Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.sciencedirect.com/journal/computers-in-industry

Implementation of a scalable platform for real-time monitoring of machine tools

Endika Tapia [a,*], Unai Lopez-Novoa [a,b], Leonardo Sastoque-Pinilla [a], Luis Norberto López-de-Lacalle [a,c]

[a] Advanced Manufacturing Center for Aeronautics, University of the Basque Country, Biscay Science and Technology Park, 202, Zamudio, 48170, Spain
[b] Department of Computer Languages and Systems, University of the Basque Country, Rafael Moreno pasealekua, 2-3, Bilbao, 48013, Spain
[c] Department of Mechanical Engineering, University of the Basque Country, Torres Quevedo enparantza s/n, Bilbao, 48013, Spain

ARTICLE INFO

Keywords:
Big data
Aeronautical manufacturing
Machine tool
Scalable data processing
Parallel and distributed computing

ABSTRACT

In the new hyper connected factories, data gathering, and prediction models are key to keeping both productivity and piece quality. This paper presents a software platform that monitors and detects outliers in an industrial manufacturing process using scalable software tools. The platform collects data from a machine, processes it, and displays visualizations in a dashboard along with the results. A statistical method is used to detect outliers in the manufacturing process. The performance of the platform is assessed in two ways: firstly by monitoring a five-axis milling machine and secondly, using simulated tests. Former tests prove the suitability of the platform and reveal the issues that arise in a real environment, and latter tests prove the scalability of the platform with higher data processing needs than the previous ones.

1. Introduction

Nowadays, industrial environments are embracing Information and Communication Technologies (ICT) techniques in order to digitize and optimize all sorts of processes (Givehchi et al., 2017). This trend, usually referred to as the fourth industrial revolution (Industry 4.0) has been followed by many companies and, thus, in 2021, the European Union stated that the fifth revolution (Industry 5.0) was already underway. This transition would be not just digital, but also green.

There are several ICT techniques used in these industrial revolutions, like Big Data and Internet of Thing (IoT), but most of them have a common pillar: to analyze the vast amounts of information produced by the increasingly digitized industrial environments (Ariyaluran Habeeb et al., 2019). This is possible thanks to the capabilities of current manufacturing machines, which allow to collect monitoring data that can be analyzed in real time or offline (Qi and Tao, 2018).

In the case of the aeronautical manufacturing sector, the main target of using ICT techniques is to improve productivity, material quality and reduce manufacturing costs (Wang et al., 2022b). Given the stringent tolerance requirements for workpieces in aeronautical manufacturing, it is crucial to put in place measures that enable predictive maintenance or online defect correction with the shortest refresh times and lowest possible latencies (Dalzochio et al., 2020).

This work presents the design and implementation of a software platform that monitors an aeronautical machining centre in order to identify outliers in the fabrication process in real time. The platform is built using open source tools and deployed in several Linux-based virtual machines, to allow for scalability. An industrial protocol is used to retrieve data from a manufacturing machine at high speed and Apache software tools for data treatment, processing, storage and visualization. The aim is to detect outliers during a machining process and to show them graphically on a dashboard.

This work has been carried out within the Advanced Manufacturing Centre for Aeronautics[1] (CFAA, for its Spanish initials). CFAA works in areas like additive manufacturing (Sendino et al., 2020), machining process (Rodríguez et al., 2021), non-conventional processes (del Olmo et al., 2022), and laser mechanisms (Wang et al., 2019), among others. The main objective of the centre is to develop novel techniques and technologies that can improve the production processes for propulsion systems and engine components.

Beyond this work and in the scope of Industry 4.0, there are many others that demonstrate the successful application of Big Data and scalable technologies. For example, Canizo et al. (2019), propose a platform that performs real-time analysis of industrial manufacturing systems, which they validate with data from press machines. Ji et al. (2019), optimize the machining conditions while simultaneously choosing the

* Corresponding author.
E-mail address: endika.tapia@ehu.eus (E. Tapia).
[1] CFAA: https://cfaa.eus.

https://doi.org/10.1016/j.compind.2023.104065
Received 20 October 2023; Received in revised form 13 December 2023; Accepted 15 December 2023
Available online 19 December 2023

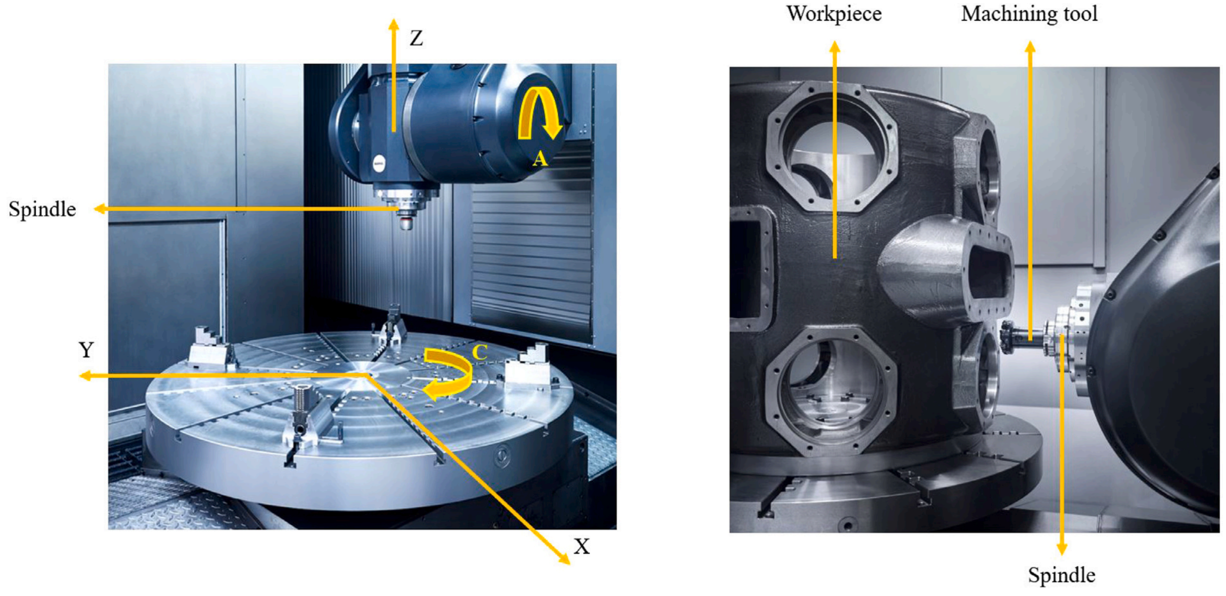

Fig. 1. Components of the Ibarmia™ THR 16 (Ibarmia THR 16, 2022).

optimal machine tool and cutting tool for a given process using Big Data and artificial neural networks. Moldovan et al. (2021), use Apache Spark to analyze defects related to discrete manufacturing processes. Landi et al. (2020) present a platform to analyze machining tool data using open source components and a custom-made web interface. Shi et al. (2021) build a software pipeline using popular Apache tools to monitor a machine tool with a Siemens interface.

In addition, there are two relevant trends in this area that should be noted. One is the development of Digital Twins, software applications that replicate the activities of a machine. These are used, e.g., to detect deviations in industrial processes (Fuller et al., 2020) and to predict the behavior of machines in real time (Wang et al., 2022a). The other trend is the shipping of monitoring tools with the machine tools themselves. Examples of these are Celos (2023) and Active Cockpit (2023) from Bosh Rexroth and DMG Mori respectively.

The remainder of this paper is structured as follows: Section 2 presents the machine being monitored in detail. The software tools and architecture of the monitoring platform are presented in Section 3, followed by a description of the algorithm used to detect outliers in Section 4. A performance evaluation of the platform is described in Section 5 and, finally, conclusions and future research directions are presented in Section 6.

