which can subsequently result in increased overall costs. Additionally, the possibility of data loss arising from transmission failures or fraudulent activities during the data transmission and measurement process emphasizes the criticality of effectively addressing this issue [7]. In these scenarios, since the reported value is not accurate, it is considered a missing value, which can generate inconsistencies, misinterpretations, and undesired results when used by HAUST.

To overcome missing data problems, data imputation techniques may replace missing data with substituted values in the pre-processing phase, keeping the main characteristics of the original values. According to Shin et al. [8] and Weber et al. [4], handling missing data represents one of the most challenging tasks during the data pre-processing phase, as erroneous values have the potential to introduce bias into the dataset. Due to the continuous data stream of smart meters, the edge computing paradigm arises as a natural solution to offload network and processing data [9]. Edge computing relies on a distributed architecture with storage and processing capabilities, closer to the end devices, such as smart meters.

However, as the number of smart meters increases, the workload generated by data imputation techniques on edge nodes also tends to rise. Inefficient allocation of resources under such circumstances can lead to increased response

times, emphasizing the significance of effective load-balancing algorithms in this infrastructure. Numerous load-balancing algorithms have been proposed in the literature for smart grids. However, to the best of our knowledge, Q-Balance is the first approach to address the simultaneous challenges of data imputation and load balancing.

Q-balance ($Queue - Balance$) adopts a learning-based resource allocation approach, which ensures the balanced utilization of heterogeneous resources by estimating the execution time for data imputation tasks. The algorithm takes into account the available resources on each server and aims to optimize the allocation of computational resources, minimizing execution time and enhancing overall system performance. Considering the continuous nature and unknown size of data gaps, Q-Balance utilizes a *MultiLayer Perceptron* (MLP) to determine the server with the lowest estimated execution time for receiving requests.

This paper contributes in three main ways. Firstly, it addresses a combined problem involving data imputation techniques and load balancing approaches. Secondly, it introduces a load balancing method using a MultiLayer Perceptron (MLP) for data imputation at edge nodes. Thirdly, it enhances edge-based approaches, leading to reliable data and quicker response times, thereby improving data imputation capabilities for NILM and HAUST applications.

II. Related Work

A. Data Imputation

Data imputation consists of filling in missing data with substitute values. Services that use the data produced by the smart meters, such as NILM and consequently HAUST, may have their results compromised due to the absence of information caused by problems in the data collection step. As a consequence, these services can produce inconsistent results due to data gaps, making the data imputation service a relevant tool for reducing inconsistencies.

Moritz et al. [10], point out different data imputation techniques and we highlight three of them, namely (i) Simple Moving Average (SMA): data imputation method by simple moving average, a technique widely used in signal processing; (ii) Seasonally Decomposed Missing Value Imputation (SEADEC): deals with data imputation through seasonal decomposition of the time series; and (iii) Singular Spectrum Analysis (SSA): attaches importance to implicit information present in the datasets to be analyzed, such as noise and trend. The recovering process of gaps present in datasets occurs through the decomposition of observed data followed by a reconstruction process, which replaces values considered to be invalid without disregarding aspects such as trends, seasonality, and noise.

B. Load Balancing in Edge Computing

Based on a three layer sensor-fog-cloud infrastructure, Ashraf et al. [11] introduce a smart grid management model and present a comparative analysis of three load balancing algorithms for scheduling electricity requests in fog servers: Round Robin (RR), Active Monitoring Virtual Machine (AMVM) and Throttled. RR, Throttled and Weighted Round Robin (WRR) were studied by Naeem et al. [12], and a three-layer infrastructure was proposed for efficient resource usage and energy management to satisfy the energy requirements of residential users. Similarly, Zahoor et al. [13] present a cloud-fog model used for resource management in a smart grid with three load balancing algorithms: Throttled, RR and PSO.

The previous work on NILM is often based on a cloud computing approach, where samples are transferred directly from the smart meter to the cloud for further analysis. This implies low sampling rates to maintain a low bandwidth, limiting the final performance obtained in identifying individual loads. Moving NILM to the edge of the network offers many advantages such as reduced operating cost and reduced power consumption while minimizing privacy concerns [14].

