UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DIONEI MIODUTZKI

DETECÇÃO DE OUTLIERS USANDO DATA STREAM COM CONTEXTUALIZAÇÃO DE FALHAS ORIENTADA POR ONTOLOGIA NA INDÚSTRIA 4.0

CURITIBA

2022

DIONEI MIODUTZKI

DETECÇÃO DE OUTLIERS USANDO DATA STREAM COM CONTEXTUALIZAÇÃO DE FALHAS ORIENTADA POR ONTOLOGIA NA INDÚSTRIA 4.0

Data stream outlier detection with ontology-driven fault contextualization in the Industry 4.0

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre em Computação Aplicada do Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Luiz Celso Gomes Jr

CURITIBA

2022

DIONEI MIODUTZKI

DETECÇÃO DE OUTLIERS USANDO DATA STREAM COM CONTEXTUALIZAÇÃO DE FALHAS ORIENTADA POR ONTOLOGIA NA INDÚSTRIA 4.0

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Computação Aplicada da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Sistemas Computacionais.

Data de aprovação: 19 de Agosto de 2022

Dedicated to Ana, Lunna, Louis, Ling and
Lango, for their motivation and understanding
the countless hours of study and work.

ACKNOWLEDGEMENTS

I would like to thank all the professors and staff of the Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná, for the professionalism and attention received throughout the course, and especially Professor Dr. Luiz Celso Gomes Jr, for his dedication, patience and cordiality throughout the development of this project. I would also like to thank Ricardo Spancerski for believing in the potential and viability of this project.

RESUMO

A detecção de outliers é importante em diversos setores da economia, na academia e no governo. No setor industrial, essas técnicas permitem identificar com rapidez e precisão falhas de equipamentos, defeitos de produtos e riscos de segurança. A evolução da Indústria 4.0, no entanto, está trazendo desafios antes incomuns na área. O grande número de dados gerados constantemente por uma infinidade de sensores representa um desafio de processamento e pode levar à identificação de um grande número de outliers simultaneamente. A escala e a complexidade desse cenário retardam o processo de solução de problemas, atrasando a identificação da origem da falha e aumentando os custos e o tempo de inatividade. Este trabalho apresenta uma solução que aborda o problema em duas frentes: (i) processamento distribuído de data streams para detecção de outliers; e (ii) contextualização baseada em ontologia dos outliers detectados. Nossa proposta suporta a tomada de decisão em um cenário de falha generalizada, onde existem vários outliers detectados em um conjunto de equipamentos com dependências conhecidas entre eles. As dependências são representadas usando ontologias, como forma de fornecer uma interpretação clara e facilitada ao usuário. Um mecanismo de inferência implementado como banco de dados de grafos é responsável por identificar as causas mais prováveis da falha. Testes de desempenho demonstram a escalabilidade de nossa implementação.

Palavras-chave: outlier; data stream; ontologia; big data.

ABSTRACT

Outlier detection is important in several sectors of the economy, the academy and the government. In the industrial sector, these techniques make it possible to quickly and accurately identify equipment failures, product defects and safety risks. The evolution of Industry 4.0, however, is bringing challenges previously uncommon in the area. The large number of data constantly generated by a multitude of sensors represents a processing challenge and can ultimately lead to the identification of a large number of outliers simultaneously. The scale and complexity of this scenario slow the troubleshooting process, delaying the identification of the source of the fault and increasing costs and downtime. This work presents a solution that tackles the problem in two fronts: (i) distributed processing of data streams for outlier detectiong; and (ii) ontology-based contextualization of the detected outliers. Our proposal supports decision-making in a widespread failure scenario, where there are multiple outliers detected in a set of equipment with known dependencies between them. Dependencies are represented using ontologies, as a way to provide a clear and user-facilitated interpretation. An inference engine implemented as a graph database is responsible for identifying the most probable causes of the failure. Performance tests demonstrate the scalability of our implementation.

Keywords: outlier; data stream; ontology; big data.

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

LIST OF SOURCE CODE

CONTENTS

1 INTRODUCTION

In general terms, an anomaly, also called an outlier, is discordant information, an exception (CHANDOLA; BANERJEE; KUMAR, 2009). Outliers can create problems if not detected properly. Outlier detection techniques are used in several sectors of the economy: to detect failures in aeronautical sensors, detect credit card fraud and identify early signs of latent disease, among others. However, modern society imposes new challenges to the problem of detecting outliers. New technologies and devices are launched every day, and the industrial sector is at the forefront of this evolution. The advent of Industry 4.0 is equipping plants with sensors and automation devices to monitor and control production processes with high precision.

All these sensors, systems and software generate data in a rate that may turn impossible to be analysed using a traditional software architecture. It is common that during an outlier detection, certain steps need more time to be computed than others. When this happens, the entire software has to wait for the analysis to be finished, keeping most of the hardware idle. One possible solution is to use a *data stream* – a type of software designed to receive and analyze data in real (or near real) time, where each processing step is a module developed individually. *Streams* can be handled in parallel processing environments, that is, certain steps can be executed by more than one processing core, or even more than one computer at the same time.

Such amount of data, however, ends up resulting in an also expressive number of detected outliers, up to the point of making it impractical for them to be analyzed manually. Typical examples of widespread failures are natural events such as a lightning strike, where a single physical incident can prompt many outliers in several equipment in a given geographic region. Another scenario, harder to interpret, is when an event causes the occurrence of cascading outliers. A failure causes an outlier observed by a sensor, and this failure is propagated to other equipment that depends on the first. To make those outliers easier to interpret and use resources efficiently, it is crucial to identify these cases and present to the user a higher-level knowledge related to the origin of the failure.

To verify such a volume of data and identify the sources of the failures found is a challenging task. Finding a flexible solution that allows representing the most diverse industrial setups and their relationships in a clear and objective way is difficult in itself. In this work we propose and architecture to address these issues (Chapter 3), based on a real industrial scenario (Section 3.1). In our proposal, we first tackle the volume of data using data streams to detect the outliers in a efficient and quick manner (Section 4.2). Then, the relationships between the devices are described in an ontology, using the OWL language (Section 4.3). The ontology simplifies knowledge management, especially in such complex scenarios of Industry 4.0. The ontology is stored in a graph database, which also receives and stores the detected outliers (Section 4.4). A graph query language is used to perform the inferences needed to determine the probable causes of the failure. The proposal was tested on large ontologies simulating complex industrial scenarios (Section 4.5). Test results demonstrate the scalability and feasibility of the proposal.

2 FUNDAMENTALS AND RELATED WORK

The industrial sector is passing through a new technological revolution. While Industry 4.0 is changing fast how industries are organized and managed, bringing new technologies and methods to optimize processes, it also create challenges on how to deal with the plethora of tools, systems and data (POPKOVA; RAGULINA; BOGOVIZ, 2019).