2. Machining process

The Ibarmia THR 16 (2022) is a machining centre that combines different technologies in a single machine: milling, drilling, turning, gear cutting and grinding. It is considered a *Multiprocess* machine, as it allows machining different types of pieces with the same tool-set. This reduces the quantity of parts that must be manufactured in batches, which shortens the production life cycle and reduces the amount of shifts between machines in a factory.

Fig. 1 highlights the most distinctive components of the THR 16. The milling machine is comprised of linear axes X, Y and Z and rotary axes A and C. A workpiece to be machined is placed on top of the rotatory table (circle shaped component in the bottom), which enables movement along the X and Y axes. Rotary axes are available to adjust the angular position of the machining tool and workpiece. The spindle assumes responsibility for rotating the tool and accurately positioning and clamping it during turning operations. A video showing the machine at work can be found online.[2].

Table 1
Monitored variables.

Variable name	Description	Dimensions
LOAD	Axis drive load in %	6
POWER	Drive power of the axis in W %	6
RPM	Rotation speed (respect to maximum speed) of the spindle in %	6
LOAD SPINDLE	Spindle motor load in %	1
RPM SPINDLE	Spindle motor revolutions in RPM	1
SPINDLE OVERRIDE	Spindle drive override in %	1
RAPID OVERRIDE	Rapid spindle drive override in %	1
TOTAL	–	**22**

2.1. Monitored variables

The THR 16 has a *Siemens® Sinumerik 840D SL* computer numerical control (CNC) integrated with a programmable logic controller (PLC) that, among others, collects monitoring data in the form of a set of variables.

The PLC can monitor up to 300 variables, including currents, hydraulic flows, temperatures, alarms, etc. From all these, those related to the load, power, and rotational speed of the machine's five axes and the spindle are considered the most relevant ones because they represent the state of the machine's critical components and can reveal issues during a manufacturing process. These variables are described in Table 1. Variables like load, power and RPM have 6 dimensions, 5 of them with the value for the X, Y, Z, A and C axes and the 6th one for the spindle.

2.2. The OPC-UA protocol

Open Platform Communications Unified Architecture (OPC-UA) is a cross-platform communication protocol intended for secure and reliable data exchange in the industrial automation space (Cavalieri and

[2] Ibarmia THR16 *Multiprocess*: https://youtu.be/QioDWNN8S9c

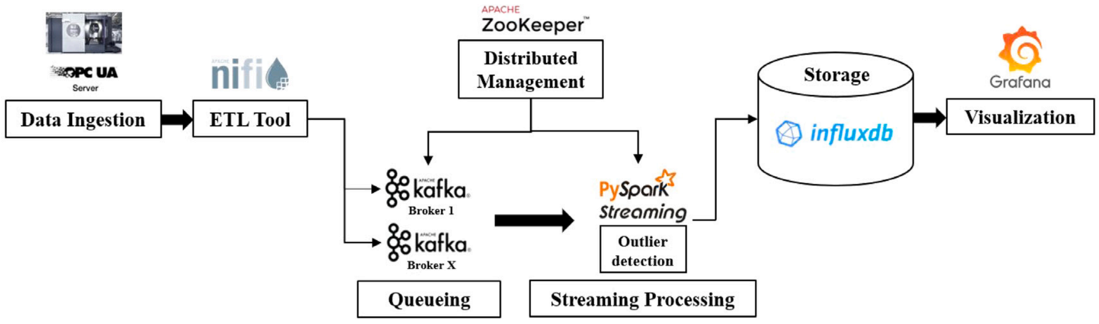

Fig. 2. Design of the software architecture.

Chiacchio, 2013). This protocol replaces an earlier version of OPC and provides improved features for digitization, independent of platform and manufacturer.

OPC-UA allows a constant data flow between multiple devices and control applications with limited restrictions, as well as serving as a means of communication between Supervisory Control and Data Acquisition (SCADA) applications and sensors (Ferrari et al., 2018). For this purpose, an OPC-UA environment will be considered as a network of one or more clients and servers. The software side of the platform will act as an OPC client and the PLC of the five-axis milling machine as the OPC server.

It is important to note that OPC-UA operates as a bidirectional protocol, allowing data exchange from both the server to the client and vice versa. This capability enhances dynamic communication in industrial automation. For example, an OPC client, serving as a supervisory control system, can send instructions to the OPC server (e.g., the milling machine's PLC), enabling real-time adjustments like milling speed or tool configurations.

On regards to security, OPC-UA supports Transport Layer Security (TLS) or Secure Sockets Layer (SSL) to ensure safe and authenticated data transfer, complemented by flexible access controls (Tapia et al., 2023). Nevertheless, similar to many other industrial protocols, using OPC-UA poses security challenges like encryption issues and addressing potential discrepancies in software and hardware integration. These challenges are set to be addressed by the regulations outlined by the European Parliament (2023), highlighting the urgent need for improved security in industrial environments.

In terms of capture frequency, OPC-UA is typically used to monitor a reduced set of variables (from 1 to 5). According to the milling machine specifications, the minimum sampling interval for 1 variable is 100 ms, but this latency increases if more variables are requested. This is explained in more detail in Section 5.1.1.

3. Monitoring platform

This section presents the software platform in two parts: Section 3.1 describes the software building blocks and 3.2 presents the design of the infrastructure, including the connection between the different software tools.

3.1. Software toolset

- Apache NiFi (2022) is a distributed Extract, Transform, Load (ETL) tool that enables to create data flows between different types of systems in a visual way. It allows to perform transformations and cleaning on data by dragging and connecting *processors*, the basic building block of NiFi. Each *processor* provides a different functionality, like retrieving data, applying a filter or forwarding data.
- Apache ZooKeeper (2022) is a centralized service used to maintain configuration information and naming, as well as to provide distributed synchronization and group services. The service will implement consensus, group management, and presence protocols so that the applications do not need to implement them on their own.
- Apache Kafka (2022) is a data stream queuing system oriented to publish, store, process and subscribe to streams of records immediately. It is designed to distribute data streams to different consumers while handling data streams from different sources. Kafka is especially used for exchanging large amounts of data in real time with low latency. Data is organized and stored in *topics* (equivalent to a folder in a file system).
- Apache Spark Structured Streaming (Karau et al., 2015) is an stream processing engine built on top of Apache Spark, an scalable, open source and unified data processing framework. Batch data are aggregated and analyzed through the streaming process, sometimes referred as the data stream (Karau et al., 2015).
  Spark follows a manager/worker design, with a master process and one (or more) worker process. The manager process sets up the environment and schedules job executions. Worker processes execute the actual data processing.
- InfluxDB (2022) is a NoSQL, open source time series database. It has been chosen for the storage of timestamped data due to its high performance and reliability. Also, it provides a SQL-like language (InfluxQL) with which is possible to query data from *measurements*, equivalent to tables in relational databases.
- Grafana (2022) is a software that provides visualization by creating dashboards and graphs from multiple sources, including time series databases. Its user-friendly interface allows the creation of different types of panels that can be used to depict the data generated by the machinery over the course of a given period of time.

3.2. Design

The architecture of the platform, built with the software tools described in Section 3.1, is depicted in Fig. 2. It should be noted that only one data stream is being considered, which starts in the monitored machining centre and ends in a graphical visualization.

The software building blocks are deployed in different virtual machines, forming a distributed system architecture. All these machines form a Kafka cluster, which works in a distributed way to store, receive and send messages between its different brokers. The brokers will be managed and coordinated by Zookeeper, who will also determine a broker to act as the *topic* leader.

First of all, a set of variables is obtained from the milling machine through OPC-UA. These variables are extracted and transformed by NiFi, which builds data streams through *processors*. In this platform, two custom *processors* have been configured to establish a data flow between the OPC Server and NiFi.