Tabanelli et al. [15] present hardware enabling the use of NILM at the edge of the network capable of managing data rates at high sampling frequencies. Real-time implementation of the algorithms for the processing of electrical signals and classification of NILM loads demonstrate the viability in nodes in the network edge. Yining et al. [16] propose a data interplay between cloud and edge, where experimental results show that it can effectively improve the accuracy and reliability of NILM.

As the relevant NILM problem, we emphasize that the HAUST problem is out of the scope of our work. We assume that the Q-balance's output serves as input for those problems.

III. Q-Balance

A. Architecture Overview

Q-Balance is a learning-based resource allocation approach specifically designed for edge resources, implemented as a Multi-layer Perceptron (MLP) neural network. Its primary objective is to optimize resource utilization by accurately predicting the execution time of data imputation tasks and assigning requests to the most suitable resource. In the context of a heterogeneous and dynamic environment, Q-Balance leverages MLP to fulfill the following functions: (i) continuously monitor and evaluate the availability of resources at each edge node in the cluster upon the arrival of data imputation requests; (ii) estimate the expected processing time for the requested data imputation tasks; and (iii) forward the request to the edge node with the lowest estimated response time. The system's data flow is illustrated in Figure 1.

The workflow in shown in Figure 1 is activated upon the arrival of a new data imputation request. Upon receiving the request, the data analysis service, as outlined in Algorithm 1, determines the most suitable data imputation method and subsequently forwards the request to Q-Balance for resource allocation. Q-Balance, as presented in Algorithm 2, retrieves the current CPU load information from each edge node within the cluster. By considering this information along with an analysis of the data imputation method, Q-Balance determines the optimal edge node capable of performing the data

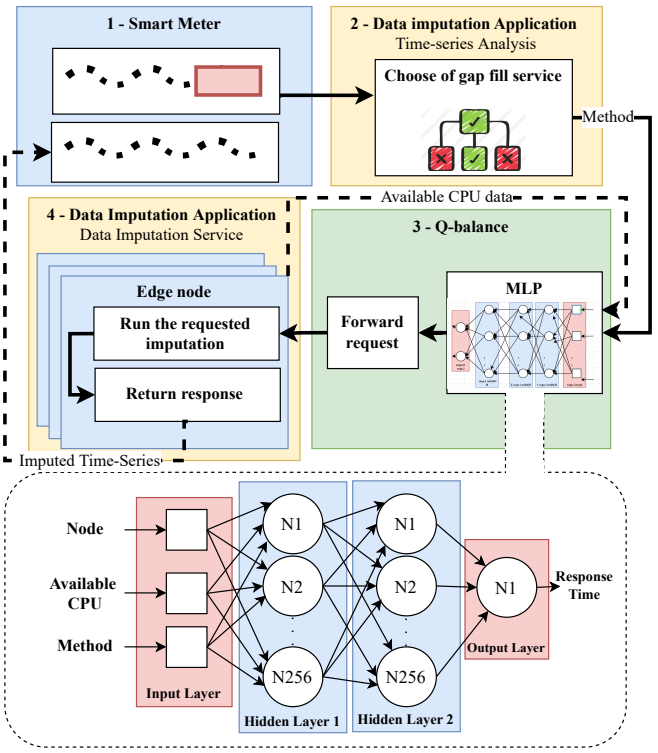


Fig. 1. Q-balance Workflow.

imputation task and forwards the request accordingly. The selected edge node then proceeds to execute the designated data imputation method, and upon completion, delivers the updated and complete time-series data back to the requester.

Algorithm 1 Data Imputation Service

Input: *request* ; // Data imputation request
Output: *result* ; // Time-series imputed
gapLength ← **checkGap**(request)
method ← **decisionTree** (gapLength)
result ← **QBalance**(method, request)
return(result)

Algorithm 2 Q-balance

Input: *method* ; // Imputation Method
Input: *request* ; // Data imputation request
Output: *result* ; // Time-series imputed
QBalance *(method, request)*:
  nodesStatus[] ← **NodesCpuLoad**()
  estimatedNodeTimes[] ← **MLP**(nodesStatus, method)
  destinationNode ← **getSmallerTime**(estimatedNodeTimes)
  result ← **destinationNode.run**(request, method)
return result

B. Data Modelling

Q-Balance takes as the input a time-series data collected by a smart meter. The data is represented as $\mathbf{T}_s(t) = \{t(1), t(2), ..., t(N)\}$, where N represents the time at which each $t(i)$ occurred. Thus, the sensing rate determines the granularity of the dataset in NILM and HAUST services.