2.1 Sensors, IoT and Industry 4.0

Although there is no clear definition of Industry 4.0, it is characterized by a list of technologies and scenarios (HERMANN; PENTEK; OTTO, 2016): Interoperability, with companies, systems and humans connected to each other; virtualization, where systems monitor physical processes through sensors and use this data in models and simulations; decentralization, as the large number of connected systems makes centralized control impractical; real-time capability, as data analysis and reaction must be immediate; service orientation, to allow easier use of all resources, they are encapsulated in large sets; and modularity, which allows quick adjustments, adaptations and scalability.

The concept of Industry 4.0 is deeply related to IoT. As Wang *et al.* (2022) have written, the Internet of Things (IoT) is an extended and expanded system network based on the Internet, built to achieve real-time interaction among things, machines and humans through various advanced technological means. IoT is bigger than Industry 4.0 and has applications in diverse fields, such as health, sports and home automation. The concepts and the technologies, either hardware, software and protocols used are complementary to those of Industry 4.0.

This increasing amount of sensors leads to an also increasing number of outliers generated in those new industrial environments. According to Gaddam *et al.* (2020), sensors used for IoT applications are often installed in harsh environments and are typically made of inexpensive electronic components, thus it is hard to guarantee a correct operation, free of malfunctions.

2.2 Outlier detection

Outlier detection seeks to identify patterns of behavior in the data and use this to classify which data does not belong to this pattern. As defined by (HAWKINS, 1980), an outlier is an observation that deviates so much from others that it creates suspicion it was generated by a different mechanism. This type of technique is used in the most varied sectors of the industry: to detect failures in aeronautical sensors, detect credit card fraud, identify early signs of latent disease, etc.

A different approach to obtain more details about an outlier is using outlying aspect mining (OAM). OAM complements traditional outlier detection because by identifying characteristics that make the observation different from other objects (DUAN *et al.*, 2015). This technique is useful in understanding why a single data was detected as an outlier, however, it is not designed to detect the source of a group of outliers (one of the focus of our proposal).

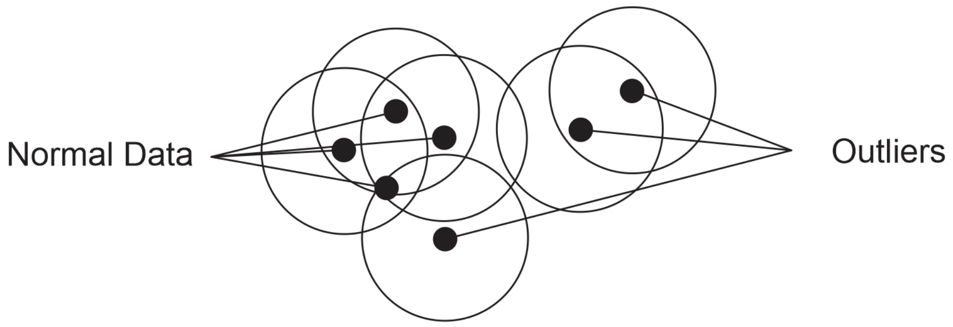
2.3 Detection of outliers in stream

Data streams are continuous streams of data that must be analyzed in real time (or near real time) at the moment they are received. Several outlier detection problems fit into this context, implying new architectural challenges. Take as an example a vehicle tracker that constantly sends data to a server about the location and status of the vehicle. The server must analyze the received data constantly as it is received, detecting outliers that may represent a defect in the car, in the sensor, or even a stolen car.

The solutions to detect outliers in these streams almost always involve the use of data windows (YU *et al.*, 2014). In this approach, a sample size is defined that will be used for data analysis. The sample (referred to as *window*) can have different configurations of sizes and rules to which values to consider.

Among the most straightforward implementations of outlier detection in data streams are the distance-based algorithms (SALEHI *et al.*, 2016). Figure 1 shows an example of how these implementations work: a radius is defined in which other data will be considered as part of the neighborhood of the analyzed sample. The data that do not have a minimum number of elements in their neighborhood are considered outliers.

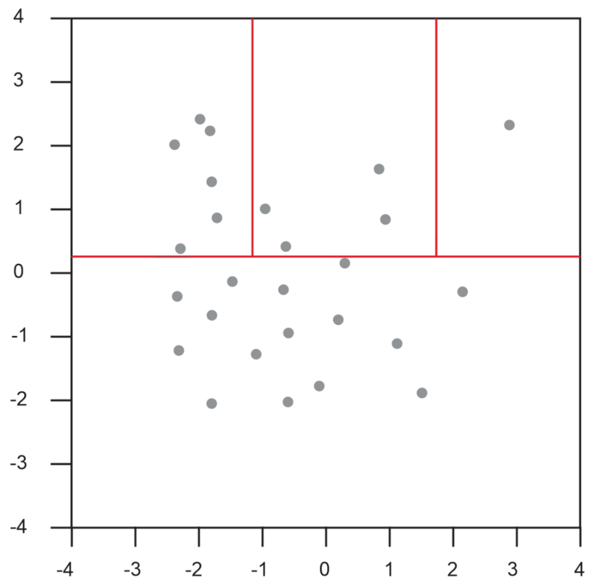
Figure 1 – Example of distance-based outlier detection algorithm



Source: The author (2022).

In this work, however, a more robust technique was used. The algorithm *Isolation Forest* randomly selects on of the attributes of the data, and creates a tree by dividing the data into a random value between its the smallest and the biggest values (DING; FEI, 2013). Figure 2 shows an example of how it works. The tree continues to be divided until every data sample is alone on one of its branches. The algorithm then ranks the changes of each sample to be an outlier, considering how far they are in that tree. It is up to the developer to define at what score the sample will be considered an outlier, or give this option for the user to decide.

Figure 2 – Example of Isolation Forest algorithm



Source: The author (2022).

The outlier detection is used as the first step of our architecture, identifying anomalies in the data generated by each equipment individually. The high volume of data is managed by using a Data Stream to do the detection using more than one computer at the same time. This may lead to a large number of outliers generated as a consequence of the same real-world event. The inference over the detected outliers is our main contribution.

2.4 Ontologies and OWL

Knowledge representation is a challenge in Industry 4.0. In the past, it was usual to represent entities, their properties and relationships directly on the code, using object-oriented programming. In Industry 4.0, however, this is not a viable method. New types of devices are added to the network constantly, and their relationship changes accordingly to what is needed in the production line. To deal with the problem of heterogeneity, ontologies emerge and play an important role to integrate seamlessly business processes with technical processes (CHENG *et al.*, 2016).

According to Guarino (1998), in AI, an ontology usually refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. In this scenario, an ontology is a document that describes entities that are part of a system, listing their properties and how they are related to each other. In an analogy with object-oriented programming, the ontology would be the classes, with their hierarchies and properties.