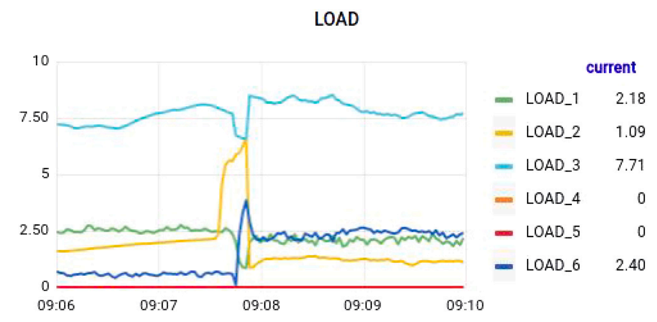

Fig. 3. Sample *graph* chart with load values of the axes and spindle.

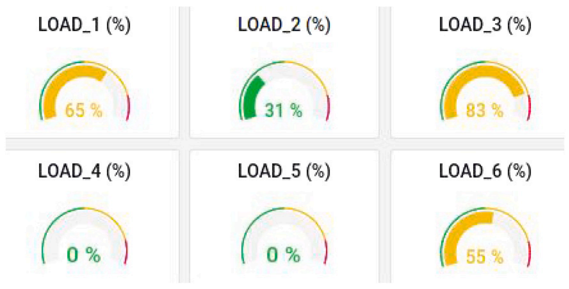

Fig. 4. Sample *gauge* charts with load percentages of the axes and spindle.

NiFi sends the batch data in JSON format to a Kafka *topic*, which will keep the data until its consumed by Spark Streaming, ensuring the order of delivery. Next, Spark's Structured Streaming library processes and analyzes batch data using *DataFrames*. The outliers will be detected by applying the technique described in Section 4. Then, the data processed by Spark is sent to InfluxDB for its storage, sorted by timestamp.

Finally, visualization is provided by Grafana, which obtains data by making queries to InfluxDB. A dashboard shows the 22 variables described in Section 2.1, mainly with two types of charts: *graphs* and *gauges*. An example of the former is depicted in Fig. 3, with some load values of the axes and the spindle captured in real time, and an example for the latter in Fig. 4, with several *gauges* displaying the capacity of the axes and the spindle of the THR 16 at a given moment.

4. Outlier detection technique

In its current state, the platform detects outliers produced in real time for any of the monitored variables. To this end, the Inter-Quartile Range (IQR) is implemented, a well know method to find outliers in continuous data (Vinutha et al., 2018). IQR is used in many domains as an analysis method and often as part of the Boxplot technique for data representation (Bruce et al., 2020).

For a given set of variables, IQR will detect as outliers the values that fall out of a set of given limits. For this purpose, it has to be trained with data that represents the target environment.

Fig. 5 depicts the overall calculations of this technique: based on an input dataset, IQR computes a range consisting of the difference between the first and the third quartile for each variable. It is necessary to select percentiles that correctly fit this case. Therefore, it has been decided that the first quartile corresponds with the 25th percentile and third quartile with the 75th percentile.

The IQR measure is considered to be:

$$IQR = Q3 - Q1 \tag{1}$$

where Q3 and Q1 represent the 3rd and 1st quartile for each variable. Next, the lower and upper limits of each variable are calculated with the Eqs. (2) and (3):

$$\text{Lower limit} = Q1 - (1.5 * IQR) \tag{2}$$

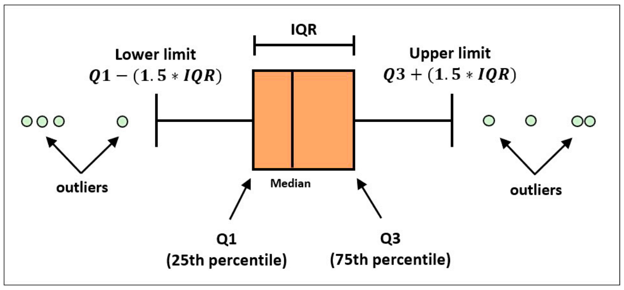

Fig. 5. *Boxplot* representation of the IQR method.

$$\text{Upper limit} = Q3 + (1.5 * IQR) \tag{3}$$

The limits of each variable are used to detect outliers with every new piece of incoming data. In other words, each time Spark Streaming receives a batch of variables, it will be compared to the limits extracted from the dataset containing the historical data. Thus, values that are beyond the lower or upper limit will be considered as outliers.

Finally, the state of a variable will be represented as a percentage of the capacity at which a variable is working relative to the known upper bound. Eq. (4) shows how to obtain the percentage.

$$\text{Pct (capacity in \%)} = \frac{\text{Current value}}{\text{Upper limit}} * 100 \tag{4}$$

5. Performance evaluation

In this section, we present a set of tests that assess the performance of the platform. Section 5.1 outlines experiments that evaluate the platform's performance in the context of the five-axis machine scenario and Section 5.2 provides an evaluation of the platform's performance using a simulator. Section 5.3 presents an analysis on the module that runs IQR, firstly on its capacity to detect outliers and secondly on the overheads for its training.

In these tests, the IQR module has been trained with historical records of various manufacturing processes (machining of different pieces) conducted on the THR 16 over a two-year period. These records contain data for the 300 variables monitored by the PLC and they were collected at a frequency of 1 s.

In addition, the whole software infrastructure was running in virtual machines (VM) hosted by a private cloud located within CFAA premises (Sasiain et al., 2020). The Ibarmia™ THR 16 and the data centre are communicated through a local Ethernet network. Every VM had the same following specifications:

- Virtual Cores (CPUs): 2
- CPU frequency: 2.1 GHz
- RAM Memory: 8 GB
- Disk space: 128 GB
- Operating system: Ubuntu 20.04

Three types of VMs have been defined, each hosting a subset of software tools:

- VM 1: NiFi, Spark Streaming manager, Zookeeper coordinator
- VM 2: Kafka broker, Spark Streaming worker
- VM 3: InfluxDB, Grafana

In this work, each VM of type VM2 hosts a single Spark Streaming worker with multi-threading capabilities, which is able to use both virtual cores. We also want to note that the data capture frequency remains constant over time, both in transmission and monitoring.

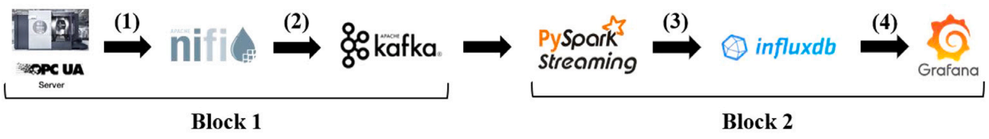

Fig. 6. Blocks of time dissection.

Table 2
Time dissection of the whole pipeline while fetching data from the THR 16.

Number of variables	Block 1		Block 2		Total
	(1)	(2)	(3)	(4)	
1	12 ms	8 ms	13 ms	12 ms	**0.045 s**
5	43 ms	9 ms	47 ms	13 ms	**0.112 s**
10	82 ms	10 ms	83 ms	15 ms	**0.190 s**
22	152 ms	12 ms	177 ms	17 ms	**0.358 s**
50	356 ms	14 ms	322 ms	20 ms	**0.712 s**
100	629 ms	16 ms	524 ms	27 ms	**1.19 s**
300	1975 ms	20 ms	760 ms	39 ms	**2.79 s**

5.1. *Evaluation using the Ibarmia™ THR 16 machine*

The experimental results of this section were obtained while a turbine blade was being manufactured, in order to detect outliers in the machining process. One VM of each type has been used.