We model energy consumption as in [17], which is defined by $T_s(n) = sw(s) + \epsilon(n) + I$. Each observation is represented as the sum of a square wave function ($sw(s)$), given by the seasonal behavior, a *lognormal* function ($\epsilon(n)$), representing the consumption variability, and a trend (I), representing a stationary process, e.g. over time the trend will converge to mean equal to zero. To simulate a data failure, a gap of size L was introduced into each time-series, resulting in a gap of size N in each, which the gap frequency is given by $\varphi = \{N - L - 1\}$. On the other hand, the data imputation task measures the accuracy of each imputation method with the Root Mean Square Error (RMSE), as $RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(Ts_i^{complete} - Ts_i^{imputed})^2}$. $RMSE$ indicates how close the imputed values are from a complete series, penalizing large errors.

Q-balance considers a set of nodes $D = \{d_1^l, d_2^l, ..., d_{|D|}^l\}$, where $l = \{e, c\}$ is the location of node ($e$dge or $c$loud), and a set of workloads $W = \{w_1, w_2, ..., w_{|W|}\}$, where $|D|$ and $|W|$ are the number of nodes and workloads, respectively. Therefore, the objective function that minimizes the processing time of data imputation tasks is given by Eq. 1, such that the sum of the workloads processing times (w) at each node (n) is the smallest for that moment (m).

$$PT_m \geq \sum_{i=1}^{|N|} \sum_{j=1}^{|WL|} n_i(wl_j) \quad (1)$$

IV. Evaluation and Performance Analysis

We conduct a thorough evaluation of Q-Balance by measuring the performance of its data imputation and load balancing algorithms. Initially, we assess the data imputation performance by comparing the effectiveness of various algorithms across different gap sizes present in the original data. Subsequently, we comprehensively evaluate the load balancing performance through extensive experimentation.

A. Data Imputation Experiments Setup

The data imputation task operates in containers, enabling the definition of the maximum amount of CPU that the service can use. Available resource on the edge node, before executing the method, is represented by the $AvailableCPU$ attribute, which is defined through usage quotas that simulate the CPU idleness.

We assume a $\mathbf{T}_s$ with $N = 1350$ observations, representing $90s$ at a sensing rate of 15Hz, enough to accommodate consumption patterns. Each observation of the series $t(n)$ was constructed considering three individual components. For the first component, we used the square wave function representing the seasonal behavior with an angular frequency equal to 15. For the second, we used a random function ϵ, defined by a *lognormal* distribution. According Kuusela et al. [17], the variability of residential energy consumption is well described by log-normal distributions, even when measured on different time or population scales. The last component is given by a stationary trend I representing the energy consumption

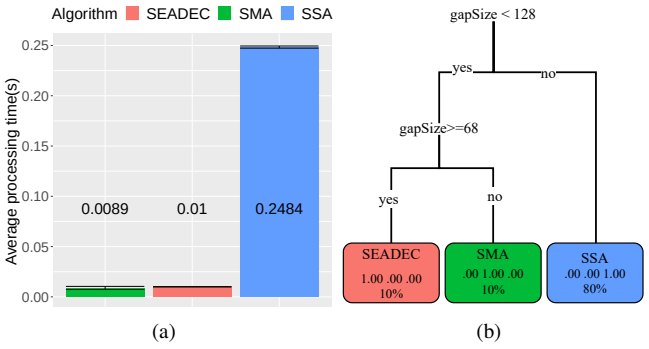
behavior. Therefore, the data adopted in the experiments can be represented by $T_s(n) = square(15) + \epsilon(n) + I$.

To compare the data imputation methods, 100 synthetic time-series were created, modifying the seed of generation of the random component ϵ. Within each time-series, a data gap of size L was introduced in order to simulate a failure of data absence. Starting from a series with a gap size $L = 15$, for each of the experiments, we increment the gap size L by 15 until $L = 600$, thus generating gap size variations of 1 to 40 seconds. The location of the gap within the series is indicated by φ, given by $\varphi = (N - L - 1)$. We observe the $mean\_time$ variable and $RMSE$ variable for each result.

