TÉCNICO LISBOA

Development of an Industry 4.0 Big Data Processing and Management System

Francisco Vidal Cabrita Carneiro

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor(s): Prof. João Nuno de Oliveira e Silva
Dr. Abdessadeq Zougari Ben Elkhayat

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. João Nuno de Oliveira e Silva
Member of the Committee: Prof. João Carlos Antunes Leitão

December 2020

To my grandfather, Victor Vidal.

Acknowledgments

I would firstly like to thank Dr. Sadeq ZOUGARI, my tutor in AKKA Technologies, for the excellent orientation received throughout the project and the clear definition of the project objective and structure. Also, for the kindness and friendliness of his orientation at the start and all throughout the project. Sadeq allowed me not only to make the best of my work but also in understanding the world of technical consulting and of AKKA.

I would also like to thank Prof. João Nuno de Oliveira e Silva, my tutor at *Instítuo Superiro Técnico* (IST) for his orientation in this Master Thesis. I learned a lot with Prof. Oliveira e Silva and his help and kindness were crucial for me to understand how to structure and present the work carried out in a scientific and rigorous way.

A special thank you to Prof. Fernando Lau, not only for his help in this Master Thesis but primarily for his in-valuable support throughout all of my two year Double Diploma at ISAE-SUPAERO. Prof. Lau is an incredible driving force for the Masters in Aerospace Engineering at IST particularly when it comes to the international influence and reach of the course.

To Prof. Isabel Ribeiro, a very special thank. Prof. Isabel has accompanied me through all the major stages of my academic path. From showing me around the day I registered myself in my university, Instituto Superior Técnico (IST), to helping me in my decision to study at ISAE-SUPAERO for my Double Degree. Your support and genuine concern will never be forgotten. It has been an honour to have such a dedicated and successful professional, teacher and researcher present in the presentation of this work.

I would also like to thank Benoit DALET for his crucial support in many technical matters, but also for keeping the mood high at the office. Without his valuable IT and DevOps expertise, and electrifying mood, the project would have been blocked multiple times.

To Benoit BAURENS, director of AKKA Research, thank you for helping me integrate not only in AKKA Research but also in the company community. Benoit always made sure I, and all the other collaborators at AKKA Research, had everything we needed to be comfortable and productive at work.

I would also like to thank IST and all the teachers and support staff who have made my academic experience challenging yet fun and also for giving me the opportunity to embark in this international experience at ISAE-SUPAERO in France which allowed me to learn another language, experience a different academic system and perform an excellent internship. I owe a great part of my accomplishments in France and other areas of my life to the preparation that I received at IST.

Finally, I would like to thank my family and friends who have made my academic experience possible and worthwhile. I would like to thank in particular my parents for all the help and love they have given me in these last five years and in supporting me in my decision to come to France. Without them, I would never have had the courage and opportunity to embark on my Double Degree. To my grandfather as well I am grateful and who I know would have been proud of what I have accomplished these past five years, and to whom I would like to dedicate this work to.

Abstract

Industry 4.0, also known as the fourth industrial revolution, refers to the enhancement of automation, connectivity and intelligence of machines within a production environment. Its objectives are increasing reactiveness, asset monitoring, decision making and value creation of manufacturing. Industry 4.0 solutions are however challenging to implement as they rely on data systems capable of handling massive amounts of data with low latency, high delivery guarantee, high fault tolerance and high information throughput. This generally implies satisfying Big Data, cyber physical system, Internet of Things and distributed computing system constraints. Understanding the high potential associated to Industry 4.0 solutions, AKKA Technologies, an Engineering and Technology consulting firm, decided to launch project ZORRO who's initial aim was to develop a data system capable of detecting manufacturing anomalies in real-time and to serve as a source of internal Industry 4.0 know-how. As a contribution to project ZORRO, in this work an Industry 4.0 Big Data solution, capable of satisfying Industry 4.0 and project ZORRO's objectives for automatic anomaly detection, is proposed providing end-point solutions consisting in machine learning platforms for automatic anomaly detection, data visualisation platforms for real-time dashboarding and monitoring, data science tools for extraction of important production insights, and batch and stream processing engines to implement virtual sensors, online predictions and other real-time calculations. The architecture was designed to respond to the various constraints associated with these types of Big Data systems whilst remaining modular, affordable using commodity hardware, scalable and microservice based.

Keywords: Industry 4.0, Big Data, Industrial Internet of Things (IIoT), Stream Processing, Machine Learning, Real-time Anomaly Detection.

Resumo

A Indústria 4.0 refere-se ao aprimoramento da automação, conectividade e inteligência de máquinas e sistemas informáticos num ambiente de fabricação. Indústria 4.0 consiste em monitorizar e controlar o estado da produção com o objetivo de aumentar eficiência e eficácia da mesma. As soluções de Industria 4.0 são difíceis de implementar pois envolvem sistemas de informação capazes de gerir enormes volumes de dados, com uma baixa latência, alta garantia de transmissão, alta tolerância à falha e alto fluxo de informação. Estas soluções têm portanto de satisfazer constrangimentos associados a sistemas *Big Data*, cíber físicos, de computação distribuída e de *Internet of Things*. Identificando o potencial da Indústria 4.0, a AKKA Technologies, uma empresa de consultoria em Engenharia e Tecnologia, lançou o projeto ZORRO cujo objetivo inicial era desenvolver um sistema capaz de detetar anomalias de fabricação em tempo real e consequentemente equipar a AKKA com o conhecimento necessário para acompanhar os seus clientes industriais nas suas estratégias e implementações da Industria 4.0. Como contribuição ao projeto ZORRO, neste trabalho uma solução Big Data Industria 4.0, capaz de satisfazer múltiplos objetivos da Industria 4.0 é proposta. O sistema foi projetado com mecanismos de *machine learning* para deteção automática de anomalias, plataformas de visualização de dados em tempo real, ferramentas de ciência de dados para extração de informações relevantes e plataformas de processamento de dados. A arquitetura foi projetada para responder às várias restrições associadas a sistemas de *Big Data*, permanecendo modular, escalável e baseado em micro serviços.

Palavras-chave: Industria 4.0, *Big Data*, *Industrial Internet of Things* (IIoT), Processamento *steaming* , *Machine Learning*, Deteção de Anomalias em Tempo Real.

Contents

List of Tables

List of Figures

Nomenclature

Acronyms

AI	Artificial Intelligence.
AMQP	Advanced Message Queuing Protocol
AWS	Amazon Web Services
CI-CD	Continous Integration, Continous Development Pipleine
CPS	Cyber Physical System.
ETL	Extract, Transform and Load
GCP	Google Cloud Platform
IaaS	Infrastructure as a Service
IIoT	Industrial Internet of Things
IoT	Internet of Things
JSON	JavaScript Object Notation
K8s	Kubernetes
ML	Machine Learning
MOM	Message Orientated Middleware
MQTT	Message Queuing Telemetry Transport protocol
PaaS	Platform as a Service
PCA	Primary Component Analysis
PvR	Precision vs Recall.
QoS	Quality of Service
ROC	Receiver operating characteristic curve.
SaaS	Software as a Service

SMACK S - Spark, M - Mesos, A - Akka, C - Cassandra and K - Kafka

SVM Support Vector Machine

TCP Transport Layer Communication.

TICK Telgraf, InfluxDB, Chronograf, Kapacitor

VM Virtual Machine.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

In an increasingly digital world, most complex and large-scale production lines have adopted what is called an Industry 3.0 method, where machines, electronics and IT systems are aiding factories in increasing the speed, efficiency, capacity and flexibility of production through automation. Industry 3.0 gives way for monitoring and connecting these machines and systems into a network known as an Industrial Internet of Things (IIoT) environment or Industry 4.0. Industry 4.0 is considered the next Industrial revolution (Figure 1.1). An Industry 4.0 solution consists in creating a connected network of machines, products and processes that make up the production context, monitoring and or controlling their status, and through this extract value and drive effective actions and decisions.

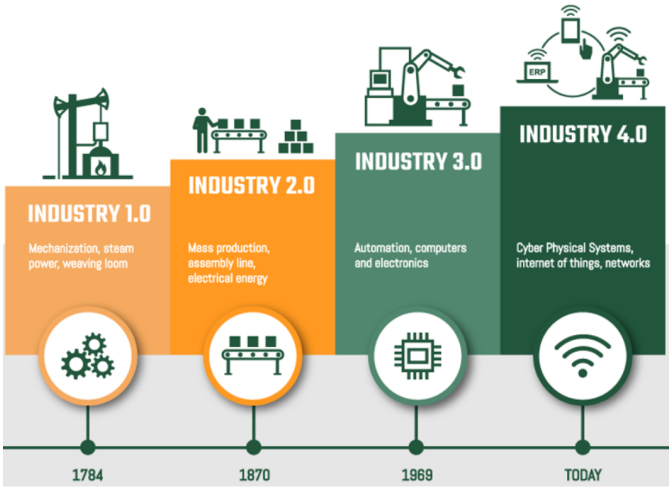


Figure 1.1: The four industrial revolutions [1].

Access to the information from this network allows for real-time anomaly detection of products, machines or IT systems, equipment predictive maintenance and general forecasting, process optimisation, process explicability and cost accounting, supply/value chain resilience. Given that machines and systems are connected through a network, Industry 4.0 also allows for automatic machine and process

control as well as inter-system cooperation.

Industry 4.0 promises manufacturers additional annual revenues of 2% to 3% on average [2]. For the European industrial sector, additional projected revenues amount to €110 billion annually. Due to this, it was predicted that by the end of 2020 European industrial companies would invest €140 billion [2] annually in Industry 4.0. In Industry 3.0 solutions, machine and system maintenance, for example, is done in a more responsive manner or with periodic check-ups which require experts and manpower to analyse and identify potential problems followed by more experts and manpower to solve the issue. Similarly, real-time supply chain resilience does not really occur since, from the point of view of the processes, the factory floor is unaware of what is happening both on the suppliers and the client's sections of the value chain. Industry 4.0 solutions can not only automatically detect anomalies in real-time and predict future problems but can at least allow for the possibility of remote analysis of the problem, saving the need for a visit from dedicated experts. Similarly, given that machines are connected to a network and to an information system, they could adapt their production rate to meet supplier or client sudden behaviour changes or respond in real-time to unexpected events such as machine failure. Finally, the data produced from this connected network can provide important insights for optimisation opportunities previously not identifiable.

Despite these clear advantages according to McKinsey Digital [3], only 30% of technology suppliers and 16% of manufacturers have an overall Industry 4.0 strategy. McKinsey suggested that the main implementation barriers were difficulties in coordinating actions across different organisational units, concerns about cybersecurity and data ownership when working with third-party providers, lack of courage to push through a radical transformation, difficulties to adopt new technologies, and lack of necessary talent. These results are understandable since an Industry 4.0 solution requires the implementation of complex data systems capable handling huge amounts of data (Big Data), complex network topologies (IoT) and large computational capacities (cloud computing) [4]. Therefore, the implementation of such systems and solutions cannot be done by production experts alone.

Having understood the high potential of Industry 4.0 and other data and digital transformation solutions with similar use cases and constraints, many consulting firms have expanded and diversified their operations to include support to implement these solutions. In fact, Industry 4.0 is one of many domains where Big Data, cloud computing and IoT intervene. It is therefore not a coincidence that many consulting firms are investing in these domains as it allows them to implement Industry 4.0 and other large scale digital transformation solutions to their clients in areas such as health care, energy production, agriculture, logistics and smart cities where data and processing is required in real-time to drive decisions and improve efficiency. In this context, AKKA Technologies [5], an Engineering and Technology consulting firm, launched a project named ZORRO with the aim of not only equipping AKKA with the know-how required to advise customers on what an Industry 4.0 and Big Data solution consists of (in terms of architecture, technology and cost), but also understand what added value it can give manufacturers.

1.2 Mission and Objectives

From AKKA's perspective, the crucial high-level mission of project ZORRO is to develop know-how regarding Big Data and Industry 4.0 solutions by understanding what they consist of in terms of architecture, technology and cost, as well as what added value they can give manufacturers. The ZORRO system alone does not have the objective of being sold as a product. Nevertheless, ZORRO still aims in becoming a Meta solution capable of satisfying Industry 4.0 objectives and overcoming the constraints associated to these systems. The idea is that this solution, allied to the know-how created around it, can then be used as a reference for AKKA when aiding manufacturers in their digital transformation and in achieving Industry 4.0. We use the term Meta solution to stress the idea that the ZORRO system, or one of its sub-systems, can be instantiated and adapted based on the concrete customer needs and requirements. The main objective of this work is therefore to help AKKA satisfy these objectives.

1.3 Implementation and Results

In this work, a Big Data system for the collection, storage and processing of real-time IIoT type data was implemented. The system was conceived with the following Industry 4.0 objectives in mind: increasing reactiveness, asset monitoring, decision making and value creation of production environments. The system was developed to have the following features:

- Data visualization / dashboarding tools supporting dynamic and interactive queries.
- Real-time (streaming) and batch data processing system.
- Machine Learning (ML) and Data Science platforms for data exploration, data analytics and general insight extraction.
- Real-time anomaly detection, and alerting system, through ML algorithms.

Furthermore, the solution was designed to address the Big Data constraints of velocity, volume and variety, data streaming constraints involving message latency and message throughput whilst guaranteeing the real-time analysis and reactiveness that Industry 4.0 promises. Figure 1.2 illustrates a high-level representation of the implemented system.

The ZORRO system was divided into three main layers: the IoT data production layer, the TCP data processing layer and the serving layer. The IoT layer represents the physical data production layer. In the context of the project it was simulated by a factory simulator. This layer would usually contain the physical machines, products and processes to be monitored or controlled. The TCP layer, standing for the data communication Transmission Control Protocol, allows for both real-time and batch data processing, insight extraction and storage. Anomaly detection algorithms, trained using machine learning, were implemented and deployed here. On the serving layer, a series of tools were developed to extract value from the developed platform. This layer can be thought of as the customer layer as it is where business value is extracted. As described above, this layer includes real-time data visualisation and dynamic query platforms, data science and machine learning development environments, and automatic

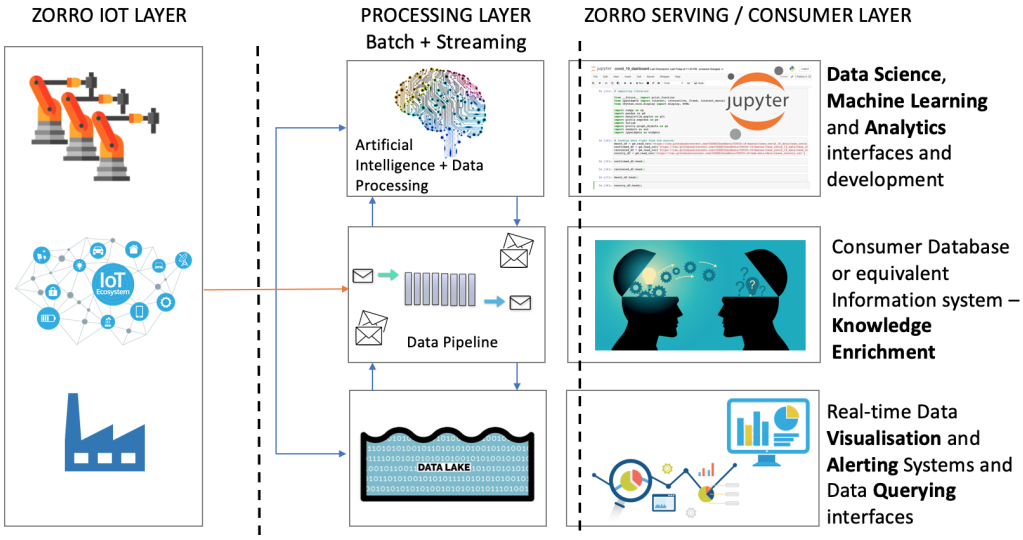


Figure 1.2: High-level representation of implemented system.

knowledge enrichment features including alerts and creation of virtual sensors, which are outputs of the stream and batch processing systems.

Following the system development and deployment, performance tests were carried out through various simulation scenarios varying the data load on the system (size of the simulated factory). In terms of quantified performance, we mainly studied the system latency. In particular, the time it takes to visualise data or detect production equipment anomalies. Data exploration and machine learning campaigns were also done in order to verify that the data scientist and machine learning engineer workflow can be done. Throughout the state-of-the-art study, development and testing processes, a consistent documentation was done to aid AKKA in its primary high-level objective which is to generate internal know-how. Likewise, all the code and system configuration files developed throughout the internship were stored in a GitLab [6] repository and a user guide was created for future use and ease of deployment.

1.4 Thesis Outline

In this thesis we shall present the work carried out as follows. Firstly, chapter 2, describes the background and context of this work. In terms of context the chapter begins by describing in more detail AKKA Technologies and the ZORRO project. The main conclusions drawn from the state-of-the-art study, namely how Industry 4.0 is being tackled today and what architectures, technologies and models are used and were chosen to build the solution in this work are described. Chapter 3 discusses in more detail how the work's mission and objectives were approached and accomplished. It describes the implementations that were done on the project and in particular how each sub-system was deployed and made to interact with the system as a whole. Chapter 4 describes the results that were obtained from the developed system both in terms of performance and system outputs. Chapter 5 concludes with a review of the work done by analysing what was accomplished, what can be improved and what were gains that were obtained from this project.

