A scalable edge computing platform for industrial applications

Wanderson Ralph Silva Vita, * **Celso José Munaro** **
Universidade Federal do Espirito Santo, Vitória-ES, BR

* *e-mail: ralph.vita@gmail.com*
** *e-mail: celso.munaro@ufes.br*

Abstract: This work presents a platform based on edge computing for the implementation of monitoring algorithms in complex industrial processes, combining safety, performance and scalability. Only free software was used and the connection of the industrial computer with processes was made via OPC UA communication standard, very common in industry 4.0. Two case studies were worked out to explore the features of the platform. First, multivariate process monitoring was applied to a pilot plant, showing steps towards training and remote monitoring. Secondly, increasing volume of synthetic data was provided by an OPC server to evaluate three performance metrics of a single node and to decide about inclusion of new nodes to share computational burden. The results achieved show the proposed environment can handle computational tasks of increasing complexity for general applications in the industry.

Keywords: Edge computing, process monitoring, IoT, Distributed computing, Industry 4.0.

1. INTRODUCTION

The monitoring of industrial processes has been gaining more and more strength with the great availability of data (Tan et al., 2020). The use of machine learning algorithms (C., 2013) brought an immense variety of algorithms that can be used for diagnostics and prognosis of equipment and processes. To afford it, growing volumes of data must be stored and processed.

Cloud computing quite expanded the reach of application usage and has emerged as a de-facto method to reduce cost with highly scalable computing services to its users (Srivastava and Khan, 2018). The possibility of maintaining algorithms based on artificial intelligence in the cloud processing the collected data, makes the architecture very attractive, due to the ease of incorporating improvements and access by a large number of customers (Labati et al., 2020).

However, cloud computing faces some challenges, such as data transmission latency, critical data security, network consumption and even the high availability of services (Ding et al., 2020).

Edge computing is a new cloud paradigm which aims to bring existing cloud services and utilities near end users (Mohan, 2019). Thus, the issues related to latency and data security are mitigated.

OPC UA is an industrial standard that brings information integration and interoperability across separated devices and applications. It was used in Beňo et al. (2019) to transform binary data from OPC UA server to JSON Publisher/Subscriber format, in order to be easily accessed from any location in the cloud. This standard is used here as well to connect the proposed platform to industrial processes. Also, an industrial computer based in the same hardware, a Raspberry Pi 3, is the main part of the proposed platform.

The challenge of this work is to propose an environment that solves these problems, and maintains the processing capacity.

2. PROPOSED ENVIRONMENT

A brief description of the resources of the proposed framework for edge computing is presented in this section. Among many existing possibilities for edge computing platforms and protocols (Mohan, 2019), the selection considered mainly open-source environments, well known and flexible hardware and industrial communication standards.

2.1 Hardware

Industrial Computer - IC. The industrial computer, is equipped with a Raspberry PI 3+ computing module as a controller,similarly to (Beňo et al., 2019), with a 1.2 GHz processor and a maximum memory of 40 GB. It also has a Raspbian operating system that is a Linux distribution, and a pre-installed Node-RED environment. Some of the benefits of this hardware are the low cost, a huge processing power in a compact board. Standard communications protocols such as UART, SPI and I^2C allow IC to communicate and exchange data with other devices.

2.2 Softwares

Python is an open source language, with powerful mathematical libraries. In this paper, `Numpy` was used for matrix calculations for multivariate statistical monitoring, `Scipy` is a library of statistical functions and was used to calculate statistical thresholds, `Scikit-learn` contains a wide

range of supervised and unsupervised machine learning algorithms, and was used to compute the principal components. These libraries make available existing algorithms from literature that can be readily implemented.

Nodered is a visual programming tool, in an open source environment, developed by IBM Emerging Technology to connect IoT devices IBM (2013). It allows running Javascript code in native form and several other languages as well, just adding libraries. Its installation on Raspberry Pi 3 is very simple. Nodered is used here to establish the communication between OPC UA and the message broker RabittMQ. It allows also user commands to perform model training.