B. Data Imputation Analysis

Analyzing the average processing time, Figure 2(a) shows that SSA algorithm is approximately 28 times slower than SMA algorithm and about 25 times slower than SEADEC. SSA obtains a higher average processing time due to the complex steps involved in constructing the time-series, which is helpful in achieving greater accuracy for larger gap sizes. Despite the SMA having a slower average processing time than the SEADEC, both are statistically equal due to the confidence interval of SMA mean varying between $0.007s$ and $0.0104s$, and SEADEC having an average of $0.0100s$.

Based on RMSE, we built a DT (Decision Tree), see Figure 2(b), to determine the best fitting algorithm for different gap sizes (L). For L smaller than 68, SMA algorithm achieved the lowest RMSE, meaning that it is the most suitable for this gap size range. In gaps with L between 68 and 127, SEADEC was the most desired method. For L greater than 127, DT indicates the use of SSA.



(a) (b)

Fig. 2. a) Average processing time (s). b) Decision Tree.

DT results from Figure 2(b) were used as a primary input for the construction of the data imputation service and are directly related to the load-balancing experiment shown next.

C. Load Balancing Experiments Setup

MPL in Q-balance was modeled with two dense layers of 128 and 256 neurons, respectively, using the activation function ReLU(*Rectified Linear Units*). The Adam optimizer was chosen with the parameter $\beta \in (0.9, 0.999)$. The learning rate chosen was 0.01, and the Mean Square Error (MSE) was used as a loss function. As input to the MLP, we define four environment attributes that influence the decision to forward the request to the nodes: (i) Node - an identifier used to specify the node where the requested method is executed; (ii) Method - represents the data imputation algorithm being executed; (iii) Available CPU - indicates the amount of available resources on the node prior to executing the request. To generate the training and test datasets, we employ Algorithm 3 and record the attribute (iv) Response Time, which denotes the total time elapsed from the request being sent until receiving the response. Each set of the four attributes represents different scenarios that the MLP learns to identify.

For each node and for each method, it was sent 3200 total data imputation requests. To simulate a 95% CPU load, where the load balancing is most important and ensure the load increase smoothly, we assume a quota of only $\chi = 5\%$ of the CPU for executing the method and for every 100 requests, the quota was increased by $\upsilon = 1\%$. After reaching 20%, we increment by χ per iteration until we reach 100%. Thus, total of $57,600$ scenarios were recorded for training and testing the MLP. In the training and testing stage, 85% of the scenario data were used for training the network and 15% for testing by cross-validation *ShuffleSplit* with $n\_splits = \chi$.

Algorithm 3 Dataset Generation

Output: Training and test dataset
Nodes = $[d_1^e, d_2^e, d_3^e, d_4^e, d_1^c, d_2^c]$
Methods = [SMA, SEADEC, SSA]
TS = $[\mathbf{T}s(t) = \{t(1), t(2), ..., t(100)\}]$
foreach *Node in Nodes* **do**
  foreach *Method in Methods* **do**
    $available\_CPU \leftarrow \chi$
    while $available\_CPU \leq 100\%$ **do**
      for *i in seq(0,100)* **do**
        sendRequest(No, TS[i], Method)
      end
      if $available\_CPU < 20\%$ **then**
        $available\_CPU \leftarrow available\_CPU + \upsilon$
      else
        $available\_CPU \leftarrow available\_CPU + \chi$
      end
    end
  end
end

D. Load Balancing Analysis

Initially, we carried out several experiments to verify the effectiveness of the Round-Robin (RR) and Min-Load (ML) load balancing algorithms, observing the average response time for each of the methods in all scenarios. The difference between the RR algorithm and the ML algorithm is that instead of the request distribution following a circular list, ML distributes the requests based on the load percentage of the nodes. With the load information of all nodes, ML assigns the requested request to the server with the lowest load value.

We divided the environment into two real scenarios. In the edge scenario, the requests are performed only on the available resources in the edge $(d_1^e, d_2^e, d_3^e, d_4^e)$, shown in Table I. In the cloud scenario, the requests are performed only on the available Cloud resources (d_1^c, d_2^c), shown in Table II.