Chapter 2

Work Context and Industry 4.0 Big Data Systems

In this chapter a more detailed description of the contextual elements of this work, namely the company with which it was carried out, AKKA Technologies, and the project in which it is inserted, ZORRO, is done. This chapter also reflects the analysis and conclusions of state-of-the-art study that was carried out prior to development of the ZORRO system. This involved analysing the domain of Industry 4.0 in terms of what the primary difficulties and solutions are and understanding the technologies that are used to build them. This study was used to justify and define the ZORRO system requirements analysis and to select the best technologies, methods and architectures on which to build it. We will focus this description primarily on the architectures, algorithms and technologies that were used in this work. Nevertheless, separate documents were produced during this state-of-the-art study explaining all the different technologies studied. The purpose of this documentation was to serve as future technical documentation and guides for engineers who pick up the ZORRO project and need to get familiar with the technologies and tools used.

2.1 Context

This work was carried out in the context of an internship at AKKA Technologies. AKKA Technologies is an engineering and technology consulting group founded in 1984. The firm is positioned in a series of technological sectors most notably: aeronautic, automotive, energy, rail, space and defence, information systems, life sciences and telecommunications (Figure 3). The company's group holds more than 50 establishments in France as well as in Germany, Belgium, Spain, India, Italy, Morocco, Romania, United Kingdom, China, North America and Switzerland.

Having reached an already substantial size, employing 21,000 people, AKKA Technologies had plans to expand and diversify its operations. This diversification mainly involved a shift towards the data and digital domains, with the aim of creating digitalisation solutions for customers. This was highly motivated by a clear shift in market demands where organisations are looking to the use of data and information

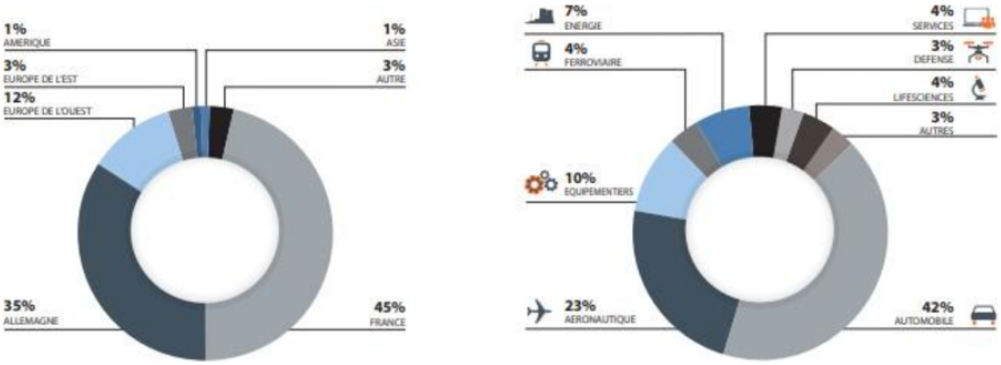


Figure 2.1: AKKA's location and activity sectors.

systems to drive business decisions. Due to the current COVID-19 pandemic, this diversification process has been accelerated in particularly because of the impact the pandemic had on the aeronautic and general transportation sector. AKKA is currently launching a new data solution team in Paris (AKKA Data Respons team) as well as investing in the training of its employees in areas such as cloud computing and cyber security.

2.1.1 AKKA Research & Development

The presented work was carried out at AKKA's Research and Development department, AKKA Research. AKKA Research is AKKA's internal hub for innovation and cutting-edge skills in technological and engineering domains. The diversity in the department's expertise gives way to its vast and extensive exploration of front-line innovative technologies. From data scientists and machine learning experts, IT and DevOps engineers to robotics and embedded system engineers, AKKA Research has pioneered projects reflecting this diversity. Some of these projects include:

- **Air-Cobot**: The first mobile collaborative robot that inspects aircraft during maintenance and check-ups.
- **Autopilot** [7]: An autonomous driving project involving cooperating self-driving cars controlled and communicating with an IoT platform in real-time.
- **Link & Fly** [8]: A flying multimodal prototype showcasing the advantages of the aircraft of the future.
- **InGeoCloudS** [9]: Solution for the creation and sharing, at a massive scale, of environmental data.
- **5G-MOBIX** [10]: New 5G cross-border corridors for connected and automated driving.

Apart from providing AKKA's consultants with an internal hub of technological know-how, AKKA Research aims in extending AKKA's impact on their customers by introducing innovation to the client's existing operations. This occurs because many enterprises may only invest in innovation that is very specific to their domain and AKKA can intervene as the umbrella innovator. With regards to the data

and digitalisation domains, AKKA Research is a clear driver in AKKA's diversification process as their projects are already very much orientated towards data and digitalisation, building solutions on cloud computing, IoT, 5G and machine learning.

2.1.2 Project ZORRO

Having understood the high potential of Industry 4.0 for increasing efficiency of production environments and wanting to increase its expertise on digital transformation and data solutions, the ZORRO project, which stands for (Zero Defect Production Strategy based on Processing Big Data of Products, Processes and Quality) was created at AKKA Research. This project emerged in the context of an European H2020 project with multiple partners and with the objective of responding to the demand for European Industrial efficiency and competitiveness.

As the name suggests, the project's main goal is to develop an Industrial IoT (IIoT) data platform solution in a multi-production-line and multi-stage factory context, capable of detecting production anomalies in real-time and attaining zero-defect production. The data streams from one production line is shown in Figure 2.2.

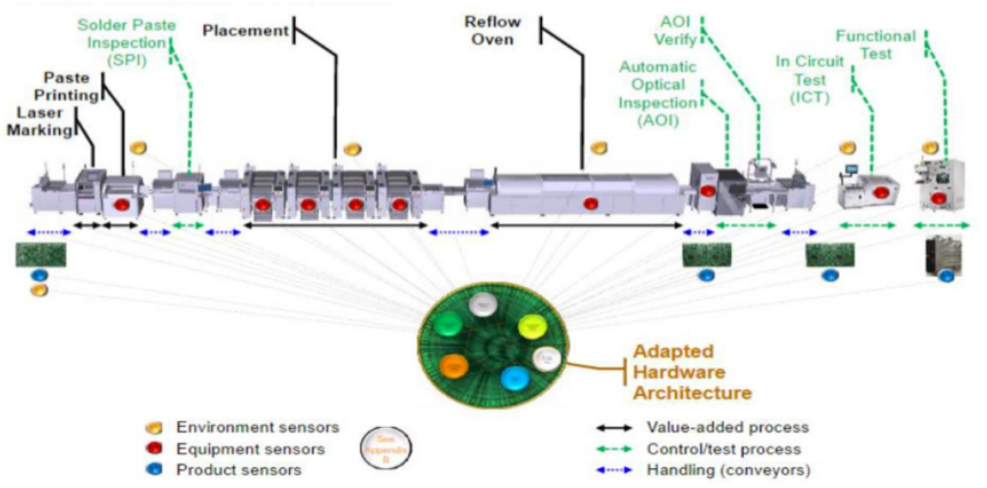


Figure 2.2: ZORRO data streams overview (inspired by a PCB production line).

This production line data flow is inspired by Printed Circuit Board (PCB) production lines. As can confirmed in Figure 2.2, data can be produced from various sources either containing production equipment data, manufactured product data or environment data (red, blue and yellow sensors). The idea behind the ZORRO IIoT system is to monitor the production status using these data streams and gather insights about the production as well as detecting anomalies and identifying root causes in real-time. These insights and anomalies are to be displayed as alerts, diagnostics and reports in a format that can be interpreted by production stakeholders as shown in Figure 2.3. To do this, the ZORRO solution must include an IoT data collection system and a database system to store data from the production processes. This data will then be used to extract insights and train machine learning models which in turn will be used to monitor the data being produced in the factory and detect the anomalies in real-time.

Given that factories can grow huge in size, particularly in mass production contexts, the ZORRO project was thought of as an Industry 4.0 solution satisfying Big Data constraints.

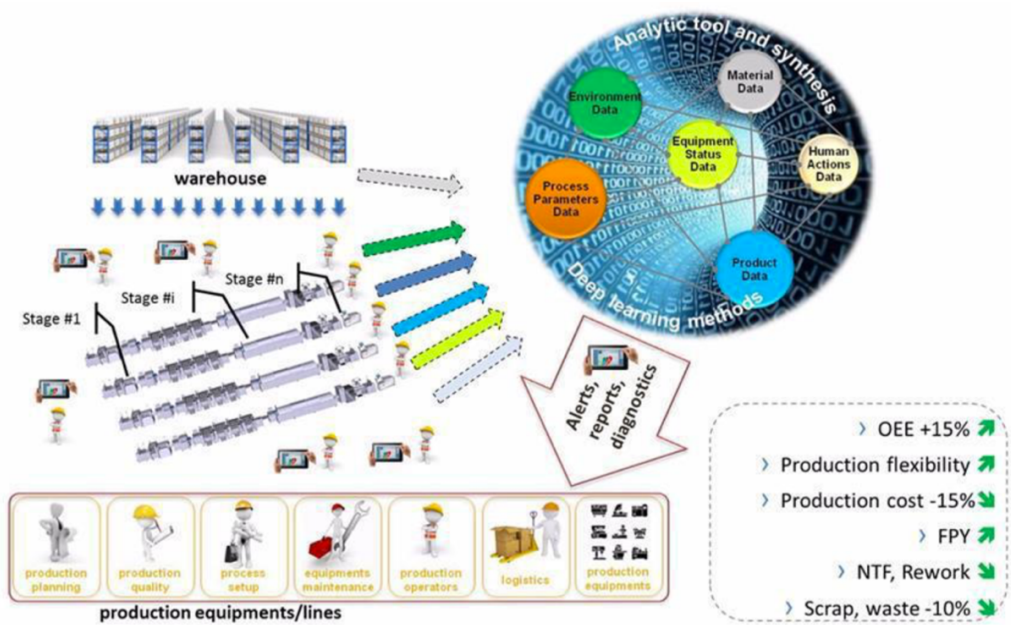


Figure 2.3: ZORRO industrial data platform solution - integration with factories.

The ZORRO solution is therefore a Big Data scale, IIoT data processing platform. The gains of the ZORRO IIoT solution are also illustrated in this figure and are improvements that are generally associated to Industry 4.0 solutions. To summarise the list, the ZORRO project aims in increasing production efficiency by reducing maintenance cost, increasing flexibility and driving optimisation.

The ZORRO project alone does not have the objective of being sold as a product, instead, it's aim is to become a Meta solution to be instantiated depending on the concrete customer needs and requirements and to equip AKKA with the know-how required to advise customers on what an Industry 4.0 solution consists of in terms of architecture, technology and cost, as well as what added value it can give manufacturers.

2.2 Industry 4.0 Today

The term Industry 4.0 was coined in Germany, in 2011, and was born through an initiative to improve manufacturing by exploiting digitalization and new technologies [11]. As discussed in the introduction of this work, the idea behind industry 4.0 is to use data from connected machines and IT systems in the production environment to increase efficiency and quality of production. Although today there is no standard on how to perform Industry 4.0, robust architecture proposals are starting to rise. One such mature proposal can be found in [12], where a layered architecture is introduced. According to this proposal, an Industry 4.0 ecosystem is composed of 5 main layers as illustrated in Figure 2.4.

The layers can essentially be grouped into two main components, a Cyber Physical System (CPS) with advanced connectivity and real-time data acquisition of the physical production system and an

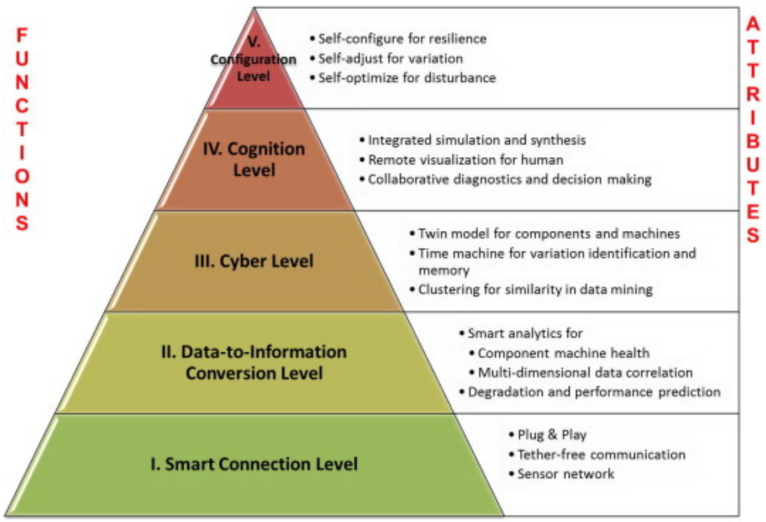


Figure 2.4: Industry 4.0 proposed general architecture [12].

intelligent data management system who's objective is to leverage the data production and connectivity of the physical level to extract useful insights from the data and increase efficiency through data-driven decisions. By grouping the Industry 4.0 ecosystem into these two main components and the 5 layers which make up the system as a whole, we can already better understand why Industry 4.0 is difficult and complex to implement as it requires expertise on the physical and IoT domain, due to the correct and efficient measurement of processes with sensors, on the data engineering domain, not only because the volume and velocity of the data to manage is significant (Big Data) but also because the data needs to be streamed in real-time and stored or processed by distributed computing systems (cloud or edge computing), and on the data science domain, as algorithms for machine learning, analytics and insight extraction need to be employed to be able to extract value from the data. Furthermore, as can be seen in the final layer of Figure 2.4, the cognitive layer, following the implementations of the previous layers, in particular the data-to-information conversion level, the data and results can be used to develop control algorithms to govern certain processes on the production line automatically. This was done in [13] on the aerospace manufacturing industry where machine learning control algorithms were developed to perform the deburring process on certain parts automatically, a process which normally requires manual work, subjective quality judgement, and could now be done quickly and involving data-based judgement. Eventhough there is substantial value to be gained here, after succeeding in implementing a robust Industry 4.0 ecosystem other issues such as cyber security and data ownership arise [3].

In spite of the difficulties associated to Industry 4.0, manufacturing companies require it to increase the quality and speed of production. In the Aeronautics sector for example, airbus has delivered in the past 40 years 10,000 aircraft and aims in delivering 20,000 aircraft in the next 20 years (2019 numbers before the COVID-19 pandemic). To achieve this goal, airbus is investing in Industry 4.0 technologies such as the mixed reality [14] and Big Data platforms [15] as in Aerospace manufacturing, an increase in quality is an increased in safety.

Some organisation like SIEMENS are attempting to become reference providers and users of In-

dustry 4.0 [16]. However, these organisations naturally struggle to provide support and expertise on all aspects of the Industry 4.0 environment, as this demands knowledge on a vast range of technological fields as described above. It is for this reason that players like Microsoft, with their software and cloud computing offerings, are intervening and forming partnerships with organisation like SIEMENS to provide all technological aspects an Industry 4.0 solution requires [17]. This particular partnership arose due to the industries realisation that disruptive technologies such as cloud computing and Big Data are tied with the feasibility of Industry 4.0 [4]. Big Data and cloud computing are technology enablers of Industry 4.0.

2.3 Big Data and Cloud Computing Processing Platforms

Data processing platforms are an essential component of Industry 4.0 systems as they contribute to layers II, III and IV of Figure 2.4. Real-time data systems used in combination with persistence storage (batch) data systems are becoming increasingly popular due to their ability to store large amounts of data and extract insights from it whilst also being able to react to changes and events represented by data being produced in real-time. The ZORRO project is focused on an industrial production context, nevertheless, there are a significant number of domains where data is required in real-time to generate alerts or make long-term or short-term decisions. Examples include energy production, agriculture, healthcare, smart cities etc. Notice that in these domains, there are also physical entities that need to be monitored or controlled (wind turbines, tractors, medical equipment etc), therefore also including IoT type data. Due to the wide variety of applications, but similar base needs, data systems that were not necessarily made for Industry 4.0 were also studied.

In this section what is required to build a Big Data and IoT processing solution both in terms of architecture and technologies is discussed. This section reflects the the state-of-the-art study carried out on Big Data and cloud computing processing systems. The section gives an overview of the technologies, architectures and enablers of Industry 4.0 and Big Data systems but focuses primarily on the technologies and architectures that were used.

2.3.1 Cloud Provider Solutions

Allied and as a consequence of the rising popularity of digitalisation and data solutions, many technological companies have entered the cloud computing field, either as users or as providers, realising that the paradigm for application deployment has changed significantly. Cloud computing refers to the availability on demand of computer system resources (e.g. data storage, servers, networks, applications) [18]. Cloud computing offers these resources in the form of 3 primary services [19]:

- Infrastructure as a Service (IaaS) - IaaS refers to the provision of physical and virtual computing resources by combining physical and infrastructure layers.
- Platform as a Service (PaaS) - As the name suggests, PaaS provides computing platforms such as operating systems, programming language execution environments and database engines.

- Software as a Service (SaaS) - This service provides on-demand applications and service through the web for internet end-users. An example of such a service is Dropbox. The provider is responsible for all the infrastructural and software maintenance.

Modern applications require a micro-service based architecture with fast and flexible scalability. Cloud computing platforms allow for this as they manage large data and processing centres with flexible virtualisation tools installed. This is what allows these platforms to provide or remove multiple computing resources in just a couple of clicks. The alternative to this would be to buy servers for internal use based on the requirements. This solution has a series of disadvantages, mainly involving scalability and flexibility, which cloud computing does not as cloud providers abstract away all the complexity associated to the maintenance and deployment of such systems and only charge the customers for what they use. Given that in Industry 4.0 systems, large amounts of data are produced and need to be processed, cloud computing has become a promising solution to respond to this on-demand computing resources [20].