RabittMQ is a popular open source message broker which uses the AMQP (Advanced Message Queuing Protocol) protocol to enable asynchronous communication between devices. It receives all messages from the clients and then routes the messages to the appropriate destination clients. Both Nodered and Python algorithms are producers and consumers of messages for RabittMQ while influxDB only consumes messages (See Figure 1). The origin of all processed data is OPC UA. The performance of RabittMQ in a IoT framework was evaluated in Vandikas and Tsiatsis (2014).

InfluxDB is a time series database designed to handle high recording and query loads, developed by InfluxDataInfluxData (2020). It is used here to store data generated by Python algorithms while performing process monitoring. The monitored variables are stored on process historians in general, but they can also be stored in InfluxBD. The models used for process monitoring are also saved in this database. Finally, InfluxDB provides data for visualization using Grafana.

Grafana is an open source web platform for creating interactive graphics visualization dashboards, developed by Grafana Labs. Here it will be integrated with InfluxDB in order to monitor the process variables.

2.3 OPC UA Standard

In this work, the OPC UA standard was used, which is standardized in IEC 62541, and defined as a platform independent standard. OPC UA meets all requirements for information and communication level of Reference Architecture Model Industry and is very common in the industry. OPC clients are easily configured on Nodered.

2.4 The proposed Framework

The selected hardware, software and standards and their interconnections can be seen in Figure 1. An OPC server provides data from the pilot plant to the Nodered OPC client. This data is published by RabbitMQ and consumed by Python algorithms to perform process monitoring. The output of these algorithms is again published by RabbitMQ and consumed by InfluxDB for storage and Grafana for visualization. Thus, the so called Edge Node 0 is responsible for collecting and publication of data and also for visualization of the results from monitoring algorithms that can run in this node. However, the great advantage of this framework is the possibility of expanding the processing power using additional nodes with similar hardware. In this case, new monitoring algorithms consume data from RabittMQ and return their outputs to the same broker, to be visualized by Grafana. Additional processing or storage requirements can be accomplished in the cloud.

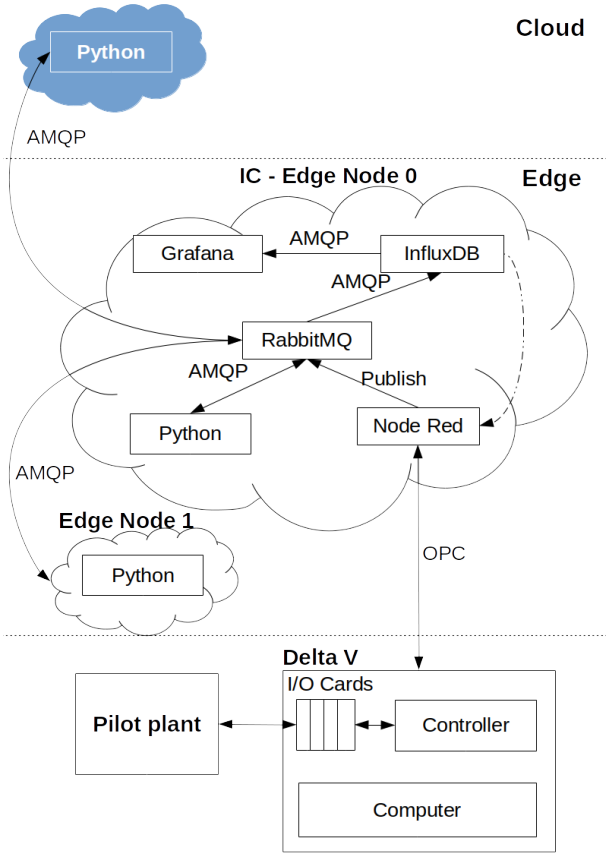

Fig. 1. Proposed Framework.