OWL (Web Ontology Language) is an ontology language for the Semantic Web with formally defined meanings (GROUP, 2012). An OWL file contains an ontology with classes, properties, individuals and data values, allowing entities to share the entire ontology easily between different software and platforms.

In this work, an ontology was used to describe the dependencies (functional and geographic) of industrial equipment in large industrial settings, following Industry 4.0 practices. This makes the solution generic enough to be easily adapted to different industrial plants, or even to be used on environments other than industrial ones.

2.5 Ontologies in Outlier Detection Tasks

Ontologies have been used in outlier detection scenarios. However, the focus is usually on detecting anomalies in the data relationships themselves. In other words, finding data that are related in ways they were not supposed to. A typical use for ontologies on outlier detection is to detect fraud in banking systems. Ramaki, Asgari e Atani (2012) used an ontology to describe how each piece of information in a credit operation relates to each other, making it easier to create a model of an expected behavior. This model is then used to detect transactions that do not follow that expected behavior.

Fleischhacker *et al.* (2014) demonstrate that ontologies can also be used to detect outliers in numeric datasets. Numeric values present in Wikipedia articles are extracted from the texts (countries' populations for example). An ontology describes how those values are related to each other (what is being counted, country or other geographic entity relevant, the year the data was collected, etc). Then, all numeric values present in the texts are compared with the inferred ontology, and those that are dissonant are quickly identified for verification.

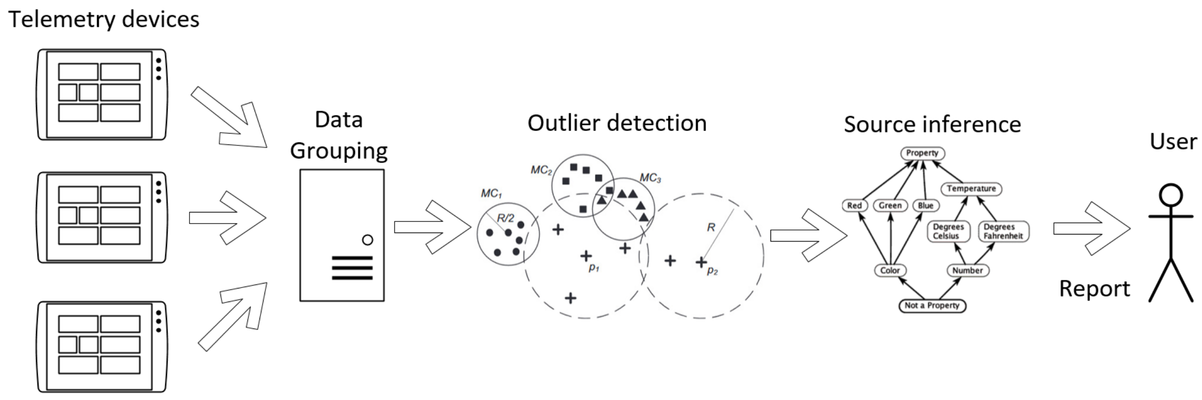
In our work, ontologies were not used directly to detect outliers. Instead, they are used to identify, among outliers that were previously detected with traditional methods, the most probable source of the real-world event.

3 PROPOSED ARCHITECTURE

The proposed architecture consists of two main processing steps: (i) the outlier detection and (ii) the source inference. As shown in Figure 3, data are collected by telemetry devices and are sent to a server that performs traditional outlier detection, independently, in each of the sensors that feed the system. In this step, there is a detection model to analyze each sensor individually, detecting outliers in its regular operation. Given the high volume of data and processing demands, our solution uses a distributed stream processing architecture.

Once the outlier detection is finished, data and the detected outliers are moved to the second step, where data from all sensors are aggregated for each time window. Therefore, for a given time window (configurable), a set of detected outliers represents the current state of the facility. The detected outliers are then associated with ontologies representing functional and spatial dependencies among devices. The inference engine determines the probable cause of the failure based on the ontologies and the outliers. Finally, the outliers identified as the sources of the anomalies are presented to the user in a report.

Figure 3 – Proposed architecture for outlier detection and fault contextualization.



Source: The author (2022).

The main goal of using ontologies in this architecture is to streamline knowledge representation, making it easier to maintain and modify the dependencies among devices. In addition, this allows the use of relationship names directly in queries, making them much more descriptive and easier to understand. Finally, as the relationships are only described in the ontology, the solution becomes more flexible and easily adaptable to other scenarios, without the need to modify the source code of the application. In a real world scenario, the ontology would be created with the help of people who knows the domain and can define how all the devices interact with themselves.

3.1 Case Study

The case used to test and explain the system is based on a real scenario for water resources management in an industrial plant. The data are related to the production of water in deep wells, pipelines, reservoirs, distribution systems and various industrial equipment that consume and manage the water. Those facilities are operated by different companies in various Brazilian cities.

Each location of interest has equipment that collects data continuously with a fixed sampling rate. The data are sent on a recurring basis to a server to be processed and stored. Each installation can have one or more variables, according to the equipment being monitored. Table 1 shows an example of data collected from two of those equipment, with a column *Device* identifying the source. *Level* is the well water dept, *Flow* is how much water is being pumped per hour, *Volume* is how much water was pumped during the period, and *Time* is how many minutes the water was pumped. The full scenario includes several types of devices with multiple monitored variables.

Table 1 – Data sample

Timestamp	Device	Level (m^3)	Flow (m^3/h)	Volume (m^3)	Time (min)
2021-06-01 10:00	Well 1	250.6	23.1	25685.4	100
2021-06-01 10:00	Well 2	123.4	0	57483.2	5
2021-06-01 10:01	Well 1	252.1	23.5	25687.8	101
2021-06-01 10:01	Well 2	124.4	0	57483.2	5
2021-06-01 10:02	Well 1	252.4	23.7	25692.3	102
2021-06-01 10:03	Well 2	124.3	0	57483.2	5

Source: The author (2022).

Among the possible scenarios where multiple outliers caused by the same failure can occur in this environment, the most directly related to this particular industrial architecture is the propagation of anomalies through the pipelines. If there is a leak in the piping right before the location where equipment is measuring the flow of the water, this equipment and possibly all the others that are supplied by it will generate outliers.

On the other hand, external factors, such as a power failure, for example, may render an entire industrial plant offline, causing all its equipment to generate outliers at the same time. The goal of our proposed system is to automatically identify, in a knowledge representation sense, the probable cause in such multi-outlier scenarios.