Understanding that there were many patterns to the type of applications that their customers were deploying and high business value, many cloud providers began to develop out of the box SaaS and PaaS solutions built on their computing resources. Many cloud providers offer IoT, Big Data and Artificial Intelligence platform solutions, which are precisely the type of solutions that ZORRO aims to provide. These providers are therefore ideal candidates for a state-of-the-art reference for this project. The big three cloud providers are:

- Amazon Web Services (AWS).
- Microsoft Azure.
- Google Cloud Platform (GCP).

In the following sub-sections we will explore the offerings of each one of these providers and focusing particularly on their IoT, Big Data and data processing platforms to understand what features and architectures these systems are composed of.

Amazon Web Services

AWS, founded in 2006, is a globally available platform of web services based on the cloud. It is considered to be the leader within the big three cloud providers. In terms of IoT platforms, AWS has a mature and complete offer. The main core of this platform is the AWS IoT Core [21], Figure 2.5.

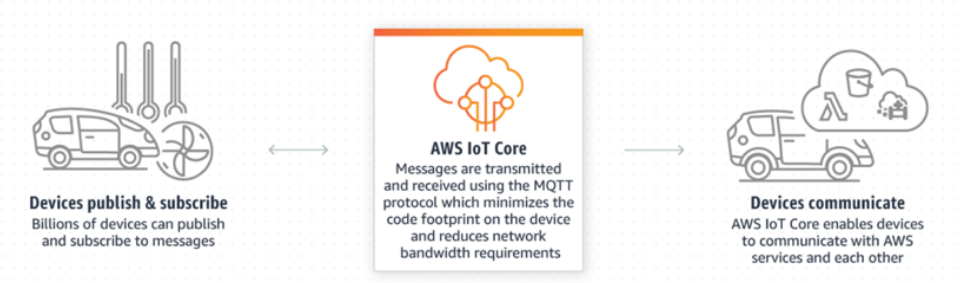


Figure 2.5: AWS IoT Core concept [21].

The solution allows for devices to produce messages (publish) and subscribe to messages (consume) to and from the platform that is built to handle this data management efficiently. The AWS IoT core also allows for interaction with other AWS services such as Lambda functions, which can trigger an action depending on the content of the messages received from the IoT core. This allows for what is known as event driven architectures, where events (described by messages) can activate and trigger certain system responses. The IoT core can then be enriched by connecting it with other AWS services such as IoT Analytics, IoT graphs and IoT events (see Figure 2.6). These complementary services are used for some form of value extraction from the data. AWS also contains core extensions such as IoT device management and defender services, which provide respectively IoT device management, creation and connection as well as secure communication to and from all devices.

Figure 2.6: AWS IoT services. [21].

Microsoft Azure

Azure is Microsoft's cloud platform founded in 2010 and considered number 2 of the big three providers. It is an extremely popular solution for the execution of services in the cloud for enterprises. Azure IoT Hub [22] is Azure's solution for collecting metrics and information from IoT devices, Figure 2.7.

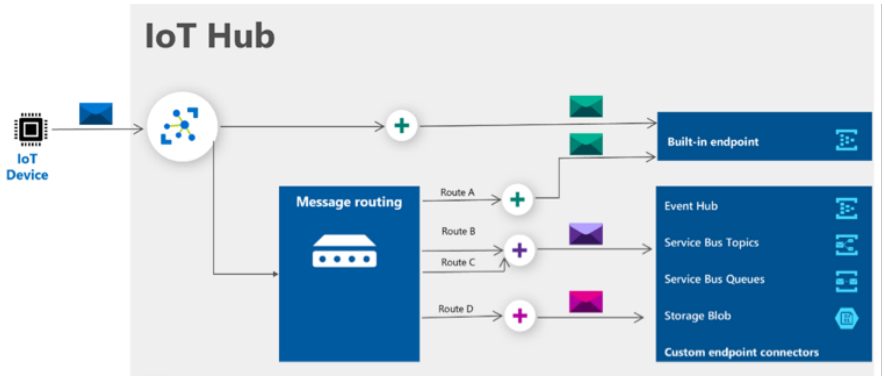


Figure 2.7: Azure IoT Hub high-level architecture [22].

Messages that are produced by the IoT devices, are sent to the IoT Hub. The messages can in turn be organised by routes, allowing for later redirection to end-point services. End-point services can be varied in nature, but essentially extract some form of value from the data. Azure event Hub for instance, allows for an event driven architecture as messages that fall into certain routes can activate certain

Azure functions, which in turn create some system response. Similarly, messages can be redirected to Azure's streaming analytics platform to perform real-time analytics on the data. Finally, messages can be sent to Azure's storage blob for persistent storage and future value extraction. Azure's IoT hub can be complemented with Azure's IoT central which allows for the connection and monitoring of individual IoT devices. Azure's IoT Central also provides template interfaces for energy, government, healthcare and retail solutions. Another end-point service that can complement Azure's IoT platform is Azure's digital-twins platform. This platform has the aim of creating digital representations of physical objects or systems. This is an extremely interesting use case for IoT systems as digital representations will ultimately aid in modelling, experimenting on and understanding the real physical system at hand. For example, if the system is a wind turbine and we wish to test new features on it, one could use the digital representation of the turbine to perform these tests rather than potentially compromising the correct behaviour of the real physical system. As illustrated in Figure 2.8, a digital twin solution provides a real-time graphic digital representation of a system connected to multiple azure services either to keep the digital twin up to date (IoT Hub) or to extract some insights from the twin.

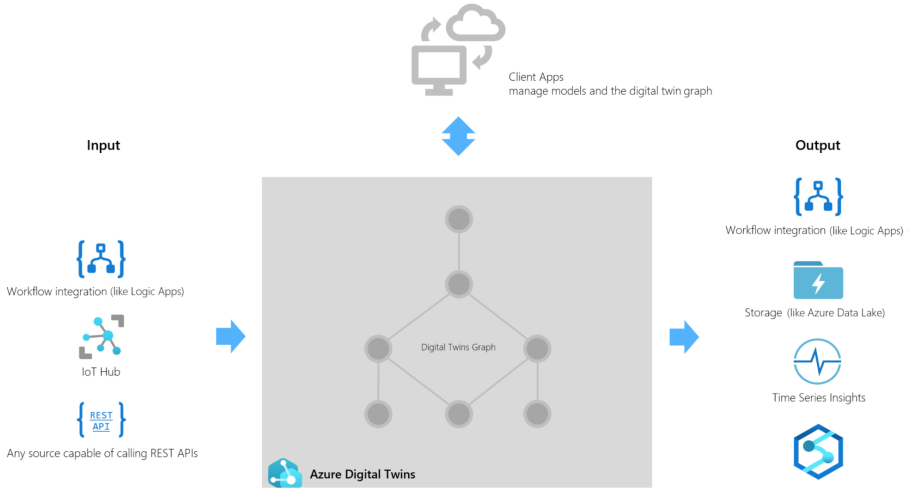


Figure 2.8: Azure digital twins GUI [23].

Google Cloud Platform

Google's GCP, founded in 2008, is considered to be the third top cloud computing provider and is growing at a very fast rate. Like the previous two cloud providers, GCP offers a mature IoT data platform solution [24] containing various sub-systems to extract value from the data. Figure 2.9 illustrates the full IoT data platform solution.

As can be seen in Figure 2.9, the idea behind GCP's IoT platform is to ingest IoT type data, store this data for Big Data insight extraction and use artificial intelligence (AI) and machine learning (ML), in real-time, to drive automatic decisions and obtain insights in real-time. Once again, the event driven architecture is seen here as events, represented by messages, trigger certain system responses. Some of the systems provided by GCP and included in the figure are:

- Cloud Pub/Sub: Central messaging system of the data platform.

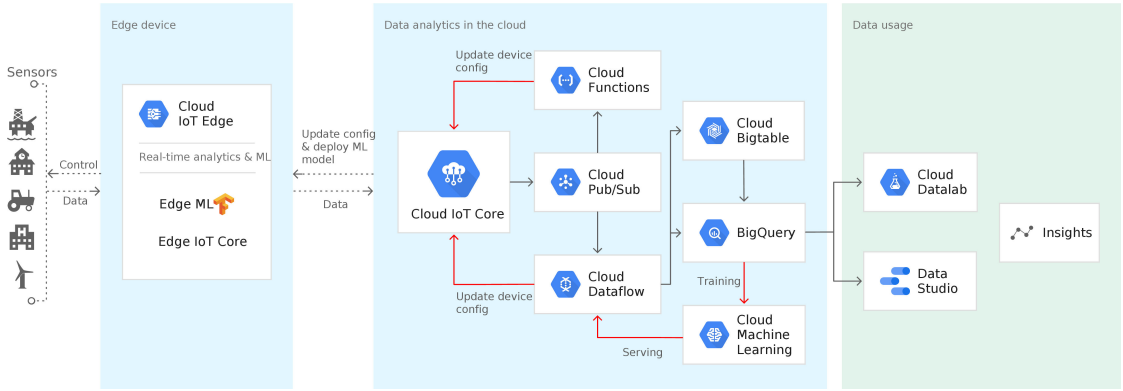


Figure 2.9: GCP IoT data-platform [24].

- Big Table: Big Data persistent storage.
- Big Query: Allows for queries on the large scale Big Table data storage system.
- Cloud Machine Learning: Allows for the training and deployment (usage) of ML models.

Notice also that GCP's IoT platform allows for the control of the IoT layer, through two way communication and using edge ML. This is an extremely powerful concept as the real-time decisions that are done on the data platform can trigger automatic reactions on the physical process IoT layer. Edge computing is an interesting aspect of Industry 4.0 introduced by GCP in Figure 2.9. Edge computing refers to computing power that is installed close to the system of interest. This proximity allows the system to provide the low latencies that some real-time actions require. In Industry 4.0, examples of these actions include automatic machine control and supply and value chain resilience. Like the previous two providers, GCP's solution also allows for an offline insight and value extraction tools through what they call a data usage layer.

Implied High-level Architecture of Industry 4.0 Data Platforms

By studying the solutions developed by these technological and cloud giants, the high-level architecture and building blocks of IoT and data platform solutions has become clearer. The objective of this study was not to compare the different cloud providers, as they all have very similar offerings, but rather to identify the similarities between them and design an architecture that is inspired by the strong points of all of these solutions. From all three solutions, it is clear that the ZORRO platform must contain a message broker allowing for the temporary retention of data so that exterior services can consume and use this data in real-time.

There must also be an intelligent processing platform capable of detecting equipment and system anomalies from the production floor. It has become clear that the algorithms that make up this platform should be developed using machine learning due to its ability to make decisions and predictions based on multiple sources of data. After detecting anomalies, a complete IIoT system should be capable of performing some action on the IoT level or other elements of the architecture (sending an alert to a dashboard). The message system must therefore allow for bidirectional communication (publish-subscribe architecture).

In order to extract insight and thus value from the data being produced in real-time, analytics and data processing engines must have access to the data as close as possible to the time when it is produced. Allied to this analytics system, there must be some sort of interactive tool for developers and Machine learning Engineers to write their applications to the platform given their use case. Finally, as was seen in the AWS solutions, communication security must be considered, particularly when there is a control loop. With this initial state-of-the-art study, one can already create a high-level system architecture of the ZORRO solution.

The system must be composed of the following generic sub-systems: IoT data measurement and production system, data collection and pipeline system, long term data storage system, data processing and analytics system, data visualisation and insight extraction system. The logical high-level structure of such a system is represented in Figure 2.10.

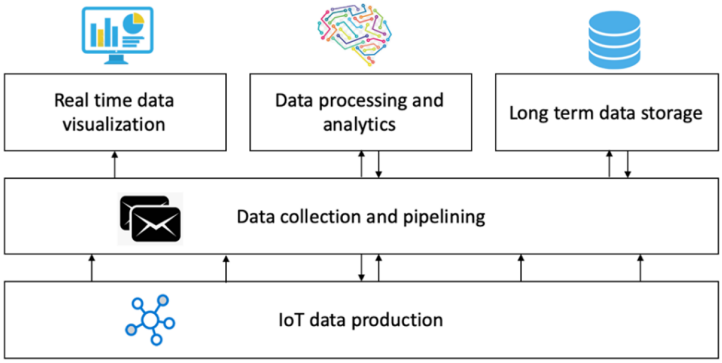


Figure 2.10: High level ZORRO solution.

These general sub systems will form the base for the future system requirement analysis. The proposed high-level system architecture not only allows for the fulfilment of the automatic anomaly detection objective of ZORRO, but is generic enough to build other Industry 4.0 end-point systems not originally planned for ZORRO. This is crucial as it allows us to make ZORRO a more complete Meta solution. In the following sections we shall answer the next question that emerges: what individual technologies are used to build each one of the sub-systems.

2.3.2 Message Broker Systems

As seen briefly above, message broker systems, or message orientated middleware (MOM), are systems that receive and temporarily retain data from various producers (publishers) and redirect the data to consumers (subscribers) that wish to consume it. The concept behind MOMs here described is illustrated in Figure 2.11. This type of interaction is known as public/subscribe communication. One question that might be asked is why not just create a dedicated channel between each two entities that wish to communicate (Figure 2.11.a). Essentially, not only will the number of communication links increase significantly (Metcalf's law, Figure 2.11.b), especially in IoT use cases, but it will shift part of the communication burden to the consumers and producers. This point-to-point communication leads to rigid applications and is unthinkable for dynamic large-scale scenarios. On the other hand, if there

is a message brokering system responsible for managing all these communication links, then not only will the number of links decrease, but the burden of managing communications is shifted to the broker, leaving consumers and producers free to focus on their original tasks. Likewise, the services communicating between each other become less rigidly coupled, which is essential in large scale systems. Publishers and consumers are decoupled in terms of time and synchronisation as they do not even need to be aware of each other. These types of publish subscribe architectures are very useful in Industry 4.0 scenarios for a number of reasons. Firstly, MOM systems abstract communications while making interoperability possible [25]. With regards to cyber-physical systems, they are communication protocol agnostic, making the adoption of communication technologies of different machine protocols easier [26]. MOM systems allow for entities in the production environment to produce data therefore making their status known and consume data allowing for them to be controlled. This is interesting in automatic production environments where machines and products communicate and cooperate in a fully autonomous production context.

(a) Communication without a MOM.

(b) Concept of MOM.

Figure 2.11: Message Orientated Middleware interest [27].

There are many message brokers available today each originally created with different architectures and use cases in mind. Examples include Kafka, RabbitMQ, HiveMQ, Mosquito etc. One extremely popular message broker for Big Data message management, being a reference technology, was Apache Kafka [28]. Apache Kafka [29] was originally developed in LinkedIn to become the unified central data pipeline of the application. Kafka is a distributed publish-subscribe based durable messaging system. In Kafka, messages, which are referred to as records, are key-value pairs that are stored in topics. Topics are classed groups of messages (family of messages). A record can include any type of information, for example, an event-based record. Data is written to disk in Kafka, but only for a specified retention period. This data retention allows Kafka to provide data replication and therefore high availability and fault tolerance. Figure 2.12 illustrates the general Kafka data production and consumption model.

If we consider the sub-figure on the left of Figure 2.12, we see that a Kafka cluster is composed of entities called brokers. Brokers store and manage the topics. Producers publish messages into specific topics. If a consumer is interested in data from a topic, it can subscribe to this topic and pull the data from it. This is a very important feature of Kafka. It follows a consumer pull data model which means that consumers will never be overloaded with messages. Topics are divided into partitions. Partitions are essentially what allow Kafka to be a scalable Big Data system. Topic partitions are stored across brokers

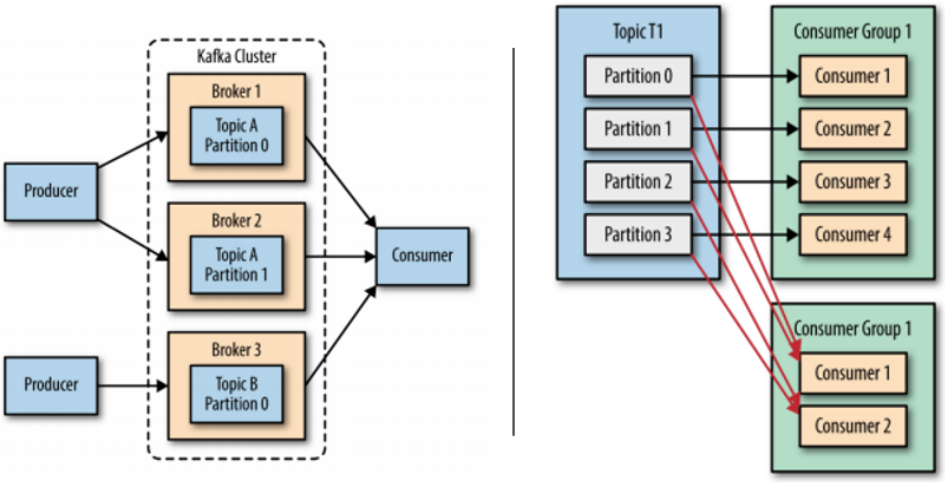


Figure 2.12: Kafka publish / subscribe model [30].

as shown in Figure 2.12 on the left. If a consumer wishes to consume all messages from topic A, it will have to consume messages from all the partitions in this topic. To parallelize the consumption of data from a topic (for higher throughput and smaller latency), consumers can work together in a consumer group (right hand figure of Figure 2.12). In a consumer group, consumers will divide up the consumption of a topic by consuming data from an allocated number of partitions. As show in the Figure on the right, if there are four partitions (topic T1) and four consumers in a consumer group, each consumer will consume data from one partition. Given that data is unique on a per partition basis, no data is consumed twice. The criteria for choosing which messages fall into which partitions is defined, in a round-robin fashion, by the key of the messages in Kafka. Messages with the same key will be stored on the same partitions. Fault tolerance is achieved through partition replication. Partitions can be replicated and stored in different brokers. This poses other difficulties in terms of partition leader allocation and constant updates of partition replicas. These are the type of difficulties that are usually associated to distributed systems. This process is completely managed both by the brokers and by Apache Zookeeper [31], which is required in a Kafka deployment. Essentially, for each partition, a broker is selected to be the leader of said partition. As the leader, it is this broker that is responsible for receiving and sending messages to and from this partition. Similarly, this broker is also responsible for making sure all replicas of that partition are up to date. If the broker fails, then Zookeeper will select a new broker, containing a replica of the partition, to be the new leader.