To simulate high and medium workload scenarios, the average arrival interval rates of requests were set to 0.1s

TABLE I
EDGE RESOURCES

Edge resources				
Node:	d_1^e	d_2^e	d_3^e	d_4^e
Arch.:	x86_64	x86_64	x86_64	armv7l
CPU(s):	4	4	2	4
Vendor:	Intel	AMD	Intel	ARM
max GHz:	3.0	3.2	2.1	1.2

TABLE II
CLOUD RESOURCES

Cloud resources (Google)		
Node	d_1^c	d_2^c
Location:	us-central1	europe-north1
CPU(s):	8	4
Vendor:	Intel	Intel
max GHz:	3.4	3.4

and 0.2s, respectively. We use an exponential distribution to describe the behavior of the request arrival interval in each scenario. Ten replications of each set of experiments were performed with different seeds. Each replication had 1000 data imputation requests with $L = 67$ and $L = 600$ to simulate the imputation requests that use SMA and SSA, respectively. These gap size values were defined based on DT rule shown in Figure 2(b).

Figure 3 shows the average response time of ML and RR algorithms. The top part shows the result of the average response time with a high workload, in which the request arrival rate is $0.1s$. In this scenario, ML-Edge obtained the best values, considering both SMA and the SSA. At the bottom part, we see the results with the request arrival rate of $0.2s$. In this case, the load requests in the edge decrease the average response time by $1.1s$ and $1.2s$ for SMA with the load balancing ML-Edge and RR-Edge, respectively, and $2.4s$ for the SSA algorithm combined with ML-Edge.

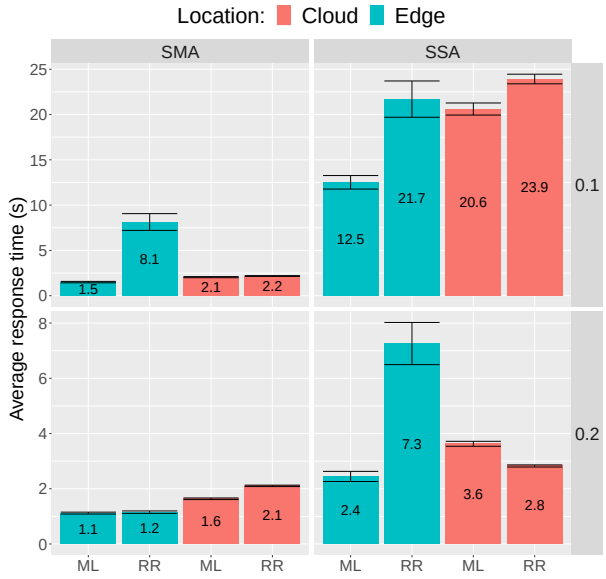


Fig. 3. Min-Load and Round-Robin, confidence level = 95%.

Due to requests dispatched by RR without any balancing control, nodes with less computational power become overloaded, generating request queues and, consequently, increasing their response times. Therefore, we applied the 2^k factorial design to evaluate the performance as described in Table III with the factors and their respective levels used.

Since ML presented lower response times, we chose it as a baseline for planning experiments using Q-balance (QB). As levels of the data imputation algorithm factor, we have chosen the SMA and SSA algorithms. To verify the influence of the request rate, we chose the request rate factor, where we defined the rates $0.2s$, for the medium workload scenario, and $0.1s$, for the high workload scenario. Finally, in the resource location factor, we used the Cloud and Edge servers detailed in Tables I and II.

TABLE III
FACTORS AND LEVELS.

Factor	Levels	
Load Balancing Algorithm (A)	Q-balance	Min-Load
Data Imputation Algorithm (B)	SMA	SSA
Request rate (C)	0.2	0.1
Resource Location (D)	Edge	Cloud

Response time was adopted as the response variable. Each experiment was replicated 10 times, where 1,000 data imputation requests were sent in each replica, resulting in 160,000 total experiments. A factorial design was used because it brings advantages such as the possible evaluation of all factors, thus being able to determine their influences/effects and interactions among factors.

Figure 4 shows the normal graph of standardized effects for the 2^k factorial design. As factors B and C move to the right of the normalized red line, the increase in the value obtained from the response time variable occurs. The most significant effects in increasing the response time appear in Data Imputation Algorithm (B) and Request rate (C). This result is due to SSA imputation algorithm having more phases in the time-series reconstruction and consuming much more CPU time than the SMA. Furthermore, it shows that the change in the rate of sending requests is the second factor that most influences the response time, since the system has a larger overhead in the rate equal to $0.1s$ in relation to $0.2s$. However, as the interaction between the AD factors, Balancing Algorithm*Resources Location, are to the left of the normalized line, we see a decrease in response time. This fact suggests that an efficient load balancing through edge can reduce the average response time for the tested scenario.