3. APPLICATIONS

Two applications are now considered to explore the main features of the proposed framework. First, a typical multivariate process monitoring algorithm is applied to a pilot plant. An industrial controller controls the plant and send data to the IC. After training, the model is used for fault detection and results are available for remote monitoring via any browser. This example highligths the ease to develope such applications to industrial processes connected via OPC UA standard, but uses negligible resources from de IC. A second application uses synthetic data provided by an OPC server to demand more IC resources. The volume of data to be processed is increased continuously to evaluate the necessity of more resources.

3.1 Application to a pilot plant

This application consists of a multivariate statistical monitoring of processes on the pilot plant described in the section 3.1.1. A leak at the pump outlet was introduced as a failure in the plant, which is emulated by opening a a_F manual valve (Figure 2) that returns the pump's outlet fluid to the reservoir.

Pilot plant. The pilot plant used in the tests consists of two coupled cylindrical tanks, 250 mm in diameter and 750 mm high, inspired in Johansson et al. (1999). The upper tank receives water pumped from a reservoir through a pneumatic control valve. The lower tank receives water from the same pump through a second pneumatic control valve and also from a manual valve at the bottom of the upper tank, that behaves like a disturbance. Both levels are controlled by the corresponding pneumatic control valves via PID controller using the levels of the reservoirs measured by pressure transmitters. The valve positions a_1 and a_2 are available via Hart protocol.The pump flow v is measured by a vortex flow meter.

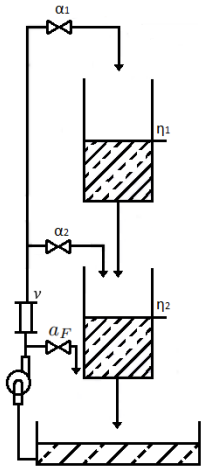

Fig. 2. Pilot plant diagram.

In order to test the fault detection algorithms, the manual valve a_f was introduced in the plant. Its opening causes the fluid to return from the pump outlet to the reservoir, emulating the loss of pump efficiency.

DeltaV DeltaV MQ Controller (Figure 1) is a powerful and flexible distributed control system (DCS) produced by Emerson. It is used here to configure the hardware, to design the control algorithms, to monitor the pilot plant and to make process variables available through an OPC-UA server. DeltaV receives from the I/O cards the measurements of flow, level and valve openings via Hart protocol. These variables and also those generated by control algorithms are made available to OPC UA clients.

Multivariate statistical monitoring In this work, Hotteling's T^2 statistic was used with order reduction of variables via principal component analysis. Consider the data matrix $\mathbf{X} = [x_1 x_2 ... x_m] \in R^{n \times m}$ with m variables and n samples. It is assumed that each variable is normalized so that it has zero mean and unit variance. Since $\mathbf{S}$ is the covariance matrix of $\mathbf{X}$ it can be approximated by the sample covariance matrix, as follows.

$$\mathbf{S} \cong \frac{1}{n-1}\mathbf{X}^T\mathbf{X} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^T + \widetilde{\mathbf{P}}\widetilde{\mathbf{\Lambda}}\widetilde{\mathbf{P}}^T \quad (1)$$

For the principal component analysis (PCA), the eigenvalue decomposition of the covariance matrix is performed to obtain the loading matrices, principal components $\mathbf{P} \in R^{n \times \ell}$ and residual matrix $\widetilde{\mathbf{P}} \in R^{n \times (n-\ell)}$, where ℓ is the number of principal components retained. The diagonal matrix $\mathbf{\Lambda}$ contains the main eigenvalues ordered in descending magnitude, while $\widetilde{\mathbf{\Lambda}}$ contains the eigenvalues of smallest magnitude.
The Hotelling T^2 statistic is calculated by

$$T^2 = \mathbf{t}^T\mathbf{\Lambda}^{-1}\mathbf{t} = \mathbf{x}^T\mathbf{D}\mathbf{x} \quad (2)$$