5.1.1. *Analysis of the whole pipeline*

The first test was a runtime analysis for the whole pipeline of the software tools. To this end, the stages of the pipeline have been separated in two blocks, as shown in Fig. 6:

- Block 1: Data captured from the machine using NiFi (1) and queued in Kafka (2).
- Block 2: Data processed with Spark, stored in InfluxDB (3) and visualized on Grafana (4).

We measured the time taken by each stage of the block when retrieving batches of different sizes (i.e. a distinct set of variables from the THR 16 was retrieved for each batch). Even if the 22 variables described in Section 2.1 are the most relevant ones for the use case, we gathered performance metrics for larger batches to test the scalability of the platform.

Table 2 shows the average result of seven iterations. Results show that both blocks of the pipeline take a similar amount of time, except for the case of 300-variable large batches, where the data capture from the machine takes much longer than any other stage of the process.

It can be concluded that the retrieval of data from the machine is the main bottleneck for Block 1 and that the Spark processing takes the most time for Block 2, as the time required to store and retrieve data from InfluxDB is significantly smaller.

As a follow-up test, we performed an analysis of the CPU and memory consumption of the whole toolset. Tests have been done for batches of 50, 100 and 300 variables and were conducted using Linux's *top* command for a period of 2 min while all resources were up and running.

Results for this test are shown in Tables 3–5 for batches of 50, 100 and 300 variables respectively. We can see that, in the three cases, the most resource consuming tool is the Spark worker, due to its heavy use of *DataFrame* data structures to process incoming data in real time. The CPU use of InfluxDB increases considerably too with larger batches of variables, but it is handled correctly with the current setup. We can conclude that, overall, the setup of 3 VMs is enough to handle the largest possible batch of variables coming from the machining centre.

Table 3
CPU and memory usage of the pipeline for batches of 50 variables.

Software tool	VM 1		VM 2		VM 3	
	CPU	RAM	CPU	RAM	CPU	RAM
Spark worker	–	–	65.8%	14.3%	–	–
Java (JVM)	–	13.3%	–	11.9%	–	8.7%
NiFi	11.3%	12.4%	–	–	–	–
Spark manager	0.3%	3%	10.6%	2.9%	–	–
Kafka + Zookeeper	6.8%	8.9%	1.4%	5.9%	–	–
InfluxDB	–	–	–	–	32.7%	8.1%
Grafana	–	–	–	–	1.4%	1.2%
TOTAL	**18.4%**	**37.6%**	**77.8%**	**35%**	**34.1%**	**18%**

Table 4
CPU and memory usage of the pipeline for batches of 100 variables.

Software tool	VM 1		VM 2		VM 3	
	CPU	RAM	CPU	RAM	CPU	RAM
Spark worker	–	–	126.5%	15.7%	–	–
Java (JVM)	–	13.7%	–	12.3%	–	8.8%
NiFi	19.2%	14.6%	–	–	–	–
Spark manager	0.3%	3%	11.2%	2.9%	–	–
Kafka + Zookeeper	7.3%	9.1%	1.4%	6%	–	–
InfluxDB	–	–	–	–	75.4%	25%
Grafana	–	–	–	–	1.6%	1.2%
TOTAL	**26.8%**	**40.4%**	**139.2%**	**36.8%**	**77%**	**35%**

Table 5
CPU and memory usage of the pipeline for batches of 300 variables.

Software tool	VM 1		VM 2		VM 3	
	CPU	RAM	CPU	RAM	CPU	RAM
Spark worker	–	–	140.8%	17.1%	–	–
Java (JVM)	–	13.8%	–	12.5%	–	8.8%
NiFi	24.1%	16.6%	–	–	–	–
Spark manager	0.3%	3%	12.2%	2.9%	–	–
Kafka + Zookeeper	7.5%	9.2%	1.4%	6%	–	–
InfluxDB	–	–	–	–	85.3%	32%
Grafana	–	–	–	–	1.7%	1.2%
TOTAL	**31.9%**	**42.6%**	**154.4%**	**38.5%**	**87%**	**42%**

5.1.2. *Analysis of the data capture stage*

Given that the retrieval of variables from the OPC server is the main bottleneck of the pipeline, we conducted two additional tests to get more insight into this stage.

The first test has been to examine the performance of NiFi. As described in Section 3.1, actions in NiFi are configured through the use of *processors*. NiFi is bundled with several built-in *processors* that enable connectivity using different protocols but, in this work, we used ones developed by Zylk (2023), a third party, to set OPC-UA communication. These *processors* were an upgrade from another third party (Linksmart (2022)) *processors*. Here in after, linksmart's *processors* will be referred to as default OPC-UA *processors*. A performance comparison of these default OPC-UA *processors* and Zylk's ones has been conducted when fetching batches of different sizes from the THR 16.

Results of this test are depicted in Fig. 7, where it is clear that Zylk's *processors* provide much better performance for batches of any size. Further investigation revealed that the THR 16's CNC sets a minimum sampling interval of 100 ms, which limits the performance of the default *processors*. However, it has been possible to obtain variables with a higher frequency thanks to Zylk's customized *processors*.

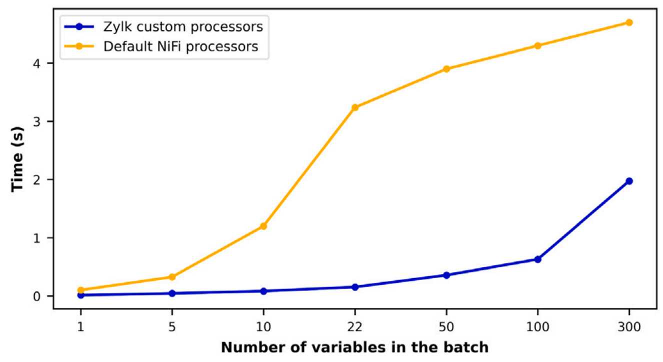

Fig. 7. Performance comparison of NiFi's default and Zylk's OPC-UA *processors*.

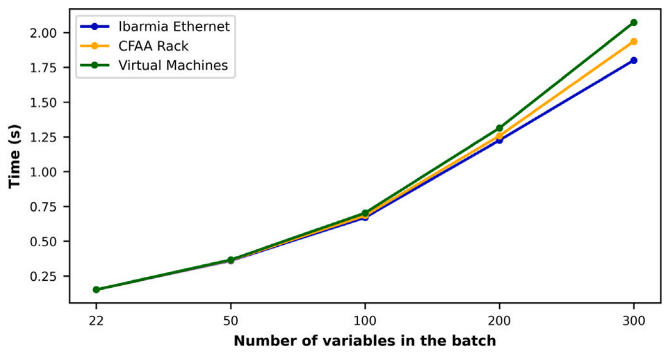

Fig. 8. Network latency with different configurations.

In order to dig deeper into the main bottleneck of Block 1, we investigated the latencies induced by the network in CFAA. In particular, we measured the time taken by NiFi to fetch batches of different sizes from the THR 16 in 3 different ways: connecting directly with an Ethernet cable to the machine, connecting to the Ethernet switch in the Rack where the rest of the CFAA workshop machines are linked and a connection through the virtual machines mentioned above. In order to measure the times of the first two connections, we used a laptop running NiFi exclusively.

Fig. 8 shows the results of the test, where it can be seen that the network has reduced impact in the capture of the data for batches with 100 variables or less. In the largest scenario, with batches of 300 variables, there is a 270 millisecond time difference between requests made directly to the milling machine and those made from a virtual machine.

5.1.3. Analysis of the spark structured streaming stage

In this section, we provide an analysis of the most relevant performance metrics of Spark Structured Streaming, as part of the data processing pipeline. These metrics have been obtained from Spark's internal web monitoring module. As a note, in this section, we shall refer to a batch of variables as a *record*, which is a term used in the Spark environment.

Firstly, we compared the *input rate* (the aggregate rate of data arriving to Spark) with the *process rate* (the aggregate rate at which Spark is processing data). This provides a measure of the capacity of Spark to process data in an streaming manner.