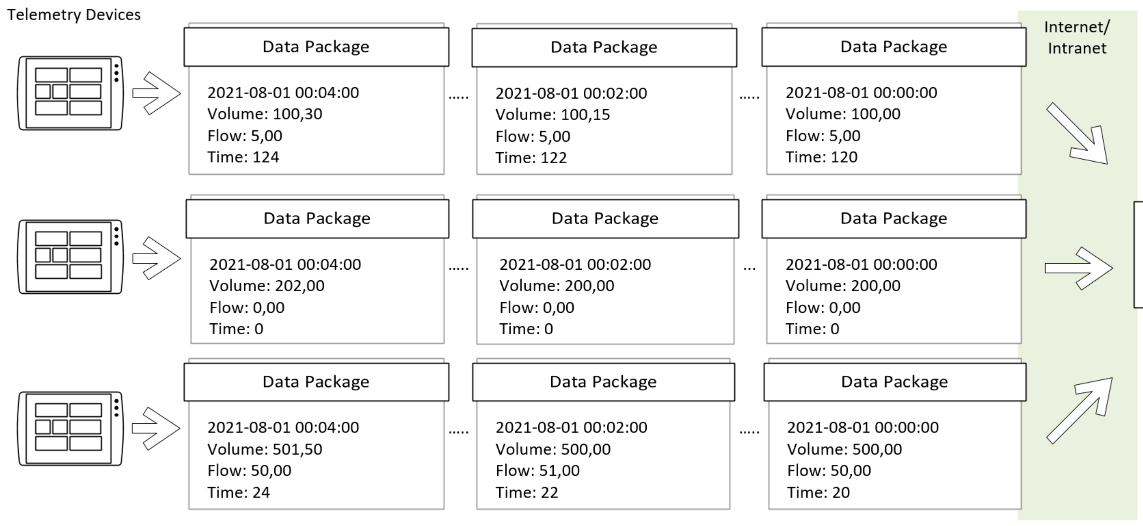
4 IMPLEMENTATION AND TESTS

To test the viability of the proposed solution, both the data stream for outlier detection and the ontology based source inference were implemented and tested accordingly to the proposed architecture

4.1 Data collection and processing

Figure 4 shows an example of how the developed data flow works. In the figure, the first block (top left) represents the readings from a set of sensors (volume, flow and time) from an equipment at timestamp 2021-08-01 00:04:00). The equipment collect data in regular intervals and transmit those data packages to the server in the order they were generated (first in, first out). Each package contains a full set of all variables monitored by that device, and all devices collect data on the same timestamps. Data are then transmitted to the MQTT broker (*Message Queuing Telemetry Transport*, a very popular message protocol for telemetry and sensors communication) by using a mobile internet connection.

Figure 4 – Data flow from the telemetry devices.



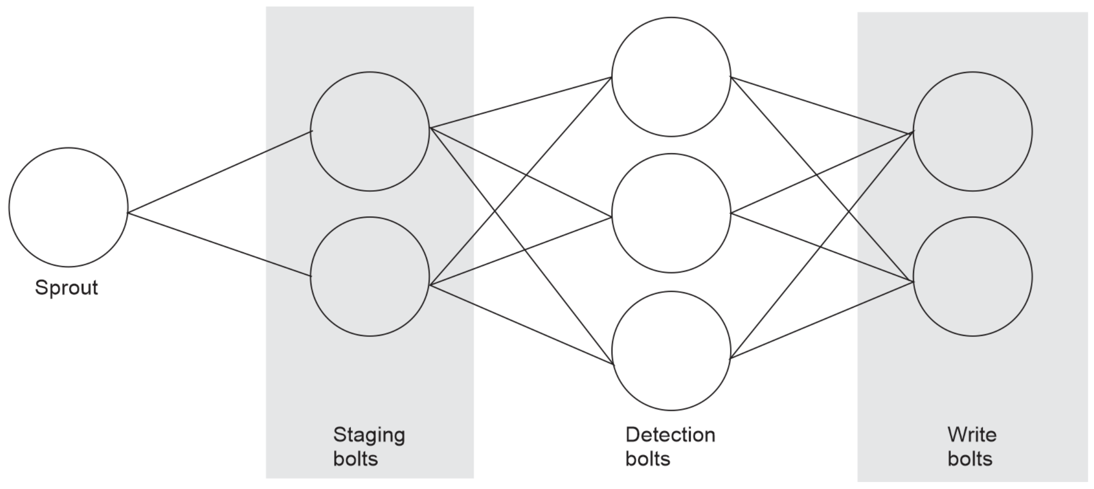
Source: The author (2022).

4.2 Distributed Outlier Detection

To process the stream of received data, we rely on a distributed realtime computing platform, the Apache Storm. The platform allows the usage of any programming language, and makes it easy to implement real-time data processing, including data streams, ensuring fault tolerance and allowing the use of clusters for distributed processing. Apache Storm uses the concept of *topology* to represent the complete set of processes that are be managed by the platform, including instructions on how data should travel between its layers; *sprouts* are the data sources that will be processed by the topology - where the processing actually starts; and *bolts* are the nodes that actually perform the necessary tasks. Both sprouts and bolts are, in a general way, independent softwares that run simultaneously.

Figure 5 presents a diagram demonstrating the structure of the topology developed. The *sprout* is a software that connects to the MQTT daemon and listens for new data packages to arrive. All sensors transmit data to the same MQTT daemon and topic, to which the *sprout* is subscribed. Once received, they are feed to 2 staging *bolts* where the data packets are split into individual registers. The data is then sent to 3 outlier detection *bolts*, where the outlier detection code takes place. Finally, the data is delivered to 2 write *bolts*, to send the data to be processed on the next step.

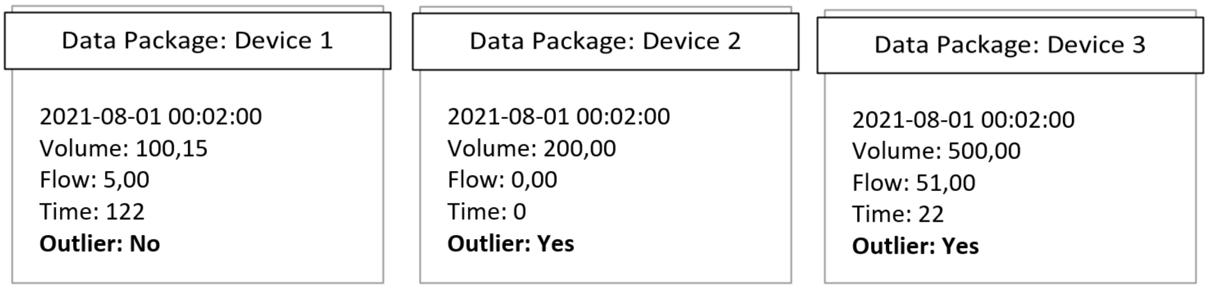
Figure 5 – Proposed topology for outlier detection



Source: The author (2022).