where $\mathbf{D} \equiv \mathbf{P}\mathbf{\Lambda}\mathbf{P}^T$ and $\mathbf{t}$ are columns of the score matrix obtained from $\mathbf{T} = \mathbf{X}\mathbf{P}$. The statistical control threshold T^2_{UCL} it is given by MacGregor and Kourti (1995)

$$T^2\text{UCL} = \frac{(n-1)(n+1)k}{n(n-k)}F_\alpha(k, n-k) \quad (3)$$

where $F_\alpha(k, n-k)$ is the upper limit for the confidence level α of the F distribution with degrees of freedom k and $n-k$.
Model training for monitoring involves decomposition to obtain eigenvalues and calculation of matrix $\mathbf{P}$. This step is performed at the workstation using a previously collected training data set. For monitoring, each new sample of variables must perform the operations: subtraction of the mean, division by standard deviation and calculation of the T^2 statistic using equation 2. These operations are performed in the IC, which receives the data sampled via OPC communication .

Environment development. The RabbitMQ was used as a messenger server to build a distributed computing environment. Three queues were created. They are described below and illustrated in Figure 3:

- **Variables**: Queue of variables read from the OPC UA server by Node-RED and sent to be consumed and processed by the Python algorithm at the processing node.
- **Monitoring**: Python processing results queue, which will be consumed by InfluxDB.
- **To model**: When the user needs to generate a new model for monitoring, Node-RED will send the range of selected variables to this queue, which will be consumed automatically by all processing nodes, to generate a new model.

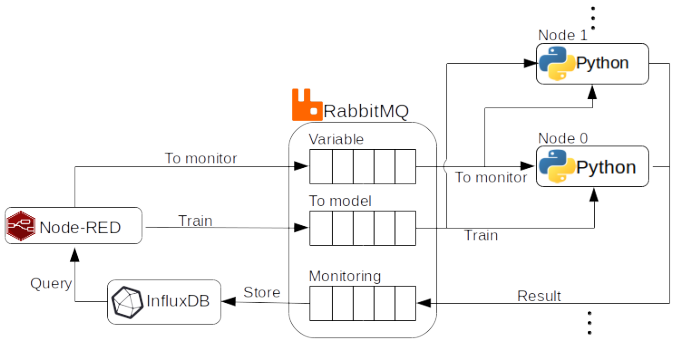

Fig. 3. Diagram of distributed computing implemented.

For communication via standard OPC by Node-RED, the installation of the library `node-red-contrib-opcua` is required. As seen in figure 4, when connecting to the OPC UA server, using the `OpcUa Client` node, where the server address is assigned, the `OpcUa Item` node is used to specify which server variable will be read, indicating the full path and type.

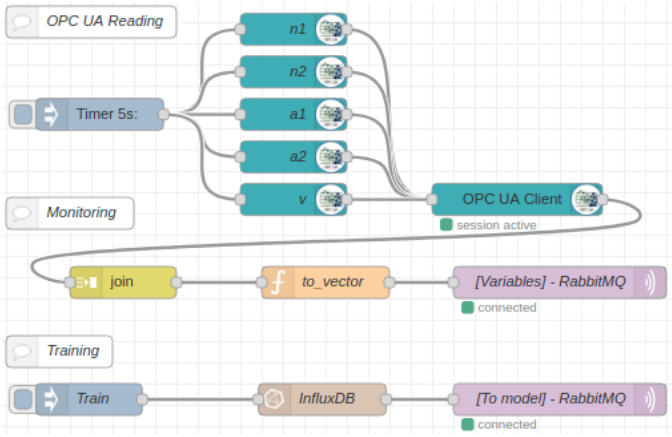

Fig. 4. Flow of reading and monitoring the variables.