Results for this test are shown in Fig. 9. With batch sizes of 22 variables or less, the input rate remains very close to the process rate, indicating no issues in the Spark stage. However, with batch sizes of 50 variables or more, the input rate starts to increase while the process rate decreases. At first sight, a situation where the input rate is significantly higher than the process rate should mean that Spark workers cannot

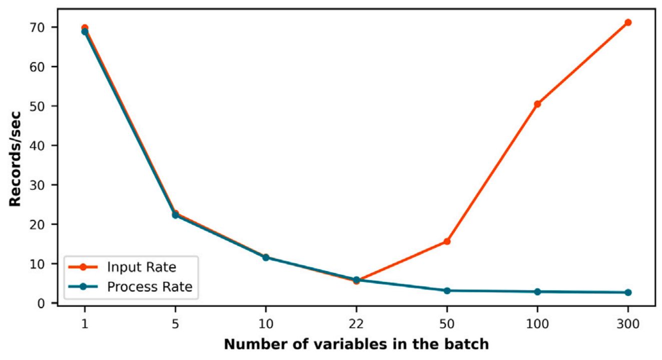

Fig. 9. Comparison of input rate and process rate in Spark processing data from the THR 16.

cope with the pace that the data is arriving at. However, we conducted additional tests with 2 and 3 Spark workers (with additional VMs of type VM2), but the ratio of input to process rate remained the same.

Further investigation revealed that the issue came from latencies induced by NiFi: Spark reports the input rate as an average value and, as NiFi sends data at the pace provided by the OPC server, there is not a regular flow of data with large batches, but one with stalls in between.

As a step further, we analyzed the *input rows* and *batch duration* values reported by Spark, which refer to the aggregate number of records processed in a trigger and to the time taken to process a micro-batch (a data structure, subset of a record) in a trigger respectively. In Spark terms, a trigger refers to the moment when a new set of data arrives to a *DataFrame*.

Fig. 10 depicts the time plots of the input rows and the batch duration for 1, 22, 100 and 300 variables in a batch. These metrics were measured for the last 100 Spark micro-batches of each test. Overall, the more variables are extracted in the NiFi batch, the fewer input rows arrive to Spark, as the sampling frequency decreases. In the case of a single variable, an average of 25 records are processed per trigger, whereas for 300 variables, only 0.7 records are processed.

Regarding batch duration, the increment of variables in the NiFi batch causes an increase in the time taken to process the micro-batch. With batches of a single variable, the micro-batch processing time on average is 332 ms, while for 300 variables, it takes 532 ms. In order to obtain the delay mentioned in the dissection of Table 2, the batch duration has been divided by the input rows. This way, it is possible to know how long it takes for the variables of the batch to be processed by Spark in the same trigger.

5.2. Evaluation using simulation

In this section, additional tests of the platform using a simulator are presented. The aim is to assess the behavior of the platform as a standalone system, with a different input data flow, to validate its scalability.

We developed a simulator that emulates the milling machine by sending batches of variables, read from a historic file generated on the THR 16. The simulator will replace the OPC server and NiFi in the current tests, as data will be sent directly to Kafka.

In these tests, we will focus on the largest batches (50, 100 and 300 variables), which will be sent every 5 ms. This will allow for an evaluation of the platform when data is received at a faster rate than the one provided by the OPC server. In contrast to the tests of the previous section, additional Spark workers will be added to explore scalability.

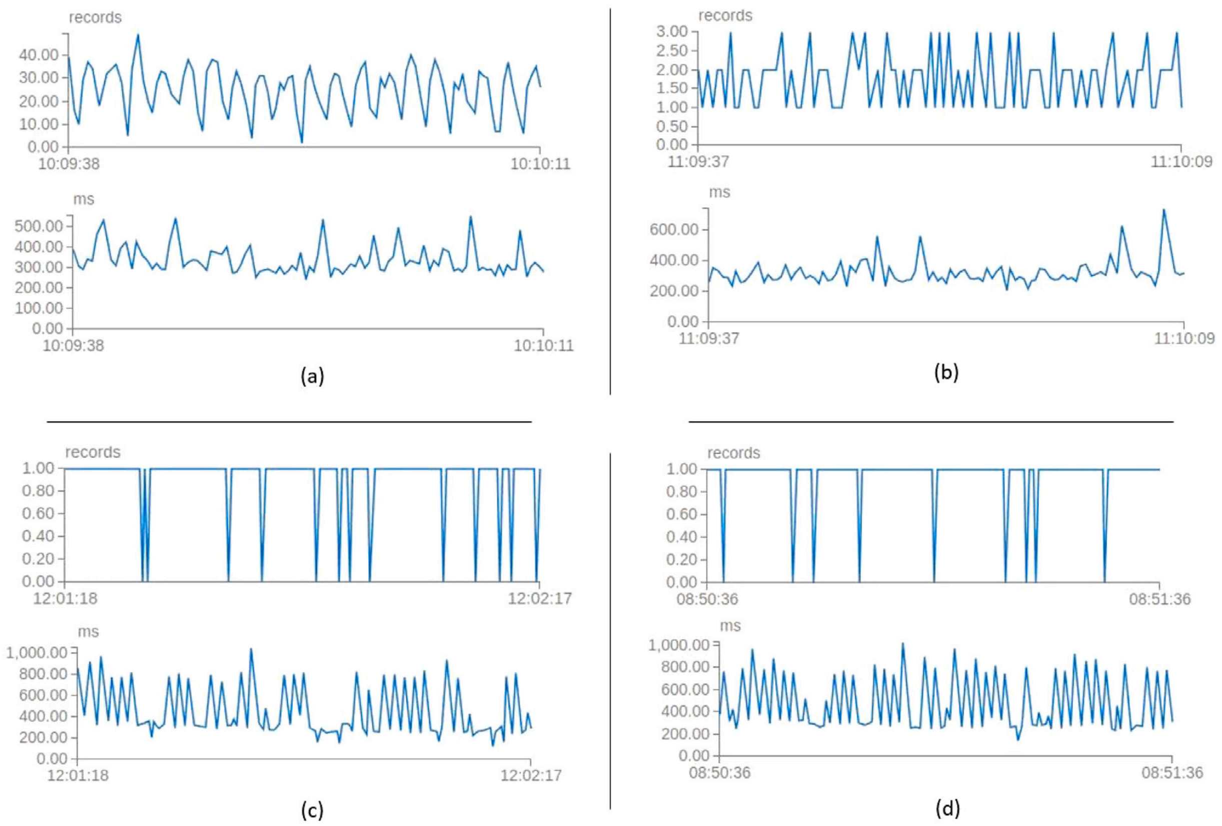

Fig. 10. Input rows (above) and Batch duration (below) for batch sizes of: (a) 1 variable, (b) 22 variables, (c) 100 variables, (d) 300 variables.

5.2.1. Analysis of the spark structured streaming stage

Spark performance metrics for these tests are presented in Table 6. For the case of 50 variables per batch, using 1 Spark worker processes up to 130.3 records/s on average, with a very similar input rate. In contrast, in the tests with the milling machine presented in the previous section, the input rate capped at 15.6 records/s and the process rate 3.4 records/s, confirming the issue with the data acquisition at machine level.

In the case of 100 variables per batch, we tested the platform with 1 and 2 Spark workers. We see that Spark is capable of maintaining similar input and process rates in both cases, and also confirm that the IQR implementation scales well: using 2 workers (i.e. 2 VMs of type VM2) makes the batch duration to be reduced roughly in half.

In the case of 300 variables per batch, a single Spark worker is not capable of processing the data in time at a rate of 5 ms, which causes the significant latencies. This is explored later in this section. With 2 and 3 workers, Spark can handle the incoming data at a proper rate.