Figure 5 provides an overview of the results of the average response time in the design of experiments. In the execution of the algorithm SMA, the balancers QB-Edge and ML-Edge, being statistically equal, have the smallest response times.

For SSA data imputation algorithm with a rate equal to $0.1s$, the difference between QB-Edge and the other balancing policies is quite expressive, $4.4s$ against $12.5s$ for ML-Edge, QB-Cloud $21.5s$ and ML-Cloud $20.6s$. With the rate equal to $0.2s$ in the execution of the SSA Algorithm, the QB-Edge algorithm has the lowest average response time of $2s$, ML-Edge has the same average at $2.4s$. In cloud nodes balancing, QB-Cloud has an average of $5.7s$, and ML-Cloud has an average of $3.6s$.

In the cloud scenario, QB-Cloud underperforms due to the delay caused by the cloud's higher latency, mainly in the

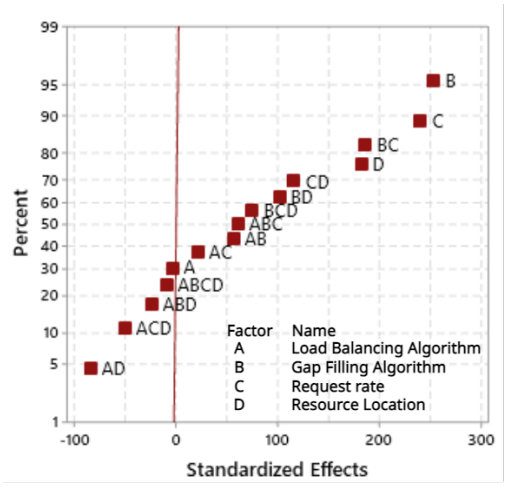


Fig. 4. Normal plot of standardized effects, confidence level = 95%.

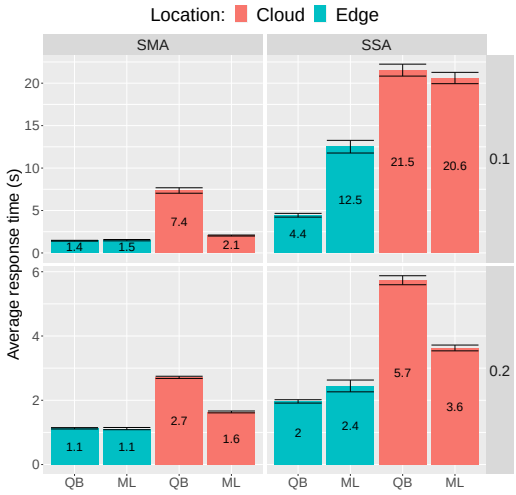


Fig. 5. Q-balance and Min-Load, confidence level = 95%.

request rate of $0.1s$. On the other hand, even when the system becomes overloaded, meaning request rate equal to 0.1 and data imputation algorithm SSA, QB-Edge appears as the best alternative for balancing requests, reaching an average of 65% smaller than ML-Edge and 79% smaller than ML-Cloud. In relation to ML, due to its characteristic of always picking the server with the lowest percentage of load, its results regarding the average response time in environments with less workload are statistically equal to QB-Edge.

V. Conclusion

The proliferation of numerous services, particularly those specific to smart grids such as data imputation for smart meters, running at the network edge with diverse characteristics presents a challenging load balancing problem. In this paper, we propose Q-Balance, an intelligent load balancing scheme that operates effectively in a dynamic and heterogeneous edge environment using a MultiLayer Perceptron (MLP). Q-Balance accurately predicts the optimal edge node for processing a set of data imputation tasks within a reduced timeframe. Our algorithm surpasses existing techniques, considered as baselines in the literature, even when confronted with various workload scenarios, both at the edge and in the cloud. Consequently, Q-Balance demonstrates its efficacy in efficiently operating within a heterogeneous and dynamic edge architecture.

Acknowledgment

This work was supported by the Brazilian research agencies: CAPES (Finance Code 001), CNPq and FAPES.