Kafka is also a highly configurable system, allowing for the user to find a compromise between message throughput, latency and Quality of Service (QoS). Essentially, this is defined by four major configuration options. The first option is the maximum number of in-flight batches sent. This option defines how many batches of messages can be sent by a producer without having received acknowledgement of reception from any broker. If set to be greater than one, and the sending of one message fails, then there is a risk of message reordering. The second option is the reception acknowledgement policy. Kafka can be configured to only consider a message received when it has been written into the respective (leader) partition and to all partition replicas. The final two options are the lingering and the

batch size options. Batch size defines how large the batch of messages to be sent to the cluster can be. Larger batches will increase throughput but also increase latency. Lingering defines how much time the producer waits before sending a batch. Larger lingering will correspond to larger latency but more throughput.

Architecturally, Kafka is sound and has the necessary mechanisms to be horizontally scalable, configurable and fault tolerant. Kafka has proven to be very successful in streaming image data in [32], being capable of streaming hundreds of thousands of images per second, but also as a viable solution in IoT [33]. It was also chosen as the MOM solution for an Industry 4.0 (IIoT) scenario in [34], acting as the central data pipeline. In these studies, Kafka integrated well both with data processing platforms and data storage systems, which ZORRO must contain. Kafka's popularity and reputation is also shown by the fact that it is used in 80% of the Fortune 100 companies [29], making it a solid solution for a MOM system.

2.3.3 Data Processing System

In the context of Big Data systems, a data processing system is a distributed processing system of Big Data sources. It is distributed in nature as it is not possible to efficiently process Big Data sources in a single node system. Examples of data processing systems include [28] Hadoop (Hive), Spark, Flink and Storm. Traditionally, at the time where Hadoop and its map reduce paradigm was introduced by Google in 2006, these systems were used for batch processing only. Nevertheless, the world of Big Data has recently extended to stream processing as well.

One of the most popular go to options for distributed and large-scale streaming and batch data-processing is Apache Spark [35]. Spark is a unified multi-purpose stack for carrying out distributed calculations. It is used as a unified analytics engine for large scale data-processing. At its core, Spark is a computing engine (built on a cluster or computer system) that schedules tasks in a distributed fashion. Spark offers rich libraries allowing for SQL, machine learning, graphing and streaming specific operations on data (Figure 2.13). All these operations can be performed at significantly high speeds on a Big Data scale. For certain operations, Spark is known for being 100 times faster than Hadoop's map reduce paradigm [35]. Spark was successfully used for image streaming and online prediction processing in combination with Kafka in [32], being capable of making a prediction throughput of 40000 images per second with 4 Spark workers (26x26 pixel images of digits).

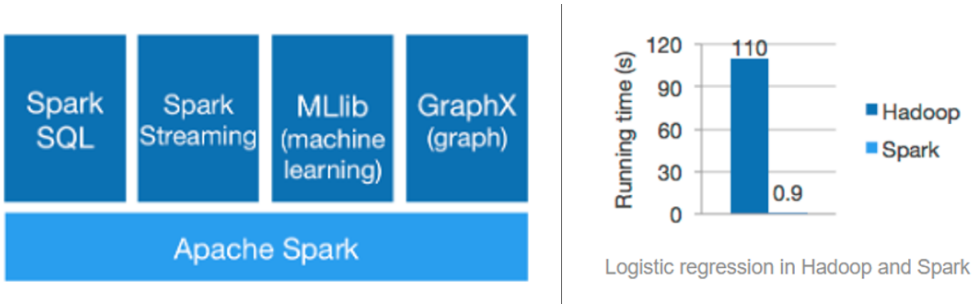


Figure 2.13: Spark ecosystem and comparison with Hadoop [35].

All spark programs consist of a driver program known as the SparkContext. The SparkContext is the entry point of every Spark application. A Spark cluster consists of a master (cluster manager) and worker nodes. The driver program reads the program (application) instructions, converts the instructions (script) into tasks and communicates these to the master node who then allocates resources and schedules tasks to the workers (see Figure 2.14). The driver program is thus a per application construct and the master is a per cluster component. The workers host executors where tasks are run. Each application gets its own executor process which stays up for the duration of the application and runs tasks which in turn are executed in multiple threads. This an advantage as there is application isolation on both the execution side of things (different executors) and the scheduling side of things (each application has its own driver program). As a resilient distributed system, Spark also guarantees fault tolerance in its executions. If a worker fails, spark reschedules the tasks that that worker was responsible for in another worker being also able to send part of the output that had already been produced by the failing worker to the new worker. A key component of Spark is the cluster manager which hosts the master node. The cluster manager is responsible for acquiring resources from the cluster. Examples of these include YARN, Mesos, Kubernetes or Spark's standalone cluster manager. In order for there to be fault tolerance in the Spark master then, like in Apache Kafka, Apache Zookeeper needs to be deployed to manage multiple master nodes and allocate one as the leader.

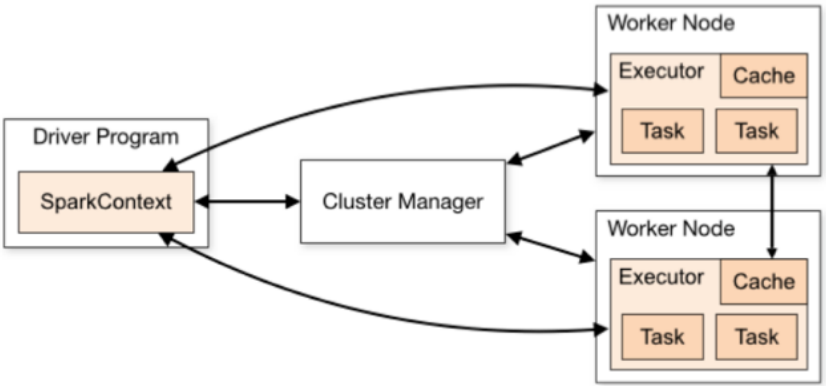


Figure 2.14: Spark cluster and execution architecture [36].

As of its second major release, Spark 2, Spark introduced streaming libraries allowing for the processing of data in real-time. Since its original introduction, the Spark streaming library has improved massively. It now consists of pretty much the same programming constructs of batch processing (Spark structured streaming). Likewise, Spark's original ML library, MLlib, has evolved to Spark ML which is faster and more data science friendly having introduced the DataFrame object. Overall, Spark's wide adoption, popularity, libraries and performance makes it an excellent choice for Big Data processing, analytics and machine learning.

2.3.4 Time Series System

One of the priorities of the ZORRO data storage system is to be optimised for large scale timestamped data. Traditional relational databases are generally not adapted for time series data mainly due to scale,

as time series data piles up fast, but also in terms of timestamp orientated queries (data stored in time-ascending order). With the rise of IoT, time series databases are becoming increasingly popular [37] with solution such as TimescaleDB, being developed by the famous PostgreSQL community, OpenTSDB, TICK stack, Graphite and others. After investigating these different solutions the TICK stack [38] was found to be the leader in terms of performance when it came to time series databases [39]. The central piece of the TICK stack is time series database InfluxDB, Figure 2.15.

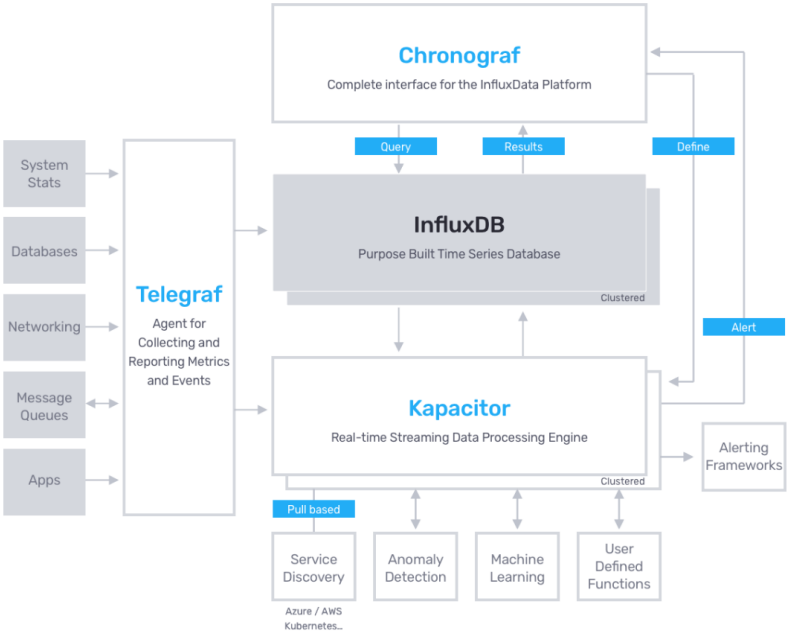


Figure 2.15: The TICK stack [38].

InfluxDB [40] is a NoSQL database purpose built for time series data. Data is ingested into InfluxDB through their metrics collection agent, Telegraf [41]. Due to InfluxDB's wide community adoption, Telegraf contains many plugins to consume data from various data sources such as Kafka, RabbitMQ, Kubernetes, MQTT, HTTP, TCP and many others. Chronograf is the user interface to interact with the stack. It provides data visualisation capabilities through dynamic dashboards and interactive queries as well as the definition of alerting programs. Finally, Kapacitor is the real-time data processing engine. It is used to perform a series of extract, transform and load (ETL) jobs. Kapacitor is also responsible for periodic data processing tasks on InfluxDB to optimise size of storage. For example, Kapacitor performs Downsamlping on the data which refers to the compression of multiple data points into a single data point containing average measurements. The TICK stack is mainly written in Go and processing jobs programmed in Kapacitor must also be written in this language.

With regards to the data model of influxDB, data is divided into measurements. These generally reflect an entity to measure. It can be something as big as an equipment or something as small as a sensor. There are two types of columns in a measurement series the fields, which refer to columns that hold numeric values, and the tags, referring to columns that hold string like values. One of the main difference between these two is that tags are indexed and fields are not.

The TICK stack has proven efficient in industrial applications such as mining [42], where real-time

IoT data is needed to be visualised and used to control certain quantities in the mine such as oxygen supply. InfluxDB contains a commercial cloud offering which is capable of ingesting millions of metrics per second. The disadvantage of this stack however is that, for Big Data applications, the commercial offering needs to be used and the open source version by itself is not horizontally scalable. Nevertheless, inspired by the state-of-the-art study developed in the previous ZORRO work packages, as well as the current findings, the TICK stack chosen as the time series specific stack mainly due to its performance with time series data and incredible dashboarding capabilities. Similarly, at the simulated scale of this work, the InfluxDB cloud offering is not required.

2.3.5 Persistent Big Data Storage System

In order to be able to store large amounts of data, so that it can be processed in parallel and efficiently by data processing systems, Big Data information systems require special types of databases. When it comes to Big Data systems, traditional relational databases are built in such a way that they have limitations in terms of scalability, fault tolerance and query speed. This is mainly due to the fact that relational databases are designed in a way such that all data is stored in one node and thus there is no partitioning of data. InfluxDB claims to be a scalable and efficient Big Data storage system for time series data but as seen previously has limitations in terms of its open source offering. Similarly, due to its relatively recent introduction, it does not have many out of the box connectors with other popular Big Data systems such as the previously studied data processing engine Apache Spark. It is for this reason that a fully InfluxDB ZORRO solution is not the most coherent choice. When investigating popular Big Data storage systems, many options are possible: Cassandra, MongoDB, HBase and Neo4j [43].

In this work, Apache Cassandra [44] was chosen as the main persistent storage database solution not only because it is open-source but also because it significantly leads in Big Data storage performance both in terms of throughput and operation speed (latency) [45]. It has also been shown to work well as a cheap and efficient IoT timestamped data storage solution [46]. In fact, Cassandra is also known for being a good choice when it comes to timestamped data due to its sequential data storage [47]. Given that Cassandra was developed and built since the beginning with Big Data and distributed storage in mind, it comes as no surprise that it is one of the most popular Big Data storage systems for large corporations today being used by 40% of the Fortune 100 companies including eBay, Facebook, Apple, Instagram, Reddit, CERN and Netflix. Close behind Cassandra in terms of performance is MongoDB, which contains a strong database as a service commercial offering. Even though MongoDB does not seem to have a very attractive latency, it could be a viable option when database experts are lacking.

Cassandra is a distributed NoSQL database system, originally developed at Facebook. It was designed to satisfy the following requirements: full multi-master database replication, global availability at low latency, scaling out on commodity hardware, linear throughput increase with each additional processor, online load balancing and cluster growth, partitioned key-oriented queries and flexible schema. Cassandra is a master-less peer-to-peer system with no single point of failure. All data in Cassandra is associated to a Token. The nodes in a Cassandra cluster share the responsibility of data storage and

occupy themselves with data from a multiple range of token values. When new nodes are added, the token ranges are simply further partitioned and free token ranges are given to the new nodes. A Cassandra row is defined by a primary key which in turn is composed of one or multiple partition keys and one or multiple clustering keys. Partition keys, as the name suggests, determine the partition in which data is stored. The clustering key defines the node in which the partition is stored, Figure 2.16. The choice of columns to use to define these keys is essential as queries in Cassandra are very dependent on the columns that form the partition key and clustering key. Naturally, the primary key must be unique.

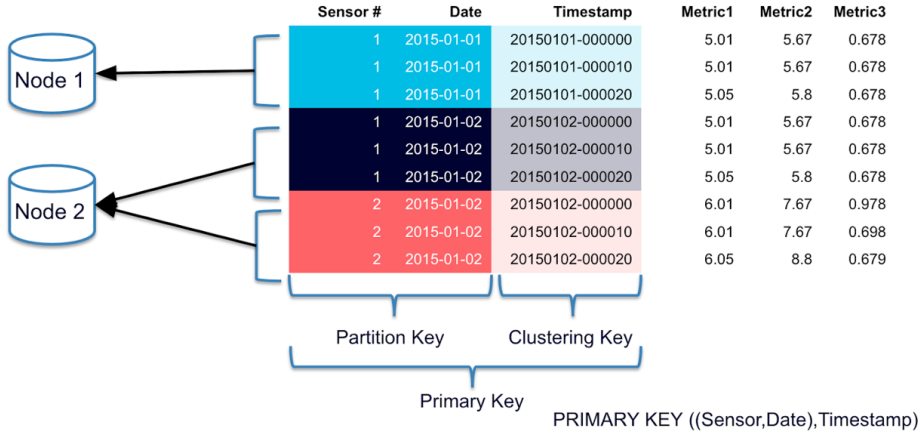


Figure 2.16: Cassandra primary key definition.

In the ZORRO context, it therefore makes sense to use the timestamp of the reading in the primary key, to ensure that no readings from different systems, but taken at the same time, coincide and overwrite each other. Data in a Cassandra system is divided into datacentres each containing multiple racks.

2.3.6 Deployment and Virtualisation

Having chosen the main technologies to be used in each one of the sub-systems, the next challenge is the choice of the technology deployment method. When it comes to deployment of Big Data information systems, or any modern application by that matter, the deployment paradigm has significantly changed in recent years. The first enterprise scaled applications were deployed on physical machines powerful enough to sustain the expected load. If the load were to increase, the solution consisted mainly in increasing the power of the machine. This rapidly became impracticable and unthinkable for Big Data use cases. The concept of Virtual Machines (VMs) arose following this, where the VMs would share physical resources and could more easily be horizontally scaled up and down. If we needed to scale an application, we would simply add more of the same type of machines on the hardware level (horizontally scaling) and spawn more VMs, Figure 2.17. The way software and applications were produced and coded naturally also changed around this new construct. Despite their attractiveness and flexibility, VM's still had limitations as they were relatively expensive to bring up and down and normally came with a fully-fledged operating systems as seen in Figure 2.17. Similarly, there was no clear bridge between local application development and deploying it in a VM. In fact, these two operations were generally done separately. In order to respond to the need for a more scalable and flexible deployment entity, and

also to bridge the gap between development and deployment, the concept of containers arose.

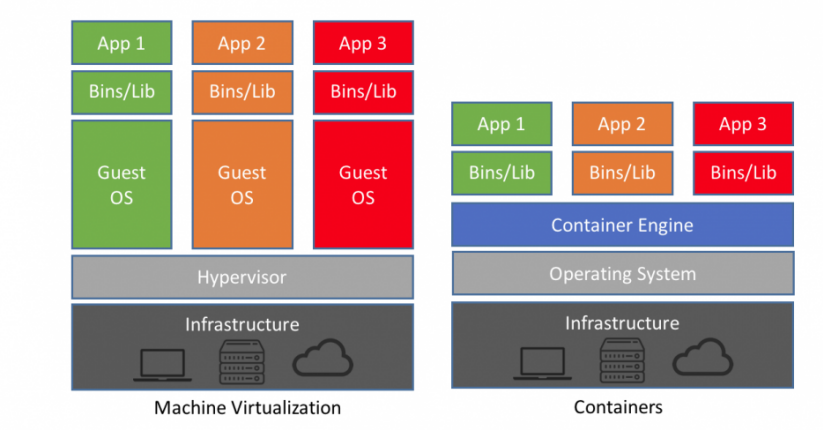


Figure 2.17: Virtual machines vs containers.