The Scikit-learn library implements the PCA algorithm, which receives the normalized $\mathbf{X}$ input matrix and the desired number of components ℓ, providing the principal components $\mathbf{P}$, in the attribute `pca.components_`, and the corresponding singular values $\mathbf{\Lambda}$ in `pca.singular_values_`. Thus, Equation 2 can be implemented as seen in line 6 of Algorithm 1.

Algorithm 1 Training

1: **procedure** TRAINING($\mathbf{X}$)
2: $pca = PCA(n\_components = \ell)$
3: $pca.fit\_transform(\mathbf{X})$
4: $\mathbf{P} = pca.components\_$
5: $\mathbf{\Lambda} = pca.singular\_values\_$
6: $\mathbf{S} = \mathbf{P}.dot(\mathbf{\Lambda}).dot(\mathbf{P}.T)$
7: **return** $\mathbf{S}$

The monitoring algorithm 2 receives the new data vector $\mathbf{t}$, normalizes it with the average and standard deviation obtained from $\mathbf{X}$, and applies the T^2 statistic as in Equation 2.

Algorithm 2 Hotelling T^2 statistic

1: **procedure** HOTELLING_T2($\mathbf{t}, \mathbf{D}$)
2: **return** $T^2 = \mathbf{t}.dot(\mathbf{D}).dot(\mathbf{t}.T)$

The control threshold, calculated by the Equation 3, is implemented by the scipy library, and can be used by calling the command: $chi2.ppf(\alpha, \ell)$.

Training In the interval between 16:28 to 16:45 (figure 5), the plant is in steady state at the point of operation described in table 1.

Table 1. Monitored variables.

ID	Description	Value
n_1	Level of tank 1	50cm
n_2	Level of tank 2	50cm
a_1	Opening of valve 1	30%
a_2	Opening of valve 2	70%
v	Flow rate	47 L/min

At 16:45 the 'Train' button (figure 4), was activated and the last 200 samples of the 5 variables were collected to generate the model, with $\ell = 3$ main components being retained, keeping 95% of the variance. The sampling rate was chosen 5s to assure that subsequent samples are statistically independent.

Monitoring The plant was monitored in the range of 16:45 to 16:53, totalizing 100 samples. In the real-time graph plotted by Grafana (figure 5), one can see that from the start of monitoring to point A, the plant was in normal operation, having T^2 statistic crossed the threshold only once. The fault was introduced at point A, opening the a_F manual valve, and at point B the monitoring T^2 statistic exceeded the threshold and remained above it.