The possibility of having multiple outlier detection *bolts* running at the same time create issues in how to assign the data of each device to the *bolt* that contains its corresponding outlier detection model. To make this possible, the proposed topology links the *bolt* chosen to execute the outlier detection *bolt* according to the value of the *Well* field, that is, data from the same well will always be sent to the same running *bolt*. The second *bolt* is the one who effectively detects outliers; it stores in memory the data models of all installations being analysed, selecting the model that will be used according to the value of the variable *Well* in the tuple received. At the end of processing, the *bolt* adds an boolean value to the tuple indicating if the sample is an outlier or not. The third and final *bolt* (write *bolt*) takes the tuple containing all the original data and sends it to be analysed in the source inference step. Figure 6 shows and example of the data after the outliers are detected.

Figure 6 – Data packages after the outlier detection.



Data Package: Device 1

2021-08-01 00:02:00
Volume: 100,15
Flow: 5,00
Time: 122
Outlier: No

Data Package: Device 2

2021-08-01 00:02:00
Volume: 200,00
Flow: 0,00
Time: 0
Outlier: Yes

Data Package: Device 3

2021-08-01 00:02:00
Volume: 500,00
Flow: 51,00
Time: 22
Outlier: Yes

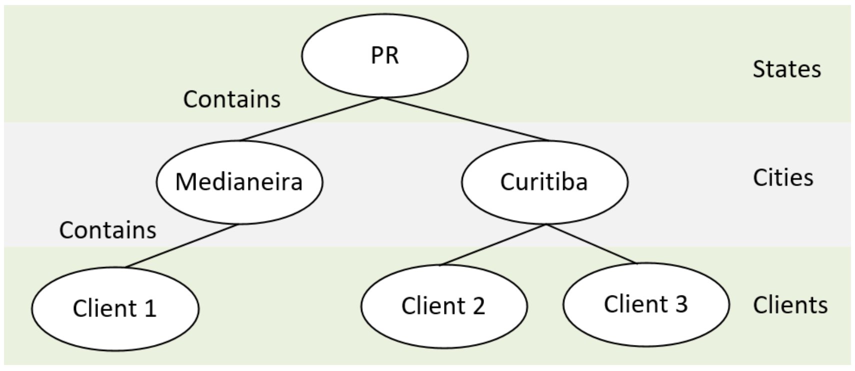
Source: The author (2022).

4.3 Knowledge representation

An important aspect taken into account during the project was knowledge representation. The goal is to centralize the knowledge about the dependencies between devices, where they are and how they relate to each other.

The ontology that materializes this knowledge contains two types of relationships. The first represents the geographic organization of the equipment (in which company, sector, city and state they are contained), as is shown in Figure 7. The ontology contains information about where the sensor is geographically located, grouping sensors at different levels of hierarchy, thus allowing generalized faults located in a specific region (power supply failure for example) to be classified as unique failures.

Figure 7 – An ontology representing geographic distribution of devices.

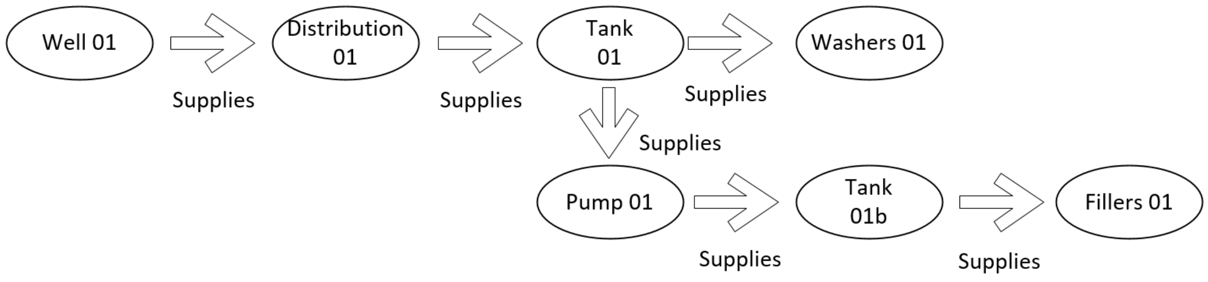


Source: The author (2022).

The equipment is distributed at various geographic levels. In the study case, each equipment is part of a sector, which is owned by a client. The client is a single industrial unit, located in a city, that is part of a state.

The second type of relationship represents the dependency between the equipment, that is, a malfunction in one equipment affects others down the line. Figure 8 shows an example of this dependency, where water is collected on *Well 01*, sent through a pipe on *Distribution 01*, then stored on *Tank 01*. From there, it is sent to either *Washers 01*, or to be stored on *Tank 01b* using *Pump 01*, from where it can be sent to *Fillers 01*. One failure on *Tank 01*, for example, may also cause issues on *Pump 01* and *Washers 01*, and as a consequence all equipment that is supplied by them.

Figure 8 – An ontology representing dependency between devices.



Source: The author (2022).

All this information, including the definition of what a sensor is, what information it contains, how it can relate to other sensors, and how it can be contained within a geographic region, is saved on an OWL file. The same file also contains details about the spatial relationship between those elements. Finally, the file also contains all the elements instances of the project - the sensors and spatial entities.

4.4 Inference

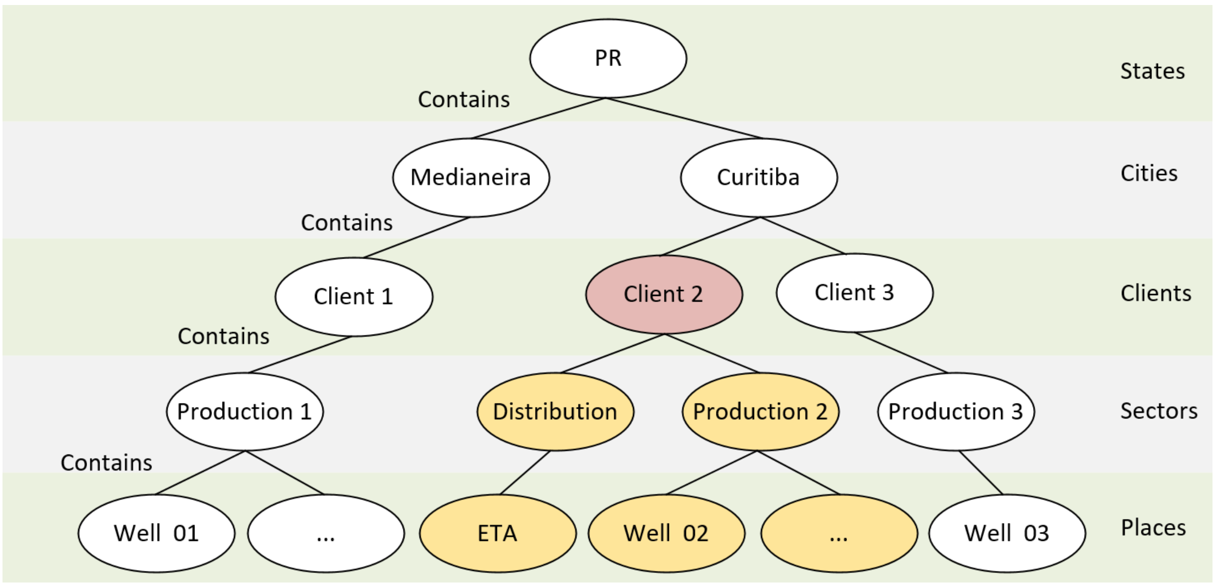
A central step of our implementation, the inference is the moment in which the outlier that originated the sequence of outliers is identified. Once the ontology is built with all the equipment, data, and its respective outliers, the information is loaded into a database to make inference easier and faster.