Containers can be described as isolated processes that contain a namespace (Bins/libs) and access to a limited amount of hardware resources (control group). Due to this, containers can be thought of as minimalist operating systems running an application (task) that is stateless. For applications to be scalable, services from these applications needed to be decoupled. This is the idea behind a micro-service like architecture. Containers allow for this as they are lightweight therefore making them easy to bring up and down. They are stateless making an application composed of containers more fault tolerant as each container is even more “disposable”. To create, deploy and manage containers, Docker is the go-to technology as they are the original creators of containers as we know them today [48]. Docker corresponds to the container engine in Figure 2.17.

Most modern-day applications which deal with Big Data sources and fast and varying demands have adopted a containerised deployment. The use of containers however is not only restricted to Big Data applications, having being used in embedded robotics systems as well due to their light weight and micro-service nature [49]. Figure 2.18 illustrates how the docker daemon works.

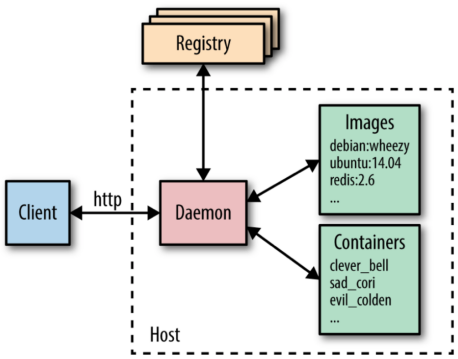


Figure 2.18: Docker daemon [50].

The client (user) communicates with the docker daemon demanding that a certain container be run or built. In the case of running a container, the docker daemon downloads the container image from a container registry and runs it on the host operating system. A container image is simply the build-time

construct of a container and a container instance is the run-time construct of a container. Docker is responsible for managing resource distribution across containers.

Docker is an extremely useful tool to deploy containers however, for large scale environments, tools such as docker-swarm [51] to manage containerised deployments distributed across many machines (cluster) is required. In fact, this concept and technology was so widely used in organisations like Google that they internally developed and open-sourced a container cluster manager solution called Kubernetes [52]. Kubernetes is the next step to docker as it allows for the management of multiple containerised systems in different machines. It is for this reason that Docker alone cannot be considered a final solution for the robust deployment of the ZORRO system.

2.3.7 Data Architectures

So far, we have talked about generic data system architectures and individual technologies that make up these systems. However, when it comes to Big Data specific architectures, there are different architectures that can be built depending on the processing modes and use cases required. Three popular Big Data architectures are Lambda, Kappa and SMACK.

The Lambda architecture [53] is a data processing architecture designed to handle massive amounts of data using both batch data processing and stream data processing. For batch processing scenarios, data is stored in a Big Data storage system (Data Lake), and then used in what is known as the serving layer, involving distributed querying interfaces and insight extraction tools (Figure 2.19). In the streaming layer, data is ingested in real-time and likewise queried in real-time. This layer is associate to use-cases where the architecture is event-driven, meaning that actions need to be taken in real-time based on events that occur.

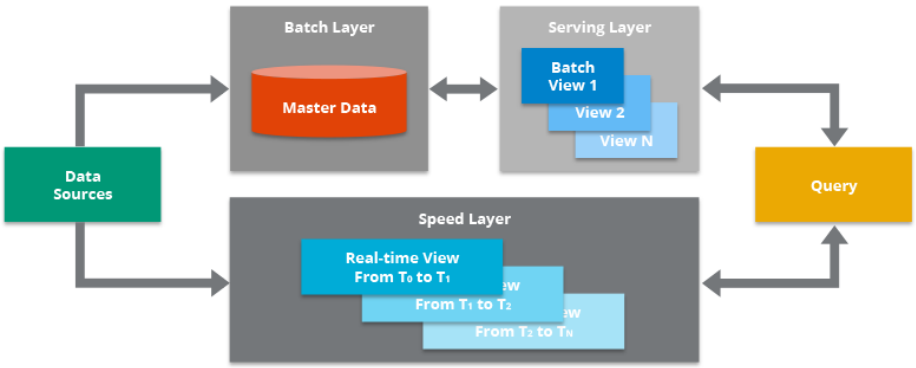


Figure 2.19: Lambda architecture – concept [54].

The Kappa architecture [55] was created by Jay Kreps based on his experiences at LinkedIn and his feedback from the Lambda architecture [56]. The KAPPA architecture is a simplification of the Lambda architecture but without a batch layer (Figure 2.20). The Kappa architecture focuses on stream processing but can also perform ”batch” processes by using MOMs such as Kafka, which persist the data into disk. The main advantage of the Kappa architecture is that the code and the stack it runs on is the same

for both streaming and historical batch processing. This is not necessarily the case for the Lambda architecture.

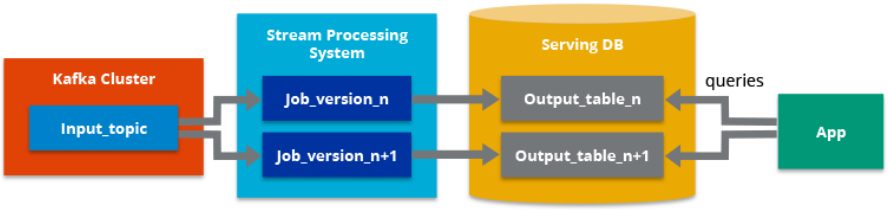


Figure 2.20: Kappa architecture – concept [55].

The SMACK architecture stands for S - Spark, M - Mesos, A - Akka, C - Cassandra and K - Kafka. The SMACK architecture was produced to respond to the new Web and general application paradigm. In a modern, always connected and data-driven world, technologies need to be able to quickly ingest data at a large scale with fault-tolerance, process this data in real-time, perform actions based on the processed results and store data and results in persistent databases. The SMACK architecture was conceived with these objectives in mind. Although the SMACK architecture uses Spark, Kafka and Cassandra, which are all promising Big Data technologies, it is locked to specific technologies and is therefore not adapted for a modular and flexible ZORRO solution. The architecture uses, for example, the Mesos cluster manager. This is a viable technology but a replacement to the also very popular Kubernetes cluster manager.

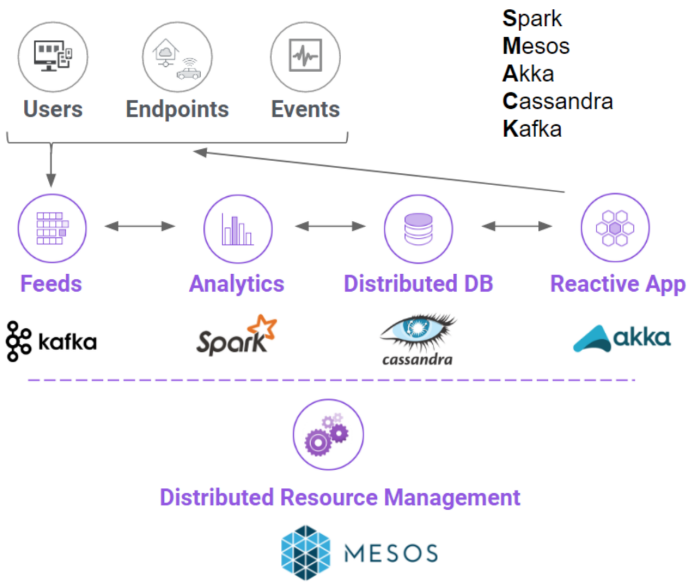


Figure 2.21: SMACK architecture – concept [57].

In ZORRO, we have decided to use the Lambda architecture as the generic, distributed, scalable, and fault-tolerant Big Data processing architecture as it allows for both batch and streaming data processing. Batch is required in ZORRO for both machine learning model training (which the KAPPA architecture does not offer) and creation of a knowledge base of manufacturing data. The streaming layer is essen-

tial for real-time data processing, anomaly detection and event driven architectures. The results and resources from both these processing layers must be available and accessible in what is known as the serving layer. After a state-of-the art study on different types of lambda architectures and the cloud provider's implementations of these architectures, Figure 2.22 was produced to reflect the architecture desired for ZORRO.

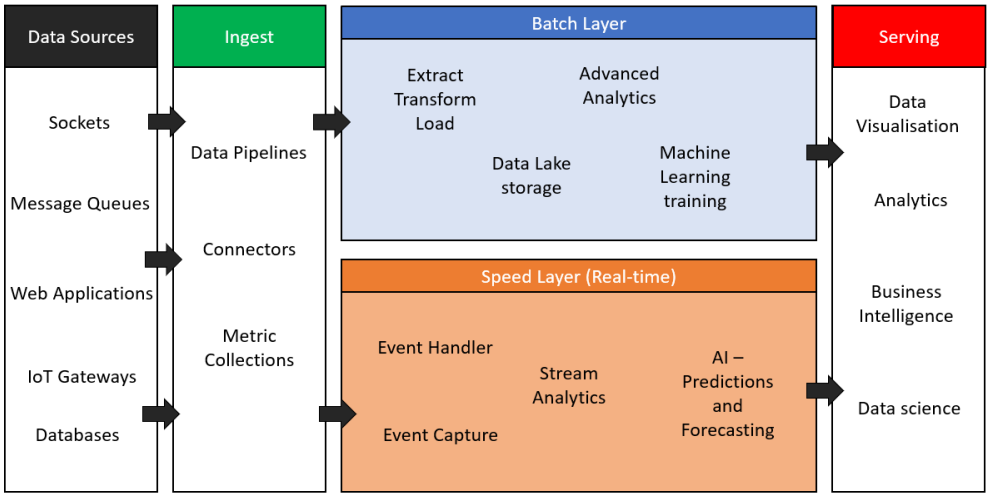


Figure 2.22: Lambda architecture – chosen Big Data architecture for ZORRO.

2.4 Automatic Anomaly Detection

As seen so far, Industry 4.0 systems process data and detect events to perform a series of actions: anomaly detection, predictive maintenance, supply chain resilience, machine control etc. A majority of these events are inferred from the numerous sensors present in the production environments and producing data in large quantities in real-time. In order to process this data at the desired speed, and in order to be able to apply the actions above mentioned, automatic and intelligent algorithms are required. The algorithms must be automatic in order to achieve the desired speeds. Intelligent algorithms, (containing Artificial Intelligence) are required in order to be able to identify events which can be function of very complex interactions and relationships from the measured sensor values. In the case of equipment anomaly detection for example, the idea is to be able to understand the dynamics of the machines when they behave correctly and be then able to tell when the behaviour is deviating from this normal value.

Artificial intelligence (AI) refers to intelligent like behaviour demonstrated by machines. As seen in section 2.3.1, AI has a lot of potential in event driven architectures as it can provide crucial insights and decision criteria in response to events. Most importantly, it can perform this task quickly and in a scalable way. According to McKinsey [58], 50% of companies that embrace AI will double their cash-flow in 5-7 years. In an IIoT context, AI can be used for general forecasting, supply chain resilience and anomaly detection. The actions generated by these algorithms can be simple alerts or control actions on the data and IoT system. In the context of the ZORRO project, the first mission is to be able to detect production

anomalies in real-time. This is done through a branch of AI called Machine Learning (ML).

In ML, computer algorithms learn and improve from experience without being explicitly programmed [59]. ML algorithms can learn from data, make sense of data or automatically learn through experience. There are three main branches of ML: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, the algorithm uses data from a dataset (training dataset) containing input-label pairs $(x_1, y_1), ...(x_n, y_n)$, with input $x_n \in X$, being p quantitative or qualitative variables, and observed labels (outputs) $y_n \in Y$, also quantitative or qualitative, to create a model $\hat{f}$ such that:

$$Y = \hat{f}(X) + \epsilon \tag{2.1}$$

Where ϵ represents the measurement noise or error made by the model. The objective of the model $\hat{f}$ is therefore to attempt to explain the variable Y based on the observed features, X. The quality of the model $\hat{f}$ for observation k, is determined by a loss function $l(y_k, \hat{f}(x_k))$, which measures the discrepancy between the model prediction $\hat{f}(x_k)$ and the observed value y_k. Therefore $\hat{f}$ is chosen such that $\frac{1}{n}\sum_{k=1}^{n} l(y_k, \hat{f}(x_k))$, the average loss across the training data set, is minimised. One thing to keep in mind during the training of a supervised learning algorithm is that trying to reduce by too much the error during training can be ineffective and lead to a phenomenon called over-fitting. When a model is over-fitted, it behaves very well on the training dataset but has very bad performance when acting on a never seen before dataset. Over-fitting, when Y is a one-dimensional continuous variable, is illustrated in Figure 2.23 where $\hat{f}$ is given by the orange curve. This same phenomenon also occurs in classification problems when Y is a qualitative variable. To avoid over-fitting, a balance between model complexity and training error must be achieved.

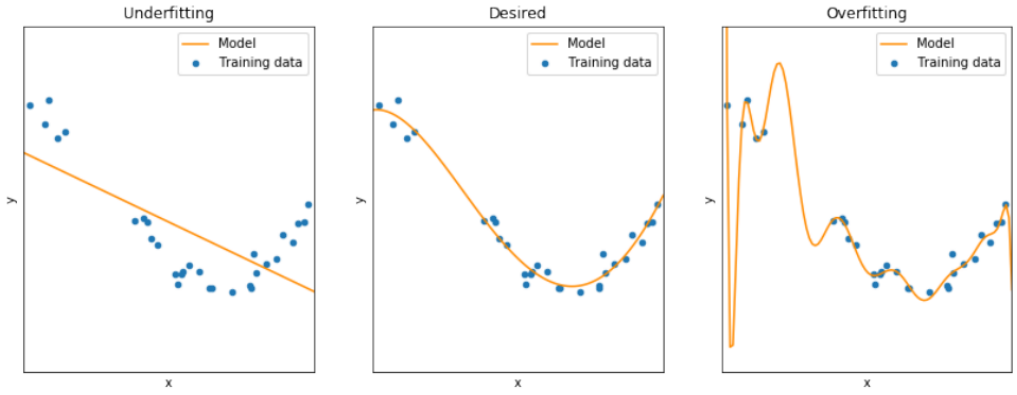


Figure 2.23: Regression over-fitting concept.

In unsupervised learning, there are no inputs, outputs or pre-existing labels, and the ML algorithm attempts to find previously undetected patterns or groups (clusters) in the data. In reinforcement learning, the algorithm adapts its behaviour and learns as it acts in the environment based on some sort of cumulative reward or penalisation.

Given that automatic anomaly detection is one of the main objectives of the ZORRO project, ML models that could effectively detect anomalies were studied. An initial naive approach was to study

supervised learning models that could be used for binary classification (anomaly 1 or no anomaly 0). This corresponds to the case where the variable y, introduced above, can only assume two values: $y \in 0, 1$.

Focusing on the algorithms offered directly by Apache Spark's ML library, we chose to study the following algorithms: Decision Tree, Random Forest, Logistic Regression and Support Vector Machine (SVM). All these algorithms are methods used for defining the function $\hat{f}$ described above. Realising the limitations of this approach, other unsupervised learning algorithms for anomaly detection were studied. In this section we will briefly, and at a high level, describe how these models work and are built. The description of the algorithms is purposely kept at a high level, but more formal documentation was separately done on these algorithms.

2.4.1 Decision Tree

Decision trees [60] are tree like structures that divide data into groups given criterion based on features of the data. Each division applied to a node creates two further nodes. The features can be both qualitative (was the machine checked in the past month – yes/no) or quantitative (temperature ¿ 20°). For quantitative divisions, a value is fixed whilst for qualitative feature divisions, one of the possible classes is chosen.

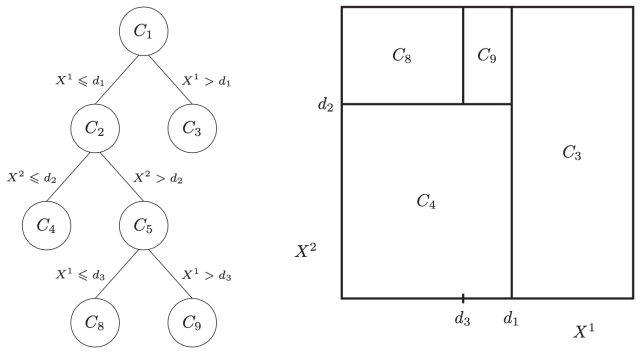


Figure 2.24: Two-dimensional concept of decision trees with quantitative features [60].

Figure 2.24 illustrates a 2D example of a decision tree constructed from quantitative features. The result of a decision tree are childless nodes which form groups in the data division. Decision trees can be used in binary classification by associating to each group one of the two possible outcomes. The criteria to associate a class to a group can vary but is generally made so that the group is associated to the mode of the class of that group. In our anomaly detection scenario, if a new data point arrives and we wish to assess if it is an anomaly or not, we simply see the group to which it falls, by travelling down the tree, and label it (make a prediction) equal to the group's associated label. Decision trees have multiple hyper parameters but one of the most important parameters is tree maximum depth. As the name suggests, the tree depth defines the number of layers, m, a tree can have and therefore also puts a limit on the number of divisions it can have $(2m - 1)$. Increasing tree depth makes the model capable of making finer and more complex prediction topologies but also exposes the model to becoming over-

fitted to the data. In a decision tree we choose the next division based on the division that will maximise the purity of the two nodes created. Mathematically, this is done by minimising the impurity of a split. Decision trees have the disadvantage however of being sensitive to over-fitting. Moving an individual point can sometimes drastically change the form of the decision tree.