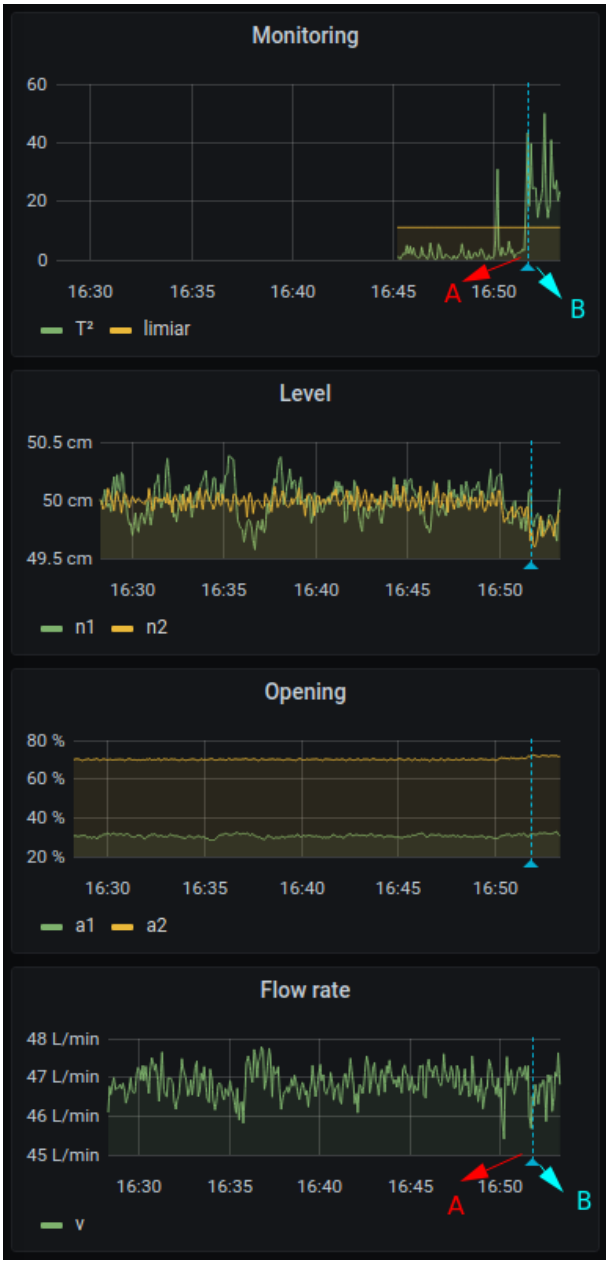

Fig. 5. Graphs for pilot plant monitoring in Grafana.

Discussion of the results During normal operation, the statistic crossed the threshold only 1% of the time, for a level of significance of 95%. All monitored variables had their values faithfully represented according to the presets set in the plant, described in the table 1. The measures of

the levels of the tanks varied around 50cm, between 49.6cm and 50.4cm. The openings a_1 and a_2 were 30% and 70% respectively. And the v flow of the pump remained close to 47L/min.

Upon introducing the fault, the pump's v flow decreased to 46L/min and the valve openings a_1 and a_2 needed to increase to 32% and 72% respectively, in order to try to recover the drop in the levels of tanks that went to 49.8cm. This change was successfully detected by Hotelling's T^2 algorithm, indicating the failure, one sample after its introduction.

Training and monitoring was performed totally in the edge device using data from the pilot plant received via OPC UA communication. The dashboards with the results are available via browser, and can be readily acessed by the pilot plant operator. Similarly, new variables can be made available to train algorithms for different faults, using specific dashboards. The huge variety of machine learning functions available in libraries of Python language allow the development of new algorithms to monitor faults and performance. The edge device resources (memory, CPU, storage) must be monitored to consider the allocation of new tasks to new nodes, as will be discussed in the next application.

3.2 Application to synthetic data

This application aims to monitor the performance of the IC as its load is increased. Instead of the five variables from the pilot plant, an easy task, one hundred random variables with normal distribution were generated. These variables are used for training (200 samples) and then monitoring starts, reading the variables via OPC each 1s and processing. After 30 readings, the performance metrics are evaluated and procedure is repeatead for two hundred variables. At each new step, one hundred more variables are added and the IC resources are monitored. The purpose is twofold: to check its capacity and to allocate tasks to new nodes when its capacity reaches its limit.

The same multivariate statistic was used to train the model and to monitor the variables. These variables were generated and made available by an OPC UA server created with the `opcua` library in Python, on a computer on the same local network as the IC.

Two tests were carried out. In the first, the IC was the unique node (node 0, Figure 1), and all the processing was done in this node. In the second test, a second node (node 1, Figure 1) was introduced in the network. Now, the IC reads the variables, processes requests, stores data and displays the results in Grafana. Node 1 is responsible for reading variables in RabbitMQ, executing the multivariate test, and return the results do RabbitM everytime a new OPC reading takes place.

Three metrics were evaluated:

- **CPU Consumption:** Measures CPU consumption in %.
- **Memory Consumption:** Measures memory consumption in %.
- **ΔT**: Time in [ms], for the IC to read the variables from the OPC UA server.

The consumption of CPU and memory were collected by Telegraf, with five measurement per second used to calculate the average value. The time ΔT is calculated by Node-RED itself, measuring the time between the moment that the variables are requested from the OPC UA server, and the moment that receives the last variable of that request.