The solution was implemented using the graph database Neo4j with the Neosemantics ontology library. While importing the OWL ontology, its classes are transformed on nodes and the class name and the information of being or not an outlier are saved as labels. The detected outliers are mapped to the ontology in each analyzed timestamp, then two queries in Cypher language are used to detect the regional and dependency outliers sources, as discussed next.

4.4.1 Geographic Inference

In this implementation, a region is considered an outlier if all devices it contains are outliers in a given time window. This rule is applied recursively, so when all devices of all sets of a client are outliers, the client is considered an outlier, and so on. This permits to quickly identify anomalies from external agents, like issues with the power supply network, for example. Figure 9 shows this detection being activated.

Figure 9 – Example of data failure in an entire client.



Source: The author (2022).

Listing 1 shows the query that does the inference on regional dependency. First, it matches all region entities, then it selects only those that contain equipment and that all equipment has outliers.

Source Code 1 – Geographic inference query.

```
match (n:region)
    where  (n)-[:contains]->(:equipment {outlier: true} )
    and not (n)-[:contains]->(:equipment {outlier: false})
    return n
```

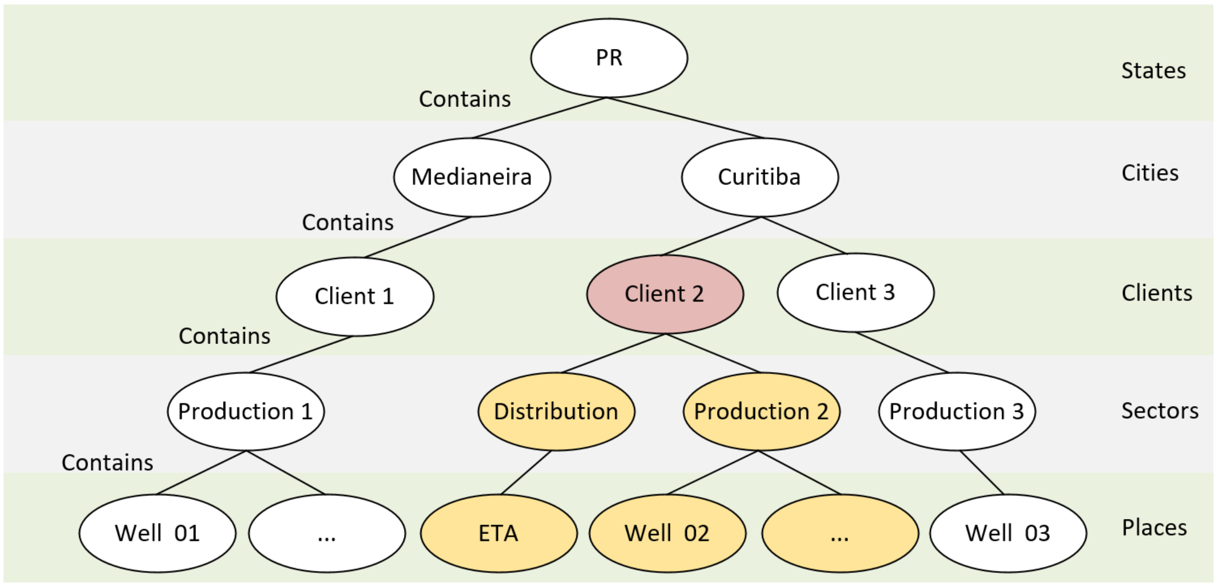
Source: The author (2022).

4.4.2 Dependency Inference

In our use case, the dependency between the equipment is given according to the flow of water. The points of origin are the wells, where the water is captured, and then each piece of equipment supplies one or more pieces of equipment, until reaching the final point of the distribution network. In the opposite direction, the equipment is supplied by each other.

In this scenario, a physical failure in *Distribution 01*, for example, can cause a variation in the data, and consequently new outliers, in equipment that is supplied by it. Figure 10 demonstrates this example, where the equipment in red is where the actual failure occurred, but the equipment in yellow also showed inconsistent data as a result.

Figure 10 – Example of data failure happening in an entire geographic area.



Source: The author (2022).

Listing 2 shows the query that does the inference of the source of outliers based on the dependency between equipment. First, it matches all equipment entities that are outliers, then it filters out the ones that are supplied by other equipment that is an outlier too. Therefore, only the equipment that is the first in a sequence of outliers is listed.

Source Code 2 – Dependency inference query.

```
match (n:equipment {outlier: true})
    where not (n)-[:supplied]->(:equipment {outlier: true})
    return n
```

Source: The author (2022).

4.5 Tests

Both characteristics of the proposed architecture were tested separately. First, the data stream used to detect outliers was tested for performance and scalability, and then the source inference was tested for both its functionality and performance.

4.5.1 Outlier Detection Data Stream

Once the topology was implemented, different threads combinations were tested, and their results were compared. For this test, an Ubuntu Linux 21.04 virtual machine was used, running in a PC Intel I5-3470s with 4 cores at 2.90Ghz, 1TB SSD disk and 8GB of RAM. The computer was running the hypervisor operational system ESXI, and the virtual machine received 4 virtual cores with 1Ghz each, and 4GB of RAM. To avoid any interference between the tests, no other virtual machine was running on the server during the tests. Between each data sample size change, the virtual machine state was reverted to a snapshot taken at the beginning of the tests, thus assuring the results are not affected by any type of cache.

To simulate using data transmitted in real time, a program was developed to read a log of telemetry records from a database and write these records to the MQTT daemon in an automated way; the MQTT daemon and the data injection software were run on a computer on the same local network as the server under test. The same data sample was used in all tests. Each configuration was tested 3 times under these same conditions and each configuration ran for 10 minutes.

4.5.2 Source Inference

Extensive tests were done to simulate the system operating on industrial plants of various sizes and setups. Testing was done in two steps: The first, focusing on semantics, uses a manually generated ontology, representing a real-world study case. The sample contains 18 equipment, divided into two cities and 3 different clients, and simulates 6 fail events where the first outlier is known. This ontology was used during development to assure the queries were detecting the expected location.