2.4.2 Random Forest

Random forest is a Supervised Learning bagging ensemble algorithm used for classification and regression. Bagging algorithms use multiple weak learners, trained with random samples of the original dataset, to form one robust learner. In random forest, the weak learners are decision trees as studied above. In classification, the algorithm therefore operates by constructing a multitude of decision trees at training time and when making a prediction, it outputs the class (prediction) that is the mode of the classifications (predictions) given by the decision trees (majority vote). The advantage of Random Forest in comparison to the Decision tree is that it is less susceptible to over-fitting. The random forest algorithm is therefore an aggregation of decision trees but where in each decision tree:

- The number of features that can be used to make a split at each node is limited to some percentage of the total. This ensures that the ensemble model does not rely too heavily on any individual feature and makes fair use of all potentially predictive features.
- Each tree is built on a random sample from the original data set when generating its splits, adding a further element of randomness that prevents over-fitting.

Random forests successfully convert the high variance predictions (over-fitting) that are normally associated to individual decision trees into low variance predictions by using majority votes. Taking the example of Figure 2.25, the chosen predictions would be 1, as it wins the majority vote. The larger the forest, the more robust the algorithm, however, the longer the training and prediction will take.

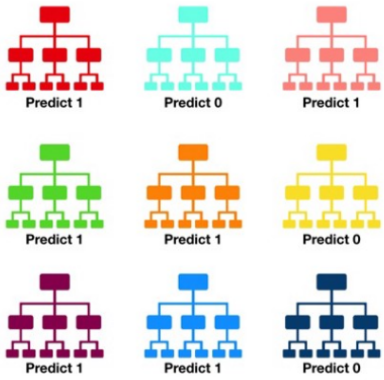


Figure 2.25: Illustration of a nine-decision tree random forest.

2.4.3 Logistic Regression

Logistic regression is an inherently binary classifier. The objective of Logistic Regression is adapted to attempting to model or describe the probabilities $P(Y = 1)$ and $P(Y = 0) = 1 - P(Y = 1)$ rather than

predicting the class Y directly. We wish to predict these probabilities based on an observed value X. This is given by $Pr(Y = 1|X) = f(X)$. Logistic regression achieves this purpose by using a logistic function. A popular choice is the logit function (our $\hat{f}(X)$)). Notice that in this case, $\hat{f}(X)$ is predicting a probability and not directly the class. The goal is therefore fit $\hat{f}(x)$ such that it is as close to 1 as possible for all $x \in X$ such that $y = 1$ and so that $\hat{f}(x)$ is as close to 0 as possible for all $x \in X$ such that $y = 0$. Visually, this corresponds to fitting a sigmoid like curve to the data as show in the one-dimensional example $(p = 1)$, in Figure 2.26.

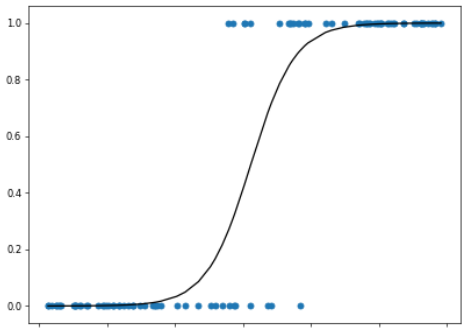


Figure 2.26: Logistic Regression one dimensional example [61].

In order to predict the class, a naive criterion would be to predict $Y = 1$ if $\hat{f}(X) > 0.5$. In other words, if a point arrives and we predict a probability that $Y = 1$ for that point that is greater than 0.5, then we predict that that point corresponds indeed to an observation where $Y = 1$. This can of course be adapted depending on the use case of the algorithm.

2.4.4 Support Vector Machine

The Support Vector Machine algorithm (SVM) [62] is, in a nutshell, an algorithm that aims in optimally separating classed points with a hyper plain in the linear case, or with another function in the general case. An optimal separation is one that perfectly separates two classes or at least does so with the minimum amount of error possible.

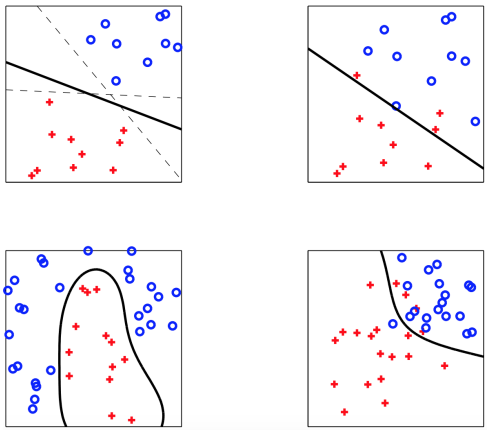


Figure 2.27: Illustration of SVM concept optimally separating two classes [62].

As can be seen in Figure 2.27, multiple curves can be used for defining the separation of classes.

The SVM problem therefore implies finding the function f, such that the optimal division is obtained. It is up to the Engineer to choose the type of curve used. As shown in Figure 2.27, a linear curve can in some cases not be enough to decently separate the two classes. One would be therefore tempted to increase the complexity of the curve in order to obtain the optimal separation. This is not necessarily desirable as very complex curves can lead to the over fitting phenomenon. In the linear case, if the problem is linearly separable, the problem is posed as the maximisation of the margin. Where the margin is the shortest distance of a point to the curve. When the problem is not linearly separable, the problem is renormalized by associating a penalty to each point that is not correctly separated and the goal is now to minimise the sum of these penalties. Apache Spark's ML library only provides support out of the box for linear SVM binary classifier. To make a prediction, we simply first see in which side of the hyper plain a point falls and predict the label associated to that side of the division.

2.4.5 Artificial Neural Network - Auto Encoder

The disadvantage of supervised ML algorithms in anomaly detection is that we need to supply the algorithm with data from both correct (non-anomaly) like behaviour and incorrect (anomaly) like behaviour. The danger here is that if a different type of anomaly, that was not initially "taught" to the algorithm, appears, there is a risk that the algorithm will provide an incorrect prediction. It is for this reason that unsupervised learning algorithms for anomaly detection were studied. From this study, one of the algorithms that stood out was the auto-encoder artificial neural network having been used in [63] and [64] for anomaly detection and shown to be efficient for time series anomaly detection.

The aim of the auto encoder network is to learn an efficient encoding of the system to monitor. The encodings are a form of dimensionality reduction. An example of a popular form of dimensionality reduction is the Primary Component Analysis (PCA) [65] which can also be useful for data exploration / visualisation purposes. The advantage of the auto encoder with regards to PCA is that nonlinear and therefore more complex interactions can be modelled and encoded. It contains an input layer, an output layer, and intermediate hidden layers. These layers are formed by nodes. The nodes of each layer are connected to the nodes of the precedent layers and to the nodes of the following layer. The output of node j in layer i is here represented by $z_{i,j}$. For each node j in each layer i, the outputs from the k nodes of the precedent $i-1$ layer are used to define the output of this node, $z_{i,j}$, such that:

$$z_{i,j} = \sigma\left(\sum_{k} w_{i,j,k} \cdot z_{i-1,k}\right) \tag{2.2}$$

where $w_{i,j,k}$ is the weight present in node j of the layer i for its connection to the node k of the precedent layer, Figure 2.28. This weight represents the importance of the connection. The function σ is a nonlinear normalisation function (sigmoid in shape). It is this function that allows the network to model non-linear interactions. The output of node k from layer $i-1$ is simply $z_{i-1,k}$.

The auto encoder is a neural network with the number of input nodes equal to the number of output nodes and with one or more hidden layers with less nodes than the input and output layers. The auto-

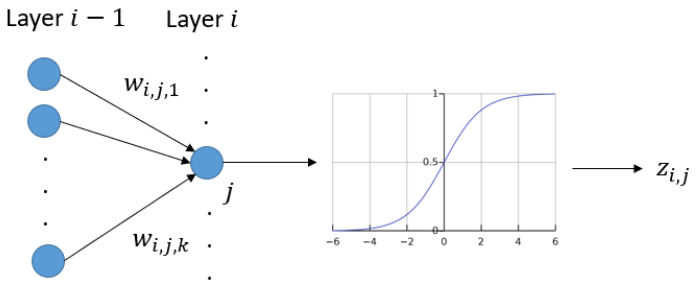


Figure 2.28: Inner workings of an artificial neural network neuron.

encoder attempts to compress the data with a minimal amount of information loss and then decompress it again in the output. Figure 2.29 illustrates an example of an auto-encoder used in the reconstruction of an image. The consequence is therefore a network that has learned an efficient encoding of the system. The network is built during training by tuning the weights of the nodes, $w_{i,j,k}$, so that the reconstruction error is minimised. In other words, we want the input to be as close as possible to the output of the network.

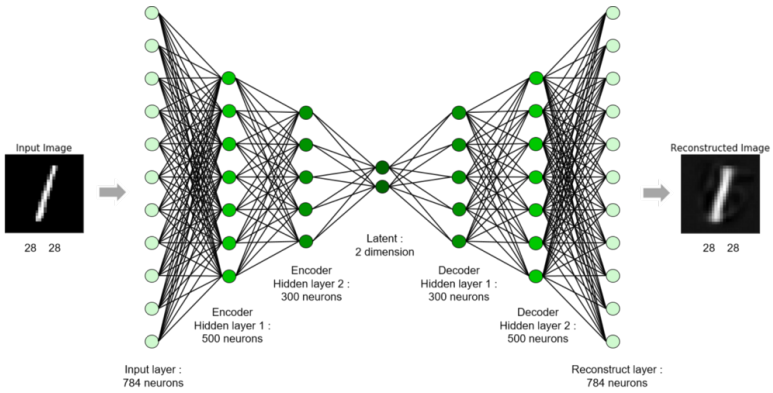


Figure 2.29: Auto-encoder example for image reconstruction [66].

2.5 State-of-the-art and Background Conclusions

At the beginning of this chapter, the context of this work and an overview of Industry 4.0 was done. The sections that followed were a summary of the documentation produced for AKKA which was deemed relevant in contributing to AKKA's ability to intervene in the implementation of Industry 4.0 projects. Not only were the theoretical concepts about the components that an industry 4.0 infrastructure should contain presented, but a description on how they can be used as final products in an Industry 4.0 environment was given. The definition of these technologies and architectures allows AKKA to concentrate its future efforts to explore technologies and approaches and systems in a limited set. The same can be concluded to the Data Science and Machine Learning applications to anomaly detection where various algorithms and methods were explored.

Chapter 3

Implementation of an Industry 4.0 Data Platform

In this chapter we will describe the engineering work done in terms of the implementations developed and the methodology that was followed to carry it out. The structure of this chapter is very much coincident with the methodology that was followed in this work, the V-cycle development method. We state the objectives of the engineering work performed in section 3.1, describe the methodology followed in section 3.2 and perform a system requirements analysis in section 3.3. The sections that follow are dedicated to the different architectural and design implementations that were done.

3.1 Objectives

As described in section 1.2, the mission of this work is to generate know-how for AKKA regarding Big Data, Industrial Internet of Things and intelligent data systems. This was done through contribution to project ZORRO, itself having a concrete objective which is to develop a Big Data Industry 4.0 platform capable of detecting production anomalies in real-time and generating alerts and reports. The objective of the technical / engineering work done was therefore to develop a system capable of doing this whilst also being capable of satisfying the constraints associated to Big Data Industry 4.0 systems.

Although automatic anomaly detection was a very concrete objective of the project, throughout the conception of the ZORRO solution, a constant attempt was made to try to make the solution as robust as possible foreseeing that after the anomaly detection capabilities, other objectives would be desired such as analytics platforms, general forecasting, predictive maintenance etc. It is for this reason, that even though the objective was fixed, we still attempted to build the foundations for a system capable of delivering the maximum number of Industry 4.0 advantages as possible.

When faced with this mission, a high-level plan was outlined. The first objective was to understand what features, architectures and technologies Industrial IoT Big Data systems capable of detecting anomalies contain. This was done through performing an extensive state-of-the-art study on both solution architectures and technologies used to build them. The results from this stage are summarised

in chapter 2. Having this done, a system requirement analysis was performed in order to approach the project with a system engineering mindset respecting the V-cycle development. Technical objectives were defined and a plan was developed and executed based on the comparison between system objectives and the current state of the project. With the implementation and improvement objectives completed, tests were done on the developed system to better understand its performance, potential and limitations.

3.2 Methodology

The adopted project methodology was largely based on the V-cycle methodology as, for each subsystem, there was a constant validation and verification mindset. The initial system requirements were outlined to determined if the right system was made, relating to the validation aspect (left hand side of Figure 3.1). Performance tests were done to confirm that the system was made correctly, relating to the verification aspect. This methodology aided in guaranteeing that the correct know-how was developed and that a mature Meta solution was developed.

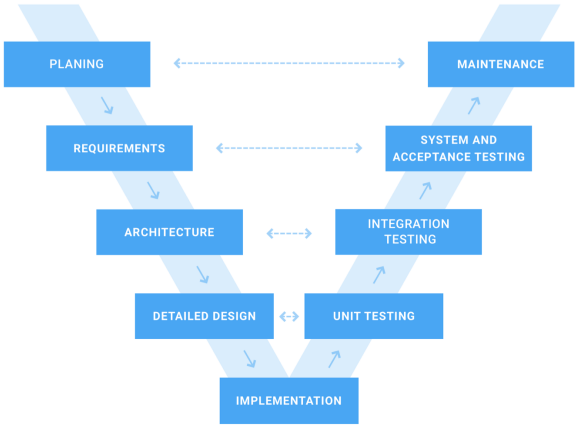


Figure 3.1: Initial V-cycle approach to system conception.

The project, however, also followed an agile method as every two weeks, a meeting with the project manager was done and whose feedback meant a loop like improvement on all the major steps including design, testing and deployment. This was useful not only due to the project manger's technical experience, but also product verification and validation know-how. Coinciding with these meetings every two weeks was a more granular definition of the tasks to be performed using the Microsoft Teams tool.

3.3 System Requirement Analysis

Inspired by the state-of-the-art study and by previous work done on the project, the ZORRO system was first divided into multiple subsystems as already seen in Figure 2.10. The requirements of each one of these subsystems were defined as follows:

1. **Overall system requirements**: requirements to be met by the final system (therefore all its sub-systems).
   (a) The system must be horizontally scalable, meaning that if the amount of data to process from one day to another doubles, then the solution can roughly double the instances/machines of each technology to respond to the increase in load.
   (b) The system must be fault tolerant. This implies that if any sub-component or entity of each system fails, then the whole system should continue to work. With respect to data, this means that if one of these sub-components or entities fails, no data is lost.
   (c) The system architecture should be based on microservices to render it more horizontally scalable, more adaptable to modern deployment methods (containers) and in order to reduce inter sub-system dependencies.
   (d) The architecture must be generic so that it can easily be adapted to fit future customer requirements and use cases. This requirement is coupled with the microservices requirement and the concept of building a Meta solution.
   (e) Scaling should be flexible. The system can scale up and down without many issues.
   (f) All elements of the system must be open source technologies.
   (g) The system must be easily built on commodity hardware.
   (h) All technologies should allow for encrypted communication with Transport Layer Security.
2. **IoT data collection and pipeline system**: responsible for collecting IoT type data and structuring it for delivery to endpoint systems.
   (a) The system must provide a tuneable message latency, meaning that if a message latency of milliseconds is desired, then the system can simply be horizontally scaled or configured to achieve that delay.
   (b) The system should provide a delivery guaranteed policy. If not required, the system should be configurable to not guarantee this in exchange for better performance.
   (c) The system must provide a tuneable message throughput, allowing for a trade-off between message throughput and latency.
   (d) The system should be compatible with IoT based message production deployments.
3. **Big Data Storage system (Data Lake)**: responsible for creating a persistent source of truth and for data processing engines to train machine learning models.
   (a) The system must provide an API to interact with the data and extract insights from it in a relatively quick and efficient manner. For example, count, aggregations, min, max and mean functions.
   (b) The system must provide time series related tools and operations.
   (c) The system must allow for distributed querying (distributed system).

4. **Data Processing and Machine Learning systems**: responsible for analytics, data processing and machine learning.
   (a) The system must allow for the processing of data in real-time (stream processing).
   (b) Must allow for batch processing of data.
   (c) Must provide a distributed architecture for performing data analysis to provide results with a low/tuneable latency.
   (d) Must provide a programming interface to develop and use Machine Learning libraries for data processing, anomaly detection, forecasting and data exploration tasks.
5. **Data Visualisation system**: responsible for dynamic dashboarding and metric monitoring.
   (a) The system should allow for labelled filter-based querying.
   (b) Must provide dashboarding tools that display data in real-time as it arrives.
   (c) Must be able to collect metrics from different resources and with different protocols.

When implementing the solution, a constant loopback on these requirements was made to make sure that the developed product satisfied these initially defined requirements (V-cycle validation). These requirements were also used as a tool to improve the system that was already in place. Although the state-of-the-art section does not go into detail about all the inner details of the technologies used in this project, a consistent and detailed documentation of the state-of-the-art technologies was done in order to later confirm that the developed solution in fact satisfied these system requirements.

3.4 Analysis of the Previous System

Following the system requirement analysis, the next step was to check if the current ZORRO system satisfied the system requirements defined. The previous internship done on the ZORRO project concluded with the following desired system architecture, illustrated in Figure 3.2.