References

[1] A. Kumari, S. Tanwar, S. Tyagi, N. Kumar, M. S. Obaidat, and J. J. Rodrigues, "Fog computing for smart grid systems in the 5g environment: Challenges and solutions," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 47–53, 2019.

[2] G. Dileep, "A survey on smart grid technologies and applications," *Renewable Energy*, vol. 146, pp. 2589–2625, 2020.

[3] A. de O. Paula, R. I. Meneguette, F. T. Giuntini, M. L. Peixoto, V. P. Gonçalves, and G. P. Rocha Filho, "Strayer: A smart grid adapted automation architecture against cyberattacks," *Journal of Information Security and Applications*, vol. 67, p. 103195, 2022.

[4] M. Weber, M. Turowski, H. K. Çakmak, R. Mikut, U. Kuhnapfel, and V. Hagenmeyer, "Data-driven copy-paste imputation for energy time series," *IEEE Transactions on Smart Grid*, vol. 12, no. 6, pp. 5409–5419, 2021.

[5] E. B. C. Barros, D. M. L. Filho, B. G. Batista, B. T. Kuehne, and M. L. M. Peixoto, "Fog computing model to orchestrate the consumption and production of energy in microgrids," *Sensors*, vol. 19, no. 11, 2019.

[6] C. Klemenjak, C. Kovatsch, M. Herold, and W. Elmenreich, "A synthetic energy dataset for non-intrusive load monitoring in households," *Scientific Data*, vol. 7, no. 1, Apr. 2020.

[7] T. Cemgil, B. Kurutmaz, A. Cezayirli, E. Bingol, and S. Sener, "Interpolation and fraud detection on data collected by automatic meter reading," in *International Istanbul Smart Grid and Cities Congress and Fair (ICSG)*, April 2017, pp. 51–55.

[8] C. Shin, E. Lee, J. Han, J. Yim, W. Rhee, and H. Lee, "The ENERTALK dataset, 15 Hz electricity consumption data from 22 houses in Korea," *Scientific Data*, vol. 6, no. 1, Oct. 2019.

[9] M. L. M. Peixoto, T. A. L. Genez, and L. F. Bittencourt, "Hierarchical scheduling mechanisms in multi-level fog computing," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2824–2837, 2022.

[10] S. Moritz, A. Sardá, T. Bartz-Beielstein, M. Zaefferer, and J. Stork, "Comparison of different methods for univariate time series imputation in R," 2015.

[11] M. H. Ashraf, N. Javaid, S. H. Abbasi, M. Rehman, M. U. Sharif, and F. Saeed, "Smart grid management using cloud and fog computing," in *Advances in Network-Based Information Systems*, L. Barolli, N. Kryvinska, T. Enokido, and M. Takizawa, Eds. Cham: Springer International Publishing, 2019, pp. 624–636.

[12] M. Naeem, N. Javaid, M. Zahid, A. Abbas, S. Rasheed, and S. Rehman, "Cloud and fog based smart grid environment for efficient energy management," in *Advances in Intelligent Networking and Collaborative Systems*. Cham: Springer International Publishing, 2019, pp. 514–525.

[13] S. Zahoor, N. Javaid, A. Khan, B. Ruqia, F. J. Muhammad, and M. Zahid, "A cloud-fog-based smart grid model for efficient resource utilization," in *International Wireless Communications & Mobile Computing Conference (IWCMC)*, June 2018.

[14] H. Wöhrl and D. Brunelli, "Non-intrusive load monitoring on the edge of the network: A smart measurement node," in *Lecture Notes in Electrical Engineering*. Springer International Publishing, 2020, pp. 477–482.

[15] E. Tabanelli, D. Brunelli, and L. Benini, "A feature reduction strategy for enabling lightweight non-intrusive load monitoring on edge devices," in *International Symposium on Industrial Electronics (ISIE)*, June 2020.

[16] G. Yining, L. Xuesong, T. Donghui, and Z. Jizan, "Non-invasive power load monitoring method based on cloud edge collaboration," *IOP Conference Series: Earth and Environmental Science*, vol. 512, no. 1, p. 012115, June 2020.

[17] P. Kuusela, I. Norros, R. Weiss, and T. Sorasalmi, "Practical lognormal framework for household energy consumption modeling," *Energy and Buildings*, vol. 108, pp. 223–235, Dec. 2015.