The second test scenario was designed for performance evaluation. A Python script was developed that generates random ontologies using the same equipment and geographic entities, with different numbers of entities and outliers. The script loads the ontologies into the graph database and runs the queries, keeping track of the time it took to process. The script run tests using ontologies with 5000 to 100000 entities, increasing in steps of 5000 entities, and varying the number of outliers between 10% and 70% of the entities. Each test was run 5 times, and the median value was used as a reference. The test was run on the same hardware and with the same precautions than the outlier data stream tests.

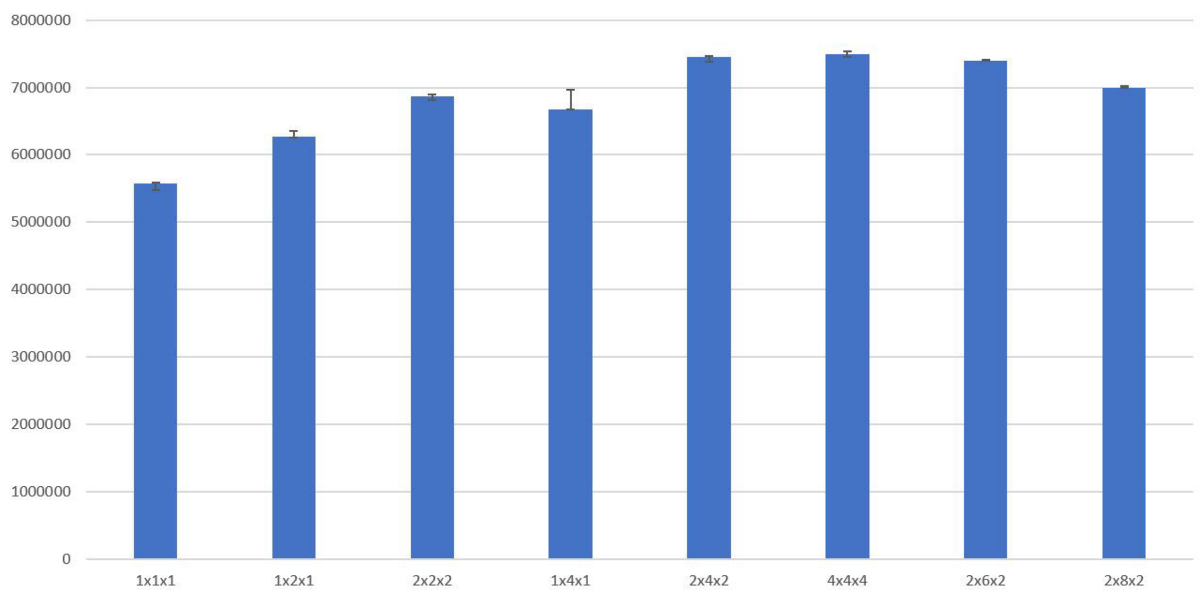
5 RESULTS

This section presents the results of the main tests performed over the implemented architecture. We first describe the tests for the distributed outlier detection phase, then describe the tests for source inference.

5.1 Outlier Detection Data Stream

Graph 1 compares the amount of records processed in each of the tested configurations. For ease of representation, each test is identified by the number of *bolts* in each layer, following the pattern: staging x outlier detection x write.

Graph 1 – Average quantity of verified data samples on each tested configuration.



Source: The author (2022).

The test show a progressive improvement in performance according to the number of concurrent threads running the outlier detection *bolt*. There was also a performance improvement by increasing the staging and write *bolts* running in parallel from 1 to 2. A further improvement from 2 to 4 outlier detection *bolts* showed another significant improvement in processing power, which did not happen when increasing the accessories *bolts* – a possible explanation is that 2 threads for each are sufficient to keep the stream flowing to the 4 outlier detection bolts. As there was no visible improvement, the next tests kept only the configuration of 2 *bolts* for staging and write. When increasing the outlier detection threads to 6 (2 threads more than the cores available in the virtual machine), there was a small performance drop, which was accentuated when increasing the number of threads even further to 8. Instantiating more threads for the same *bolt* than there are cores available on the machine appears to degrade performance, or at the very least show no improvement.

The use of parallel processing, however, proved to be effective in considerably increasing the processing capacity of the hardware, where the configuration with the most optimized performance (2 staging *bolts*, 4 outlier detection *bolts* and 2 write *bolts*), achieved 37% better performance compared to linear execution with a single thread for each activity. Although tests using real distributed clusters were not made, the results suggests that such architecture would keep improving scalability.

5.2 Source Inference

The first test, using the manually written ontology, suggested that the solution could correctly classify the expected location on all simulated fail events. For both dependency and geographic inference, all simulations resulted in the correct most probable source identified.

The results of the performance tests were divided into two, showing the variation in the time for processing the ontologies when the number of entities is increased, and when the number of synthetic outliers is also increased.

5.2.1 Scalability

Graph 2 shows the median result of how much time (in milliseconds) it took to run the regional and dependency inferences. There is a visible but linear increase in the processing time of the regional inference while the size of the ontology is also increased. The time needed to process the dependency inference also increases linearly but at a slower rate. Tendency lines were added for an easier interpretation and suggest consistent linear growth, conclusive to good scalability.

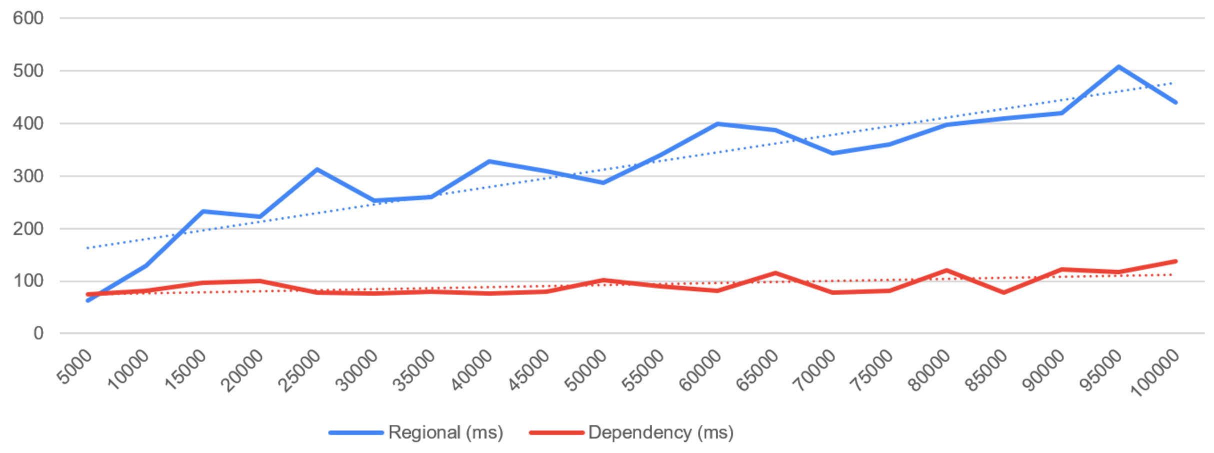
5.2.2 Increasing ontology size

Table 2 – Results increasing ontology size

Outliers	Regional (ms)	Dependency (ms)
10%	297.9806662	80.47699928
30%	312.7410412	81.6538935
50%	365.3517962	77.27694511
70%	347.4471569	76.98905468

Source: The author (2022).