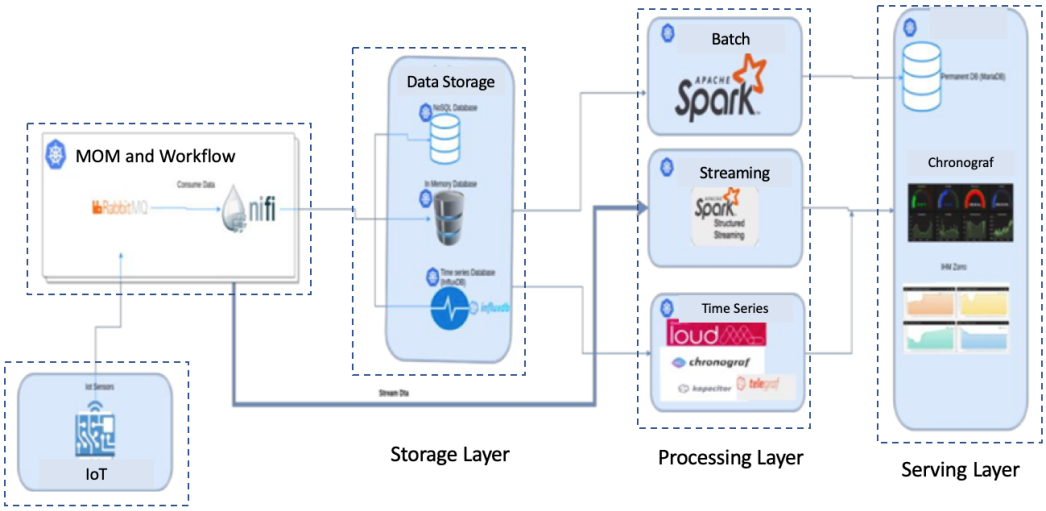


Figure 3.2: Proposed system in previous internship.

We identify the five primary layers: the IoT data production layer (bottom left); followed by the data collection and pipeline system (MOM on the centre left); the data storage layer; the data processing layer composed of three sub layers, batch, streaming and time series); finally the serving layer containing result aggregations and data visualisation tools. The symbol on the top left of the entities is the Kubernetes container cluster management system.

It is important to notice that although this was the proposed architecture, not all features were deployed and tested. The aspects that were missing involved mainly the deployment of Spark (data processing) system which was used locally (not in a cluster) and interacting with static data sources (csv). Similarly, Apache Nifi and RabbitMQ were deployed in virtual machines but not in containerised form (using docker). The NoSQL and in Memory Databases were developed by a previous intern and deployed in a cloud provider. Although a lot of work was done to study the feasibility and interest of deploying the architecture using Kubernetes, the deployment itself was not carried out. Finally, the communication protocol throughout the whole stack was based on InfluxDB's line protocol.

Based on the current system status, and the requirements analysis, the following objectives were defined:

- Deploy missing systems on AKKA's local cloud and in containerised form (Docker) to obtain a reproducible Meta architecture.
- Optimise data storage system given that technologies have changed (no need for in Memory DB due to Spark structured streaming).
- Continue working towards Kubernetes deployment of the system (through dockerisation).
- Test the possibility of replacing RabbitMQ (and Apache Nifi) by another message broker - Kafka, motivated by the state-of-the-art study.
- Deploy a data scientist, machine learning and analytics development tool which is currently not present in the serving layer.
- Apply and deploy the lambda architecture using automatic anomaly detection as the use case of the online machine learning (ML) algorithms (batch and stream processing with deployed ML).
- Change the communication protocol to a more internationally used one in order to reduce inter-system dependency and increase modularity of the architecture.

All the above objectives were defined bearing in mind the system requirements defined in section 3.3. In the following sections, we will describe the work done on each of the 4 primary subsystems described and how the above objectives were accomplished.

In order to simulate the ZORRO data flows, described in section 2.1.2, previous collaborators of the ZORRO project have developed a factory simulator. This is a Python based simulator which uses multi-threading and queue libraries to simulate a network of IoT devices producing data. This script therefore simulates what has been called thus far the IoT layer.

In a mass production context, multiple production lines will be present performing the same or different transformations on products. Each production line is composed of stages which are logical groupings of sub-transformations that are applied on products. To perform these transformations on the products, the stages can contain one or multiple equipments (machines). At instance t, the simulator contains

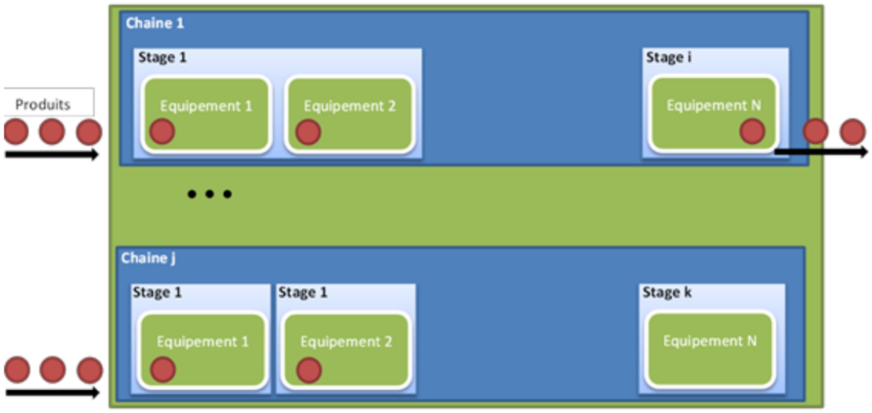


Figure 3.3: Factory simulator concept.

Sensors	Product	Product Surrounding	Equipment	Equipment Surrounding
Temperature	✓	✓	✓	✓
Humidity		✓	✓	✓
Vibration			✓	
Brightness	✓			
Dust	✓		✓	✓
Electric Consumption			✓	

Table 3.1: ZORRO Properties measured.

the state of the products and of the equipment. This state is defined by on entity properties and entity environment properties. A subset of these properties is summarised in Table 3.1.

The simulator is equipped with artificial anomaly generators. Essentially, each one of the readings of table 3.1 could have been individually configured to have some sort of anomaly, notably: Variable collective, constant collective and global anomalies. Constant collective involves a sensor that temporarily freezes at a given high point, in variable collective a sensor value that changes suddenly in mean and global that randomly peaks significantly. This tool is what was used to simulate faulty equipment behaviour. All the described data flows are required for real-time monitoring and visualisation and for long term persistent storage. In addition, the equipment data streams (on equipment and equipment surrounding), will be used for equipment anomaly detection.

3.5 Architectural Proposal

As a crucial stage in the V-cycle, the system architecture needed to be defined. The proposed architecture was a product of the state-of-the-art study, system requirements analysis and the analysis of the previous system. The proposed architecture is illustrated in Figure 3.4.

The architecture of Figure 3.4 is similar to that of Figure 3.2 but with the substitution of RabbitMQ and Nifi by Kafka as the central message pipeline, the removal of the in-memory database and the addition of a development platform to interact with the data processing system (requirement 4.d). Querying interfaces were also added to interact with the raw data present in the data lake (requirement 3.a). The

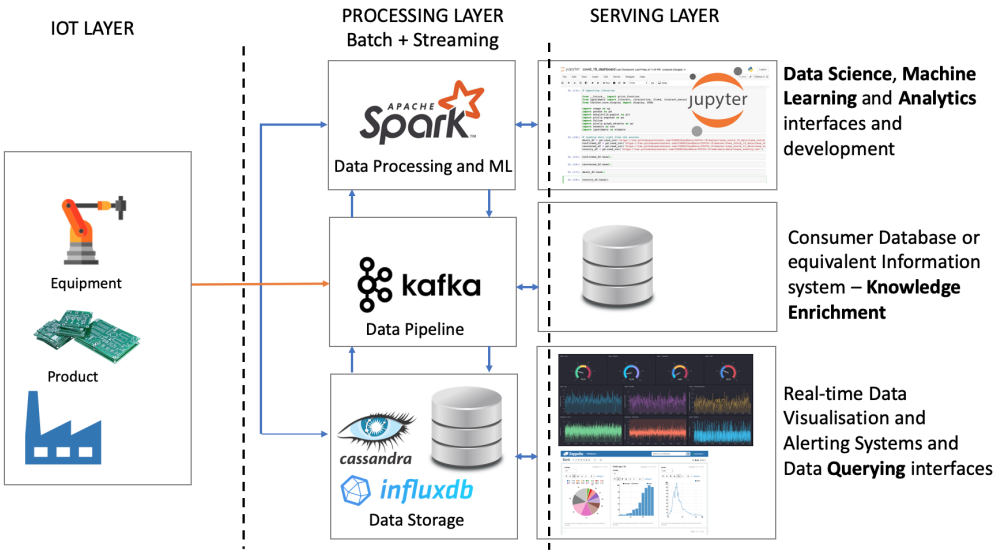


Figure 3.4: Proposed architecture.

Lambda architecture can clearly be seen as the data processing engine is connected both to Kafka, for stream processing, and Cassandra, for batch processing (requirements 4.a and 4.b).

3.6 Deployment Strategy

Before starting to build the new proposed architecture, the deployment and virtualisation strategy was defined so that a consistent deployment model was follow throughout the project. This strategy was followed in almost all of the sub-systems. As of the previous internship related to the ZORRO project, the final objective of ZORRO, with regards to deployment, was set to use Kubernetes (k8s) [52] as the cluster deployment solution. This choice is very consistent with the system's requirements as k8s and containers combine to form a robust microservice, fault tolerant and scalable system. k8s manages deployments of containers and therefore all elements of the ZORRO system must be in containerised form before they can be deployed using k8s. As a container deployment cluster manager, k8s works by defining a desired container state.

Therefore, in this work, after using and testing a technology, the next step was to containerise it and define a sub-system container state. To do this, either the official docker container images were used or custom containers were built, using docker, to perform ZORRO specific tasks. Another advantage of containerising an architecture is that it allows for the easy version control and storage in container registries of each sub-system.

To work towards the final k8s objective, all containerised deployments in ZORRO were defined using a tool from docker known as docker-compose [67]. The idea behind docker-compose is to define a desired container deployment (state) for one host OS through a configuration .yaml file. This file can be used to define containers (microservices), their configurations and the environment variables to be passed to them. It can also be used to define the networks the containers are in and the volumes they are connected to. This is much more desirable than declaring, configuring and creating each one of

these docker “objects” (containers, volumes and networks) individually. Once the docker-compose file is written, it can be sent to the docker daemon in a single command. This tool is very similar to k8s in the way that it defines a deployment state. An illustration of how a docker-compose file is used is given in Figure 3.5. As described in the figure, the docker daemon reads the contents of the docker-compose file, obtains the containers from a registry (container repository), and deploys them on the host operating system (Host OS). The docker daemon has access to machine hardware resources and distributes these across the containers. Notice the networking capabilities of the docker-compose files as they can define multiple networks and which containers belong to which networks. This can be extremely useful for microservice architecture security and process isolation. Given that containers are stateless entities, connections to external data volumes are many times required and can also be defined and created in a docker-compose file. Overall this file allows for the creation and definition of a containerised architecture containing all the main docker and container constructs.

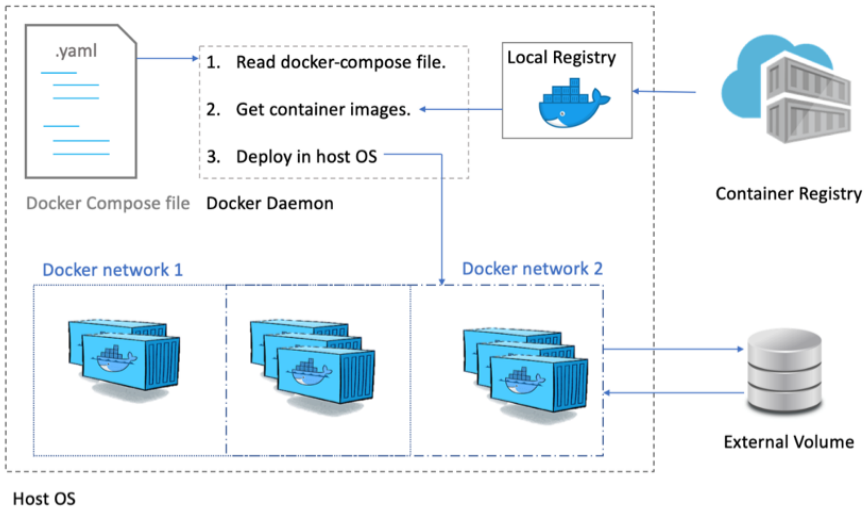


Figure 3.5: Docker-compose idea and inner workings.

The use of docker-compose on its own naturally has its limitations. For example, it can only be used to deploy containers on one host (one machine). To deploy on multiple hosts, docker-swarm [51] is required. This is docker’s equivalent of k8s. Nevertheless, the type of “objects” and processes done when defining a docker-compose file are like those defined when creating a k8s deployment. Therefore, it brings the architecture and deployment method one step closer to the k8s objective. All machines used in this internship were OpenStack virtual machines [68], from AKKA’s internal cloud. These machines were of type either 1 or 2 as shown in table 3.2. These machines belonged all to the same network with 1.5 GB/s of bandwidth.

Machine Type	OS	nº CPUs	RAM (GB)
Type 1	Linux, Ubuntu	2	4
Type 2	Linux, Ubuntu	2	6

Table 3.2: OpenStack machine types available.

3.7 Developed Proposed System and Meta Architecture

Having defined the system requirements, the desired architecture and the deployment strategy, the next step was to develop the system. By compiling the developed sub-systems and all their connections with each other, we obtain the system illustrated by Figure 3.6. This was the final detailed system architecture of this work. The architectural proposal described in section 3.5 and illustrated in figure 3.4 can be recognised. Each box represents a commodity hardware machine that was used either of Type 1 or of Type 2 (table 3.2). A docker symbol is included in the top left corner when all the deployed technologies and tools were containerised for that machine. The developed system was designed to reflect similar Big Data, stream processing and IoT data systems studied in the state-of-the-art, having a clear microservice based architecture where each sub-system satisfies a particular purpose and can be scaled independently of the others. All technologies used to build the core of the system are industry used Big Data technologies guaranteeing scalability and fault-tolerance. In addition, all the elements of the developed architecture (languages, technologies and cloud infrastructure) are open source. The system is modular as the technologies chosen can be integrated with other technological choices different from the ones used in this work. All these aspects marked the completion of requirements 1.a to 1.g.

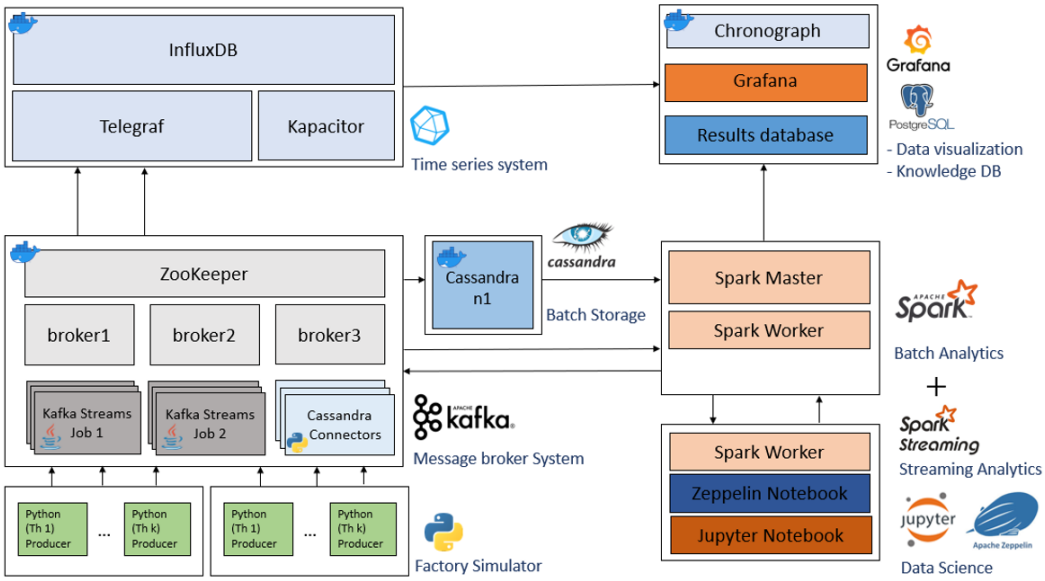


Figure 3.6: Final ZORRO system architecture.

Allied to each architectural or technological feature / component, documentation was produced for AKKA in order to justify the design choices made, understand the various architectural and technological options that exist today to build these systems and understand the advantages, disadvantages and trade-offs of the available options. In the sections that follow the details of the deployment of each one of the sub-systems are described.

3.8 Message Brokering and Data Pipeline System

As seen in section 3.4, before this work, the technology used as the message broker was RabbitMQ [69]. Although this broker communicates with the AMQP protocol, which is suitable in an IoT layer, it has a series of limitations particularly concerning Big Data scalability [70], where Apache Kafka outperforms RabbitMQ. This was consistent with the state-of-the-art study as many sources confirmed Kafka as a reference solution for Big Data scale messaging pipeline.

Realising this, the first objective of the internship was to study the differences between these two message brokers and study the interest of developing a Kafka solution. The overall results of the study were the following:

- In terms of architecture, latency and throughput, a Big Data scale solution obtains better performances with Kafka than with RabbitMQ. Kafka particularly shines in terms of throughput [71].
- Message order in Kafka can only be guaranteed at the partition level. Whilst it is guaranteed at the queue level in RabbitMQ (a queue in RabbitMQ is the equivalent of a Kafka topic).
- The communication protocol of RabbitMQ, the Advanced Message Queuing Protocol (AMQP), is more compatible with IoT devices when compared to Kafka's binary encoded protocol and has support for the famous IoT protocol - MQ Telemetry Transport protocol (MQTT). This is mainly due to Kafka's producer API complexity and inability to handle poor network performance.
- Under high loads, RabbitMQ's performance can degrade fast if RAM memory is filled. The same is true for Kafka when the cache is full, but performance degradation is smaller [71].
- RabbitMQ can handle more complex message routing.
- Kafka consumers follow a data pull model avoiding the possible overloading of consumers with data.