For both tests, these metrics are measured 30 times, for 30 samples of the variables collected for monitoring, and a box plot is generated to evaluate their median and the dispersion. The metrics are measured only in the IC (node 0), in order to evaluate its load. If consumption of CPU or memory are close to their limits, tasks should be allocated to other nodes. Similarly, if ΔT is greater than the sampling rate of 1s, the number of variables reached the limit.

In the second test the sampling rate of 1s is maintained, and a second processing node (Node 1) was added, and the RabbitMQ client in Python from IC (Node 0) was disconnected.

Results The boxplots in figure 6 compare the performance metrics for the two tests, for the number of variables acquired and processed ranging from 500 to 3500.

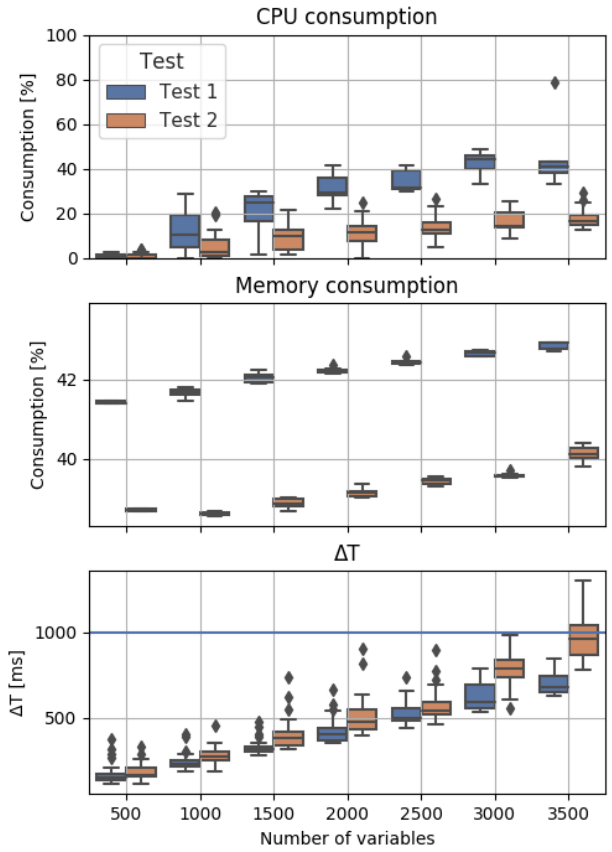

Fig. 6. Performance metrics for both tests.

In the first test, the CPU consumption increases as the number of variables increases, but the median is smaller than 44%. Memory consumption median is smaller than 43% and has increased by only 1% increasing the number

of variables. CPU and memory consumption are very dependent of the algorithms that are running in the node. The time to read the variables (median) ranged from 100ms to 600ms. Thus, even for 3500 variables ΔT is smaller than 1s, the sample time selected for monitoring.

In the second test, one can see that the CPU consumption prevailed below the consumption observed in the first test. The memory consumption was smaller than 40.2%. The reduction of CPU and memory consumption is explained by the fact that the RabittMQ monitoring client is disconnected from node 0. The reading time ΔT was similar up to 2500 variables, since in both tests this task is performed only by node 0. The extra time for more than 2500 is due to the data traffic from RabbitMQ to node 1, which is managed by node 0, where ΔT is being measured. This metric would require concern for more than 3500 variables. In this paper, the sampling rate for OPC readings was 1s. It could be increased to allow increasing the number of variables. As discussed in Cavalieri and Chiacchio (2013), the read and write services must be optimised for bulk read/write operations and not for reading/writing single values, as done in this paper. Another possibility to reduce this metric is to include a new node for OPC readings, sending data to RabbitMQ on node 0, that manages the tasks.