In addition, we present in Fig. 11 the evolution of the number of input rows and batch duration for the case of 300-variable batch sizes. With a single worker, incoming batches from Kafka get retained but unprocessed, which turns Spark into a bottleneck as time goes by. With 2 and 3 workers, we can observe that the amount of input rows and batch duration remain roughly stable over time.

5.2.2. Analysis of the hardware resources

In this section, the CPU and RAM usage obtained from the simulator-based tests are described. As the simulator sends the batches directly to Kafka, VM1 machines are not used in these test, hence their results are not presented.

Table 7 shows the results for 50-variable batches with 1 Spark worker, Table 8 for 100-variable batches with 2 Spark workers and Table 9 for 300-variable batches with 3 Spark workers. In the two latter cases, the values for VM2 are the average for the VMs of this type.

In the case of 50-variable batches, we can observe that VM2 is close to using all the available CPU. This is in contrast to the tests with the THR 16, where an average of 77.8% of CPU was used. The higher use of CPU is likely due to the higher frequency of data arrival using the simulator. Regarding VM3, InfluxDB uses 60.1% of CPU, which is almost twice the amount used in the tests with the milling machines (34.1%).

In the cases of 100-variable and 300-variable batches, with 2 and 3 workers respectively, we can see that CPU usage is 120.7% at most, which means that current tests have not been limited by compute capacity. We can also see that the CPU usage by InfluxDB reaches 71.4% in the tests with 300-variable batches, far from being a bottleneck in VM3.

5.3. Analysis of the outlier detection module

In this last section, we provide some insight on the outlier detection module of the platform in two ways: firstly, on the outlier detection conducted by the IQR and, secondly, on the overheads of the module as part of the platform.

In order to understand the outlier detection capability of the platform, we conducted an statistical analysis on a subset of the dataset used in the previous section, which contains 812250 samples. We focus on the 22 variables described in Section 2.1. The values are shown in Table 10, where we provide the following values for each variable:

- Minimum, average and maximum values (*Min*, *Average* and *Max* columns).
- The lower and upper limits computed as part of the IQR (*Lower limit* and *Upper limit* columns).
- Number of detected outliers in each variable and percentage with respect to the total number of samples (*Num. outliers* and *% outlier* columns).

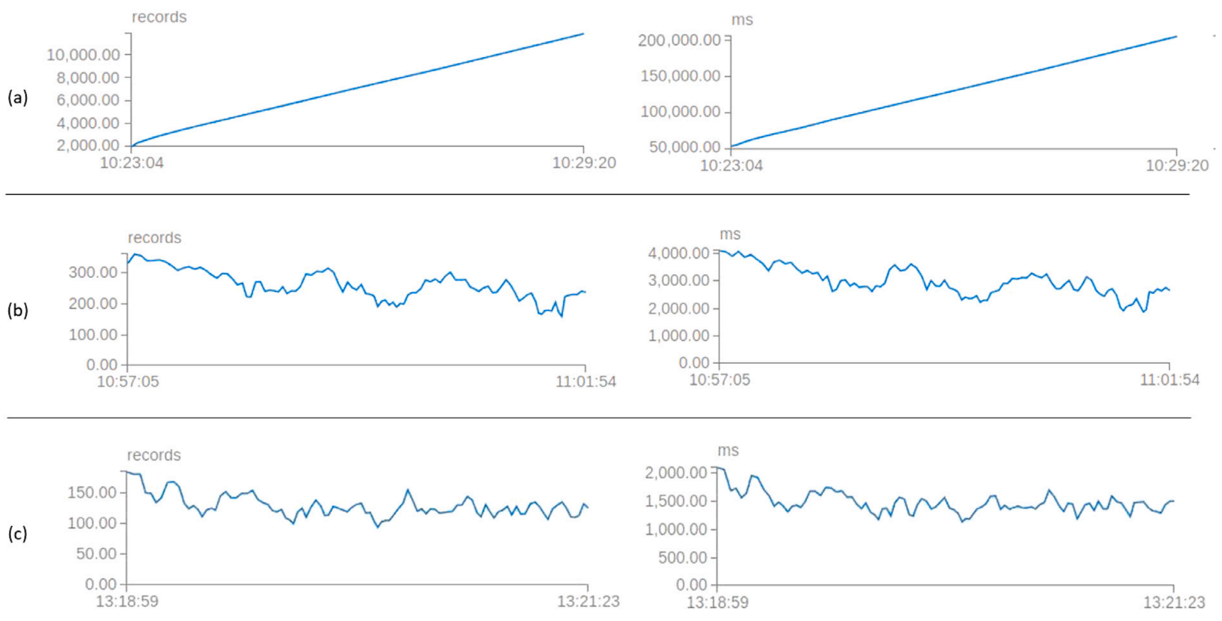

Fig. 11. Input rows (left) and Batch duration (right) for 300 variables using the simulator with 1 worker (a), 2 workers (b) and 3 workers (c).

Table 6
Spark Structured Streaming metrics using the simulator.

Number of variables	Number of workers	Input rate (records/s)	Process rate (records/s)	Input rows per worker (records)	Batch duration (s)
50	**1**	128.9	130.3	52	0.390
100	**1**	112.3	113.1	84	0.913
	2	114.7	115.8	54	0.450
300	**1**	70.6	38.6	>300 (linear increment)	>5 (linear increment)
	2	74.1	70.7	215	3.15
	3	87.1	87.5	124	1.42

Table 7
CPU and memory usage of the pipeline with batches of 50 variables, simulator-based tests with 1 Spark worker.

Software tool	VM 2		VM 3	
	CPU	RAM	CPU	RAM
Spark worker	80.8%	17.1%	–	–
Java (JVM)	–	12%	–	8.8%
Spark manager	9.7%	2.9%	–	–
Kafka + Zookeeper	1.5%	6.1%	–	–
InfluxDB	–	–	60.1%	8.1%
Grafana	–	–	1.2%	1.4%
TOTAL	**92%**	**38.1%**	**61.3%**	**18.3%**

Table 8
CPU and memory usage of the pipeline with batches of 100 variables, simulator-based tests with 2 Spark workers.

Software tool	VM 2		VM 3	
	CPU	RAM	CPU	RAM
Spark worker	95.6%	18.4%	–	–
Java (JVM)	–	12.6%	–	9.1%
Spark manager	10.7%	2.9%	–	–
Kafka + Zookeeper	1.5%	6.4%	–	–
InfluxDB	–	–	64.4%	21%
Grafana	–	–	1.2%	1.4%
TOTAL	**107.8%**	**40.3%**	**65.6%**	**31.5%**

Table 9
CPU and memory usage of the pipeline with batches of 300 variables, simulator-based tests with 3 Spark workers.

Software tool	VM 2		VM 3	
	CPU	RAM	CPU	RAM
Spark worker	106.8%	18.9%	–	–
Java (JVM)	–	12.9%	–	9.2%
Spark manager	12.3%	2.9%	–	–
Kafka + Zookeeper	1.6%	7.2%	–	–
InfluxDB	–	–	71.4%	27%
Grafana	–	–	1.2%	1.5%
TOTAL	**120.7%**	**41.9%**	**72.6%**	**37.7%**

The analysis shows that the percentage of outliers is relatively low for every monitored variable (2.53% in the largest case), which are the expected values for the experiments conducted in the Ibarmia machine. However, these values could be different in other machines and, to our knowledge, there is no definitive quantification in the literature regarding the acceptable number of outliers for a machining process. This is attributed to the intricate variability inherent in each stage of the machining process. Machining centers are designed under the assumption that internal and controllable factors, such as geometric compensation of moving machine components and minor variations in energy supply, minimally impact the process. External factors, including the material being machined, machining parameters, tool type and condition, coolant type, composition, and quantity, significantly influence the machining outcome.