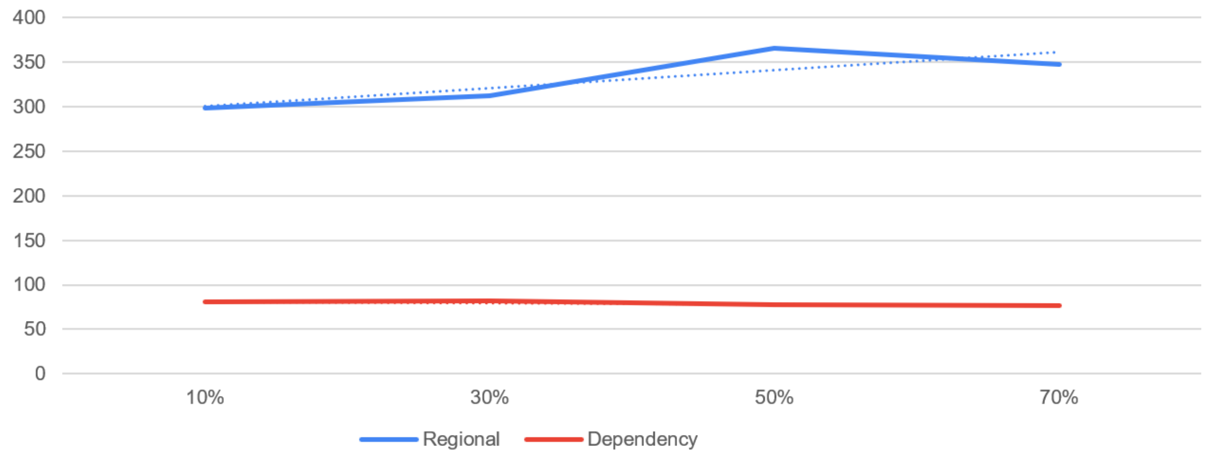
Graph 2 – Time variation while ontology size is increased.



Source: The author (2022).

Table 2 shows how the inference time varies according to the number of synthetic outliers present. The number is the median result between all tested sizes of ontologies. Analyzing this data in Graph 3, there is again a tendency of increase in time for regional inference when more synthetic outliers are added. The dependency inference, in this case, stays stable between the different tests. Again, scalability trends suggest that the proposal is applicable in large industrial settings. In those test, regional inference takes significant more time than dependency inference because all elements are on the same regional graph, while they divided and multiple depedency graphs.

Graph 3 – Time variation while number of outliers is increased.



Source: The author (2022).

6 CONCLUSION

The proposed architecture tackles challenging aspects introduced by Industry 4.0, including the large number of sensors and the complexity of their dependencies. By testing a solution to detect outliers in an efficient way by using data streams, and then combining ontology-based knowledge representation and graph-based inference, our proposal enables timely decision-making in a failure scenario with such challenging settings. As far as we know, this is the first time that ontologies and outlier detection were combined in such a manner.

Our tests demonstrated the feasibility of the proposal. The data streams made it possible to scale up the outlier detection, taking advantage of the available hardware to achieve better performance. The results were also positive for the failure source inference, both in terms of semantics (using manually created ontologies and qualitative analysis of the results) and scalability. The performance tests demonstrated good performance even in the off-the-shelf hardware used. The fact that the processing time increases linearly as both the number of equipment sending data to the data stream and the number of entities in the ontology increases, makes it more predictable in terms of required hardware as the industrial installations expand.

The implementation used generally available languages and software, demonstrating that the solution is effective and usable in a real production environment. Also, all tools used were open source.

In a future work, we plan to test the outlier detection in a distributed computing environment (with multiple servers); add a higher-level language for inference specification, allowing more flexibility in the expansion of the inference capabilities; better define the relationship between real world elements and ontology properties,; experiment with detecting sensor failures when an outlier is not spread to the rest of the devices, and add an aspect mining step to provide further information about the detected source of the failure.

BIBLIOGRAPHY

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: a survey. **ACM Computing Surveys**, 2009.

CHENG, H. *et al.* Manufacturing ontology development based on industry 4.0 demonstration production line. **Third International Conference on Trustworthy Systems and Their Applications**, 2016.

DING, Z.; FEI, M. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. **IFAC Proceedings Volumes**, 2013.

DUAN, L. *et al.* Mining outlying aspects on numeric data. **Data Mining and Knowledge Discovery**, 2015.

FLEISCHHACKER, D. *et al.* Detecting errors in numerical linked data using cross-checked outlier detection. **The Semantic Web - ISWC 2014**, 2014.

GADDAM, A. *et al.* Detecting sensor faults, anomalies and outliers in the internet of things: A survey on the challenges and solutions. **Electronics**, 2020.

GROUP, W. O. W. **OWL 2 Web Ontology Language Document Overview (Second Edition)**. 2012. Disponível em: https://www.w3.org/TR/owl2-overview/.

GUARINO, N. **Formal Ontology in Information Systems**: The science of microfabrication. [*S.l.*]: IOS Press, 1998.

HAWKINS, D. **Identification of Outliers**. [*S.l.*]: Chapman and Hall, 1980.

HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. **2016 49th Hawaii International Conference on System Sciences (HICSS)**, 2016.

POPKOVA, E. G.; RAGULINA, Y. V.; BOGOVIZ, A. V. **Industry 4.0: Industrial Revolution of the 21st Century**. [*S.l.*]: Springer, 2019.

RAMAKI, A. A.; ASGARI, R.; ATANI, R. E. Credit card fraud detection based on ontology graph. **International Journal of Security, Privacy and Trust Management**, 2012.

SALEHI, M. *et al.* Fast memory efficient local outlier detection in data streams. **IEEE Transactions on Knowledge and Data Engineering**, 2016.

WANG, J. *et al.* The evolution of the internet of things (iot) over the past 20 years. **Computers and Industrial Engineering**, 2022.

YU, Y. *et al.* Time series outlier detection based on sliding window prediction. **Mathematical problems in engineering**, 2014.