Monitoring these metrics is fundamental to ensure the edge device is able to accomplish its tasks. When limits are reached for any of these metrics, new edge computing nodes can be added to share the load. The extra nodes can share tasks related to reading variables via OPC, processing variables available on RabbitMQ, with node 0 managing all nodes and responsible for visualization.

The example with 2 nodes (second test) shows that tasks can be easily allocated to new nodes using this platform. The dynamic allocation of tasks is beyond the scope of this paper, and the interested reader can find some proposals in Rego et al. (2013).

4. CONCLUSION

A scalable edge computing platform for industrial process monitoring was proposed. This platform brings powerful machine learning algorithms close to end users, reducing costs, increasing safety and readiness for increasing burden.

An industrial computer based on Raspberry Pi 3 was configured and programmed to be an edge device, communicating with industrial processes via OPC UA standard. Web services for storage, message broker, visualization dashboards based on open source sofware were programmed and configured in the edge device, using Python language.

An application to a pilot plant controlled and supervised by a comercial distributed control system was performed, appling well known multivariate statistical process monitoring to the control loops. The usual steps for training and monitoring were performed and show that any industrial process with OPC UA communication could be considered as well. The great advantage in this case is privacy, avoiding data leaving the industry automation network.

A second application was performed to evaluate the performance of the edge device, using three performance metrics. They alowed monitoring the vicinity of the performance limits and additionaly to consider the allocation of tasks to new edge computing nodes. Up to 3500 variables were acquired and processed by a single node every second. A second node was added to share computational burden, showing with these good results that the proposed platform is ready for a huge variety of apllications.

5. ACKNOWLEDGMENT

This work was supported in part by the FAPES, that provided funds for the pilot plant and DCS. The authors are grateful to 2Solve, for providing the industrial computer and all support to explore its resources.

REFERENCES

Beňo, L., Pribiš, R., and Leskovský, R. (2019). Processing data from OPC UA server by using edge and cloud computing. *IFAC-PapersOnLine*, 52(27), 240–245. URL `https://doi.org/10.1016/j.ifacol.2019.12.645`.

Cavalieri, S. and Chiacchio, F. (2013). Analysis of opc ua performances. *Computer Standards Interfaces*, 165–177.

Ding, L., Wang, Z., Wang, X., and Wu, D. (2020). Security information transmission algorithms for iot based on cloud computing. *Computer Communications*, 32–39.

IBM (2013). Node-red. international business machines corporation. URL `http://nodered.org/`. Accessed: 1-June-2020.

InfluxData (2020). Influxdb. https://www.influxdata.com/products/influxdb/. Accessed: March 14, 2021.

Johansson, K., Horch, A., Wijk, O., and Hansson, A. (1999). Teaching multivariable control using the quadruple-tank process. volume 1, 807 – 812 vol.1.

Labati, R.D., Genovese, A., Piuri, V., Scotti, F., and Vishwakarma, S. (2020). Computational intelligence in cloud computing. In *Recent Advances in Intelligent Engineering*, 111–127. Springer.

MacGregor, J. and Kourti, T. (1995). Statistical process control of multivariate processes. *Control Engineering Practice*, 3(3), 403–414.

Mohan, N. (2019). Edge computing platforms and protocols. *PhD Thesis - University of Helsinki*.

Rego, P., Coutinho, E., and de Sousa, J. (2013). Estratégias para alocação dinâmica de recursos em um ambiente híbrido de computação em nuvem. *XII Workshop em Clouds e Aplicações (WCGA2013)*, 1–12.

Srivastava, P. and Khan, R. (2018). A review paper on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8(6), 17–20.

Tan, R., Cong, T., Ottewill, J.R., Baranowski, J., and Thornhill, N.F. (2020). An on-line framework for monitoring nonlinear processes with multiple operating modes. *Journal of Process Control*, 89, 119–130.

Vandikas, K. and Tsiatsis, V. (2014). Performance evaluation of an iot platform. *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, 141–146.