Table 10
Statistics, IQR limits and outliers of historic dataset.

Variable	Min	Lower limit	Average	Upper limit	Max	Num. outliers	% outliers
LOAD(X)	0	−1.48	2.56	5.72	54.7	16 520	2.03%
LOAD(Y)	0	−3.5	3.14	8.5	59.6	2425	0.29%
LOAD(Z)	0	−4.35	7.05	18.3	100	2914	0.35%
LOAD(A)	0	−0.63	2.81	6.3	76.4	1267	0.16%
LOAD(C)	0	−0.65	0.54	1.1	34.3	8551	1.05%
LOAD(SP axis)	0	−1.84	348.3	6.03	1114.1	7129	0.87%
POWER(X)	−5387.4	−153.9	23.2	273.7	15 044.6	18 541	2.28%
POWER(Y)	−1994.3	−356.9	28.2	606.8	7466.5	16 091	1.98%
POWER(Z)	−3848.6	−4.85	81.5	8.09	11 383.9	20 588	2.53%
POWER(A)	−5742.9	−1406.3	563.7	2343.7	55 171.9	11 075	1.36%
POWER(C)	−7022.9	−1632.9	142.9	3225.4	45 074.2	7119	0.87%
POWER(SP axis)	−6093.8	−1101.6	350.9	3960.9	64 781.3	18 191	2.24%
RPM(X)	0	−0.46	3.07	0.75	100	14 231	1.75%
RPM(Y)	0	−0.28	3.71	0.47	100	15 483	1.91%
RPM(Z)	0	−0.5	2.49	0.84	100	14 244	1.75%
RPM(A)	0	−0.002	0.14	0.006	26.4	8622	1.06%
RPM(C)	0	−0.18	1.51	0.3	95	14 208	1.75%
RPM(SP axis)	0	−6.34	13.3	17.1	100	11 082	1.45%
LOAD(SPINDLE)	0	−2	0.97	3.32	90.9	11 074	1.36%
RPM(SPINDLE)	−3001.5	−606.9	292.1	1035.7	12 045.9	4083	0.50%
SPINDLE OVERRIDE	0	0	74.5	160	100	0	0%
RAPID OVERRIDE	0	−25	76.9	175	120	0	0%

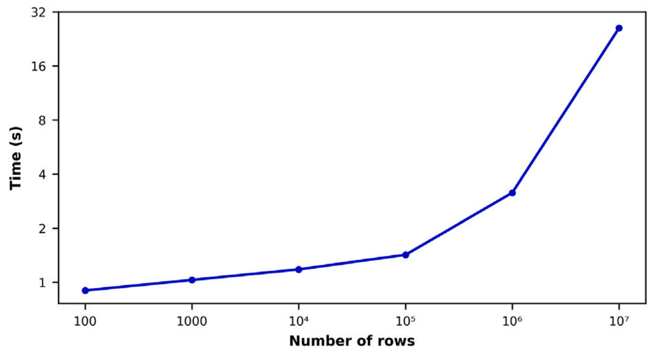

Fig. 12. Elapsed time to calculate IQR and limits.

We also want to note that the variables whose minimum value is 0 have a lower limit lower than 0. This is due to the way that IQR is computed and, in these cases, the detected outliers are values that fall between the upper limit and the maximum value, which is the expected behavior in those cases.

Additionally, we present the time required by the IQR technique to compute the limits for the variables of a given dataset, a step required to use IQR afterwards. For this end, we created trimmed down versions of the dataset described at the beginning of this section. It should be noted that this is a process conducted in the start-up of the software tools, not required as part of the online outlier detection pipeline.

Results are shown in Fig. 12. It is observed that in the largest case, with a dataset of 10M rows, it takes 26 s to compute the limits needed by IQR. In a scenario where a whole re-calculation of the limits would be needed, this is considered to be a reasonable amount of time. If a faster recalculation of the limits would be needed, processing a 1M row dataset takes 3.15 s, which is significantly faster but it would lead to less accurate results.

6. Conclusions

This work has presented a software platform to monitor machining processes in the context of aeronautical manufacturing. This platform is built using open source tools, mostly from the Apache ecosystem, and runs on Linux virtual machines. The tools are arranged as a pipeline: data is captured from the manufacturing machine, gets processed, then stored and finally visualized in a dashboard in real time.

We conducted some tests to evaluate the performance of the pipeline using a real machining centre located at the CFAA in Zamudio, Spain. These tests revealed that the main bottleneck was the data capture stage: even if the platform was configured to obtain data at maximum speed, the communication with the machine introduced latencies which had an impact in the performance, specially when trying to retrieve batches of data with significant sizes.

We also did some additional assessment using a simulator to assess the scalability of the platform. This simulator sent data to the pipeline, as a replacement of the milling centre, in order to test the performance of the software tools with a different input rate. These tests confirmed a good scalability of the platform, specially of the Spark stage, which conducted some processing to detect outliers in the monitored data.

This work has many open lines of future work, but we will lay the three most relevant ones. The first one is reducing the time taken to fetch data from the OPC server. To that end, the intention is to replace OPC with *Profibus* or *Profinet*. These two are industrial communication and control protocols that provide much better latencies than OPC, but with increased control complexities.

The second line of work is to replace or complement the IQR with an AI technique that provides prediction capabilities. Machine Learning algorithms could be used to conduct time series analysis and foresee tool wear or malfunction. A step further away would be to use these algorithms to create a Digital Twin which provides real time diagnosis and forecasting.

The third aspect involves implementing variable frequency monitoring based on data relevance. This means adjusting the monitoring frequency dynamically, especially for factors like spindle speed, to ensure optimal performance in different scenarios. For example, a frequency of 1 s could be sufficient when the spindle speed is 0 but, if a significant change was detected, like exceeding 1 RPM, the system would transition to a 0.1 s interval. This adaptive approach would not only optimize real-time performance but also significantly improve scalability.

CRediT authorship contribution statement

Endika Tapia: Conceptualization, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Unai Lopez-Novoa:** Conceptualization, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing. **Leonardo Sastoque-Pinilla:** Conceptualization, Supervision, Validation, Writing – review & editing. **Luis Norberto López-de-Lacalle:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

The authors are grateful to Ibarmia™ Innovatek, S.L.U. for providing the industrial case study, and specifically Eng. Aitor Ginto for his support in this work. We would like to thank Iñigo Angulo and Andoni Angulo from Zylk for the aid to configure their custom NiFi *processors*. We want also to acknowledge the support received from the Artificial Intelligent Manufacturing for Sustainability Unit (AIMS) of the University of the Basque Country. This work was in part supported by the Smart Networks for Everything (SN4E) experimental facility, funded by European Union NextGeneration EU.

Funding: This work is co-financed by InterQ project that has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 958357, and it is an initiative of the Factories-of-the-Future (FoF) Public Private Partnership. Thanks are addressed to MCIN/ AEI/10.13039/501100011033/ and European Union NextGeneration EU/PRTR (Proyectos de Transición Ecológica *y* Transición Digital). Thanks are also addressed to Basque Government, Spain for the support of University research groups, IT1573-22. Results were analyzed by models developed in Project KK-2022/0065 Lanverso. This work was also partially supported by grants QUOLINK (TED2021-130044B-I00) and Orchesmart-5G (PID2020-117876RB-I00), funded by the Ministry of Science and Innovation.

References

Active Cockpit, 2023. Bosch rexroth ACTIVE cockpit. https://www.boschrexroth.com/en/es/products/product-groups/assembly-technology/topics/manual-production-systems/cip-software-solution-activecockpit/. (Accessed 24 December 2023).

Apache Kafka, 2022. Apache Kafka documentation. https://kafka.apache.org/30/documentation.html. (Accessed 2 November 2022).

Apache NiFi, 2022. Apache NiFi documentation. https://nifi.apache.org/docs/nifi-docs/html/overview.html. (Accessed 13 July 2022).

Apache ZooKeeper, 2022. Apache ZooKeeper documentation. https://zookeeper.apache.org/. (Accessed 15 September 2022).

Ariyaluran Habeeb, R.A., Nasaruddin, F., Gani, A., Targio Hashem, I.A., Ahmed, E., Imran, M., 2019. Real-time big data processing for anomaly detection: A survey. Int. J. Inf. Manage. 45, 289–307, https://doi.org/10.1016/j.ijinfomgt.2018.08.006.

Bruce, P., Bruce, A., Gedeck, P., 2020. Practical Statistics for Data Scientists: 50+ Essential Concepts using R and Python, second ed. O'Reilly Media.

Canizo, M., Conde, A., Charramendieta, S., Miñón, R., Cid-Fuentes, R.G., Onieva, E., 2019. Implementation of a large-scale platform for cyber–physical system real-time monitoring. IEEE Access 7, 52455–52466. http://dx.doi.org/10.1109/ACCESS.2019.2911979.

Cavalieri, S., Chiacchio, F., 2013. Analysis of OPC UA performances. Comput. Stand. Interfaces 36 (1), 165–177. http://dx.doi.org/10.1016/j.csi.2013.06.004.

Celos, 2023. DMG mori celos PC version. https://en.dmgmori.com/products/digitization/celos/pc-version. (Accessed 24 November 2023).

Dalzochio, J., Kunst, R., Pignaton, E., Binotto, A., Sanyal, S., Favilla, J., Barbosa, J., 2020. Machine learning and reasoning for predictive maintenance in industry 4.0: Current status and challenges. Comput. Ind. 123, 103298, https://doi.org/10.1016/j.compind.2020.103298.

del Olmo, A., López de Lacalle, L., Martínez de Pissón, G., Pérez-Salinas, C., Ealo, J., Sastoque, L., Fernandes, M., 2022. Tool wear monitoring of high-speed broaching process with carbide tools to reduce production errors. Mech. Syst. Signal Process. 172, 109003. http://dx.doi.org/10.1016/j.ymssp.2022.109003.

European Parliament, 2023. Regulation 2023/1230/EU for machinery regulation (eu) 2023/1230 of the european parliament and of the council of 14 june 2023 on machinery and repealing directive 2006/42/ec of the european parliament and of the council and council directive 73/361/eec. https://eur-lex.europa.eu/eli/reg/2023/1230/oj. (Accessed 22 November 2023).

Ferrari, P., Flammini, A., Rinaldi, S., Sisinni, E., Maffei, D., Malara, M., 2018. Impact of quality of service on cloud based industrial iot applications with opc ua. Electronics 7 (7), http://dx.doi.org/10.3390/electronics7070109.

Fuller, A., Fan, Z., Day, C., Barlow, C., 2020. Digital twin: Enabling technologies, challenges and open research. IEEE Access 8, 108952–108971. http://dx.doi.org/10.1109/ACCESS.2020.2998358.

Givehchi, O., Landsdorf, K., Simoens, P., Colombo, A.W., 2017. Interoperability for industrial cyber–physical systems: An approach for legacy systems. IEEE Trans. Ind. Inform. 13 (6), 3370–3378. http://dx.doi.org/10.1109/TII.2017.2740434.

Grafana, 2022. Grafana for time series. https://grafana.com/docs/grafana/latest/basics/timeseries/. (Accessed 6 March 2022).

Ibarmia THR 16, 2022. Ibarmia THR 16 website. https://www.ibarmia.com/en/machining-centres/t-series/t-multiprocess/. (Accessed 23 November 2022).

InfluxDB, 2022. InfluxDB documentation. https://docs.influxdata.com/influxdb/v2.1/. (Accessed 14 November 2022).

Ji, W., Yin, S., Wang, L., 2019. A big data analytics based machining optimisation approach. J. Intell. Manuf. 30 (3), 1483–1495. http://dx.doi.org/10.1007/s10845-018-1440-9.

Karau, H., Konwinski, A., Wendell, P., Zaharia, M., 2015. Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, Inc.

Landi, L., Chiavatti, N., Grilli, L., Preteni, M., et al., 2020. Configurable monitoring of machine tools status in smart factories. In: Proceedings of the 30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference. Research Publishing, Singapore, pp. 3584–3591.

Linksmart, 2022. Linksmart's nifi *processors* for OPC-UA. https://github.com/linksmart/nifi-opc-ua-bundles. (Accessed 21 September 2022).

Moldovan, D., Anghel, I., Cioara, T., Salomie, I., 2021. Apache Spark for Digitalization, Analysis and Optimization of Discrete Manufacturing Processes. Springer International Publishing, Cham, pp. 37–57. http://dx.doi.org/10.1007/978-3-030-38836-2_2.

Qi, Q., Tao, F., 2018. Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison. IEEE Access 6, 3585–3593. http://dx.doi.org/10.1109/ACCESS.2018.2793265.

Rodríguez, A., Calleja, A., de Lacalle, L.L., Pereira, O., Rubio-Mateos, A., Rodríguez, G., 2021. Drilling of cfrp-ti6al4v stacks using co2-cryogenic cooling. J. Manuf. Process. 64, 58–66. http://dx.doi.org/10.1016/j.jmapro.2021.01.018.

Sasiain, J., Sanz, A., Astorga, J., Jacob, E., 2020. Towards flexible integration of 5 g and iiot technologies in industry 4.0: A practical use case. Appl. Sci. 10 (21), 7670. http://dx.doi.org/10.3390/app10217670.

Sendino, S., Gardon, M., Lartategui, F., Martinez, S., Lamikiz, A., 2020. The effect of the laser incidence angle in the surface of l-pbf processed parts. Coatings 10 (11), 1024. http://dx.doi.org/10.3390/coatings10111024.

Shi, X.-L., Sun, W.-T., Song, J., 2021. Design and implementation of real-time monitoring system for multiple machine tools. Procedia Comput. Sci. 183, 274–280. http://dx.doi.org/10.1016/j.procs.2021.02.059.

Tapia, E., Sastoque-Pinilla, L., Lopez-Novoa, U., Bediaga, I., López de Lacalle, N., 2023. Assessing industrial communication protocols to bridge the gap between machine tools and software monitoring. Sensors 23 (12), http://dx.doi.org/10.3390/s23125694.

Vinutha, H.P., Poornima, B., Sagar, B.M., 2018. Detection of outliers using interquartile range technique from intrusion dataset. In: Satapathy, S.C., Tavares, J.M.R., Bhateja, V., Mohanty, J.R. (Eds.), Information and Decision Sciences. Springer Singapore, Singapore, pp. 511–518.

Wang, Y., Kang, X., Chen, Z., 2022a. A survey of digital twin techniques in smart manufacturing and management of energy applications. Green Energy Intell. Transp. 1 (2), 100014, https://doi.org/10.1016/j.geits.2022.100014.

Wang, J., Sánchez, J., Iturrioz, J., Ayesta, I., 2019. Artificial intelligence for advanced non-conventional machining processes. Procedia Manuf. 41, 453–459. http://dx.doi.org/10.1016/j.promfg.2019.09.032.

Wang, J., Xu, C., Zhang, J., Zhong, R., 2022b. Big data analytics for intelligent manufacturing systems: A review. J. Manuf. Syst. 62, 738–752, https://doi.org/10.1016/j.jmsy.2021.03.005.

Zylk, 2023. Zylk's custom *processors* for OPC-UA. https://github.com/zylklab/nifi-opc-ua-bundles. (Accessed 12 February 2023).