In addition, Kafka has better integration with other Big Data technologies such as Apache Spark and Apache Cassandra. With regards to message ordering, not only has Kafka recently released new out of the box features to solve this issue [72], but messages can also be sorted through the insertion of a timestamp field on the JSON being sent. This is what was done in the case of this work. Overall, it was concluded from this study that Kafka is very interesting to have, on a Big Data scale, once the data is on a TCP communication level. Due to this, and because of its popularity and performance, different services such as dashboards and analytics engines integrate very well with Kafka. It was therefore decided that it would be interesting to integrate a Kafka solution into ZORRO.

In addition to the points discussed above, remembering one of AKKA's high-level objectives, generating know-how about the feasibility of a Kafka solution is already valuable knowing that state-of-the-art showed that it could be interesting. Likewise, due to its popularity, Kafka can either already be integrated in a customer's information system, or can be the bridge to integrate ZORRO with existing customer systems. On the other hand, the integration of Kafka should not be thought of as a final solution due to its poor compatibility with IoT device communication. Confluent [73], the company that open-sourced Kafka and provides a cloud maintained commercial product, has suggested multiple ways in which Kafka can integrate with an IoT layer [74]. Essentially, in order to maximise the potential of Kafka, an entity to as-

semble IoT layer data is desired. This entity can be deployed as a type of IoT gateway. These gateways form part of the edge computing layer and can efficiently aggregate IoT data and send IoT data to Kafka [34]. Another solution described in [74], could include a RabbitMQ-Kafka combined interaction as suggested in [70], or an MQTT broker Kafka interaction as done with HiveMQ [75]. This solution essentially consists in installing a broker closer to the IoT devices and capable of managing millions of connections and with unreliable network connections, and a Kafka broker managing fewer connections but capable of handling massive amounts of data. When considering this solution a question that one may ask is why not just deploy a single MQTT broker? Essentially, even though MQTT brokers are optimised for managing communication of IoT devices, few of these brokers scale efficiently and the MQTT protocol offers poor connectivity with other enterprise applications (unlike Apache Kafka). Similarly, few MQTT brokers can achieve the data throughput that Kafka can.

3.8.1 Kafka Cluster Deployment

To enrich the Kafka deployment and to better satisfy the scalability, microservice and fault tolerance requirements, the message broker was deployed in containerised form. This was an upgrade with regards to the previous deployment of RabbitMQ which was directly installed on the virtual machines. A docker-compose file was used to define the Kafka deployment consisting of 3 Kafka brokers, 1 instance of Zookeeper to manage the distributed system that is a Kafka deployment and 1 instance of the confluent control centre to monitor the state of the Kafka deployment and provide a GUI to the cluster. The deployment architecture is illustrated in Figure 3.7. The Kafka ecosystem was deployed on a single OpenStack virtual machine (Type 1, table 3.2). Kafka was configured using mostly the default settings which favour latency over throughout and guaranteeing exactly once delivery. This however can be easily adapted and tuned for use cases where throughput is more important or a strict delivery policy is not required, thus satisfying requirements 2.a, 2.b and 2.c. Having the Kafka cluster setup, the factory simulator was connected to it through the Kafka producer API. As defined in the objectives, the communication protocol was also changed to JSON given its wide usage across many communications and IoT systems and better integration with Apache Spark for example. This therefore allowed us to better satisfy the requirement of loosely coupled sub-systems (1.c). Kafka also satisfies the communication security requirement (requirement 1.h) as it allows for Transport Layer Security.

Having deployed a Kafka cluster as a data pipeline, the data model used needed to be defined. This is always an important choice in Kafka as data distribution, message ordering and general performance depends on it. In the ZORRO project there are four main sources of information (measurement groups): on equipment, equipment surrounding, on product and product surrounding. In Kafka, the scalable entity is the topic, not the partition. As seen in section 2.3.2, having a topic divided into multiple partitions is what allows Kafka to be a very scalable system due to the possibility of parallel data consumption and parallel data production. It is for this reason that it is preferred in Kafka to have few topics with many partitions rather than many topics with few partitions. This justified the initial choice of having one topic per measurement group (on equipment, equipment surrounding, on product and product surrounding),

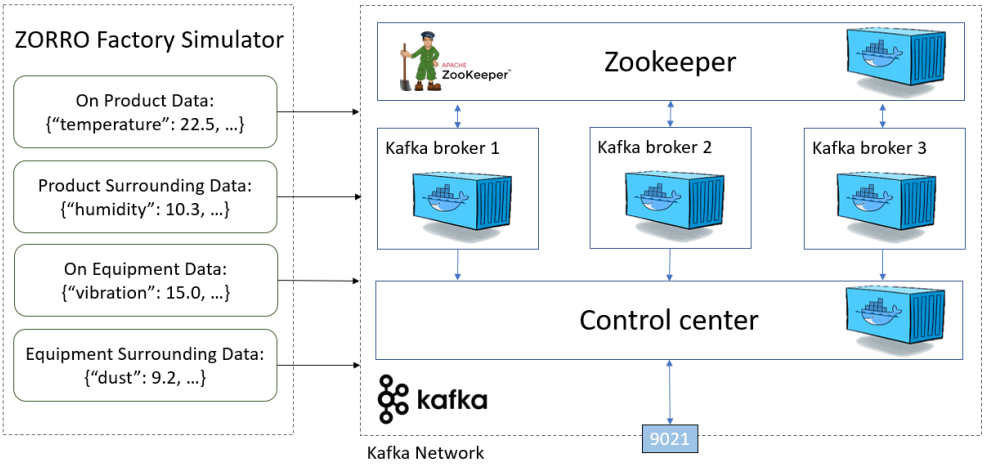


Figure 3.7: Kafka cluster containerised deployment.

rather than one topic per equipment or per stage. Once the topics are chosen, given that messages in Kafka are key-value pairs, the second important entity to define is the key of the Kafka messages. The message key will define the partition in which data will be stored. Kafka groups messages with identical keys by storing them in the same partitions. Given that message ordering is only guaranteed at a partition level, for equipment readings, the equipment id (unique identifier of the equipment that is being simulated) was used as the partition key. Similarly, for product readings, the product id was used as the partition key. This made sure that all measurements from the same equipment or from the same product were ordered as they were produced to the same partitions. Considering the on equipment topic only, divided into three partitions and with a replication factor of two, a single node Kafka cluster (containing 3 brokers) could store the partitions of the topics as illustrated in Figure 3.8. If we had three equipments in our production line, for example, Kafka would store the information from each equipment on a separate partition.

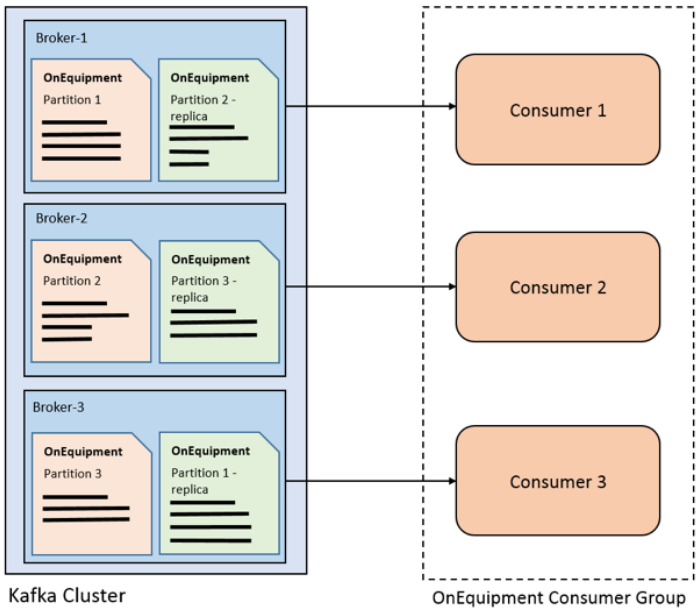


Figure 3.8: Kafka on equipment topic distribution and parallel consumption model.

As seen in the previous figure, a multi partition topic allows for data consumption to be done by multiple consumers (in parallel). With this setup, we are satisfying the Big Data latency, fault tolerance and throughput system requirements defined (1.a, 1.b, and 2.a). If the Kafka cluster is reaching its limit in terms of data throughput, it can be easily horizontally scaled by increasing the number of brokers, partitions and consumers.

3.8.2 Kafka Streams API

Following the deployment of Kafka, the Kafka streams API was used to process data arriving in Kafka in real-time. Kafka Streams is a Kafka Java library API that allows developers to write streaming data processing applications, consuming data from Kafka, doing some processing on it, and writing data back to Kafka. This could be done in Spark, implying however that data be transferred to a Spark cluster, which would add overhead to the processing. The Kafka streams API was yet another advantage with regards to RabbitMQ as it allows for intermediate data processing without the intervention of an exterior data processing engine such as Spark. To test Kafka streams, two streaming apps were coded. One application created a simple demonstration of an emerging technology in Industry 4.0 called virtual sensor [76]. A virtual sensor differs from a traditional sensor as the data it produces is not measured by a physical sensor on the IoT layer. Instead, the value of the virtual sensor is calculated as a function of one or multiple readings from the physical layer (sensors). It therefore involves some processing of the data from the physical layer before it can be visualised or delivered to the end-user. To demonstrate that our Kafka platform was capable of implementing virtual sensors, for each simulated machine, a reading of equipment electricity cost from the electric consumption of the machine was calculated. The other Kafka streams application that was developed performed a stream-to-stream join of data from two topics where the data key was the same, Figure 3.9. To test the second application, a stream-to-stream join between the on equipment and equipment surrounding topics was done. The resulting information was stored on an Equipment topic. The JSON contents of the original messages from each topic were simply concatenated. This can be useful, as will be seen in a future section, for machine learning models that were trained using data from both on equipment and equipment surrounding data sources. If this was the case, then during predictions these models need these data sources joined in order to make the prediction.

These Kafka streams apps were coded in Java, the language in which Kafka is written, and containerised using docker. Having been built on top of the Kafka consumer and producer API's, the Kafka streams apps are horizontally scalable. In order to increase processing capacity, all that is needed is to increase the number of application instances.

Kafka streams was an important addition to the ZORRO platform as it allows for the execution of quick, fault-tolerant and scalable Extract, Transform and Load (ETL) jobs to data before sending it to any of the end-point systems. As seen above, these are typical transformations needed for virtual sensors for example where a reading must be calculated as a function of multiple others.

Apart from satisfying the Big Data constraints and requirements, Kafka streams also brings the

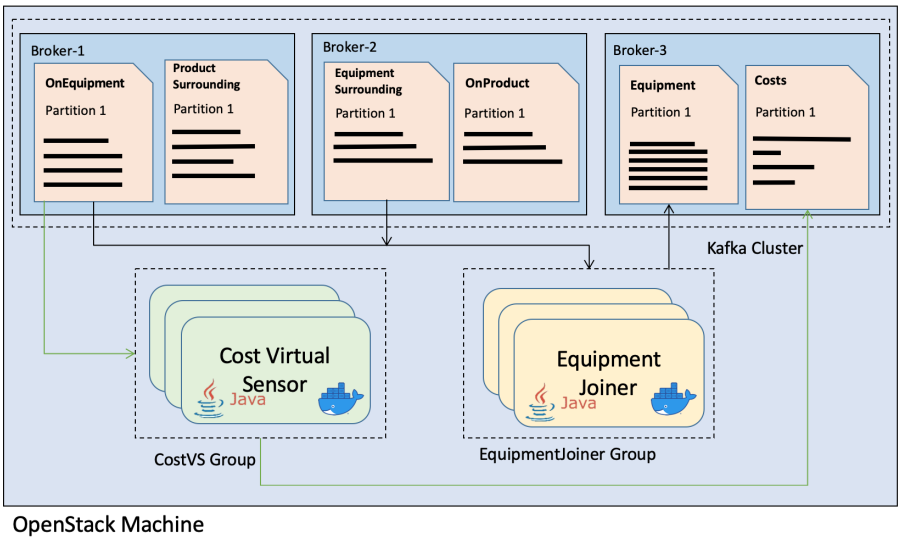


Figure 3.9: Kafka streams ZORRO applications.

ZORRO architecture one step closer to a micro-service based architecture on the processing front, as each streams application is dedicated to performing a particular task. With these Kafka streams applications, we have showed that we are capable of performing general data processing tasks and providing virtual sensors, used in Industry 4.0 [77], without the need of a dedicated data processing cluster (Spark). This can also be useful in scenarios where pre-industry 4.0 machines are deployed on the production floor and more information on these machines is required other than that provided by its embedded sensors. Likewise, scenarios where online data processing is required but without complex machine learning and data science heavy tasks where Spark would be used, the Kafka streams API can be an efficient lighter solution.

3.8.3 Kafka Cassandra Connectors

As will be seen in the next section, the Cassandra database system was the technology used for persistent data storage. In order to populate Cassandra with the data from the factory floor (simulator), another docker-compose file was defined to create a cohort of Kafka-Cassandra connectors. These connectors were developed in Python and use the Kafka consumer API and the Cassandra connection API. The script polls a specific Kafka topic and writes to a specific Cassandra table. For each topic, different but similar scripts were written and were containerised. Figure 3.10 shows the state of the full implemented Kafka ecosystem plus its connection with the Cassandra cluster. As explained in section 2.3.2 and in the previous section as well with the Kafka streams API, given that these connectors were coded using the Kafka consumer API, they can be scaled by adding more instances which would share the data consumption load. It was desirable to have one connector per Cassandra data table allowing for them to be scaled independently of each other. For example, if the number of machines from a factory remain the same, but more products are being produced and monitored, we would want to scale the Cassandra product connectors (to transfer more product data from Kafka to Cassandra), but keep the Equipment connectors as they are.

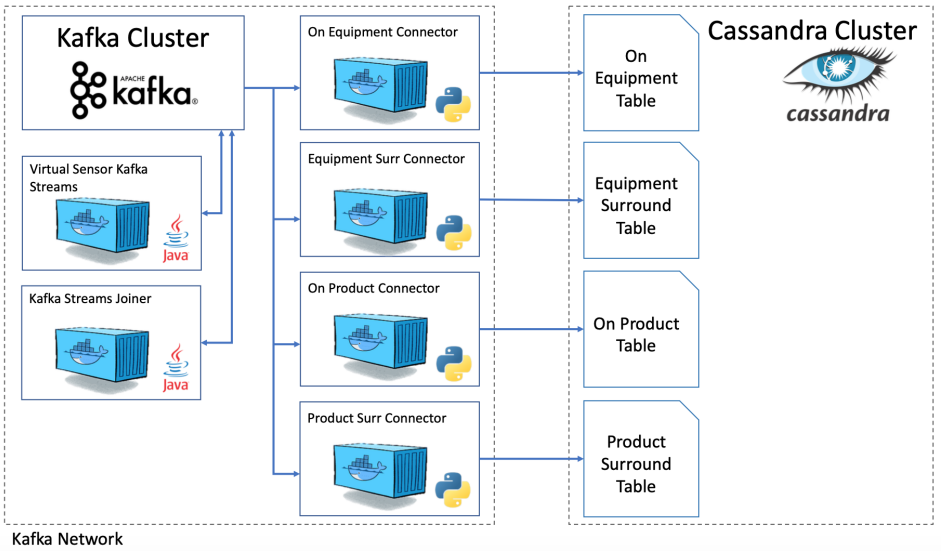


Figure 3.10: Kafka deployment with Cassandra Connectors.

3.9 Time Series Database and Data Visualisation

Inspired by the state-of-the-art study carried out in the previous internship, the choice of the TICK stack as the time series data management, metrics collection and visualisation system was maintained. A docker-compose configuration file was used to describe the desired containerised TICK deployment state. The final deployment consisted in all elements of the stack, namely, one Influx database container (connected to the virtual machine's persistent file system), one Kapacitor instance and one Telegraf instance. Chronograph, considered a serving layer technology, was deployed on a separate machine. In both cases, the technologies were deployed on type 1 machines (table 3.2).

Apart from creating dynamic and interactive dashboards of the arriving data, Chronograf also allows for the general interaction with the time series stack, from interactive queries on data, to the generation of automatic alerting systems based on the values received from data in real-time (Chronograf is used to define the criteria of the alert and Kapacitor is used to process the data). Telegraf was configured to consume data from the various ZORRO topics in Kafka, commit this data into the database (InfluxDB) so that it can be visualised and processed. Telegraf was chosen as a replacement of Apache Nifi, used in the previous deployment, as it is designed and optimised to send data specifically into the TICK stack. Previously, data was communicated using the InfluxDB line protocol. This guaranteed a good compatibility with InfluxDB but not with other systems such as Spark or Cassandra. As described in the Kafka section, to tackle this, the adopted data format was JSON given its popularity and wide adoption. Telegraf was configured to be able to ingest data from Kafka in JSON format, and convert it to InfluxDB's line protocol (defining which JSON fields were tag-sets and which fields were field-sets). Figure 3.11 shows the final, two machine, TICK stack deployment which consumes data from Kafka, stores the data in InfluxDB and visualises the incoming records in real-time through Chronograf. The overall TICK stack deployment improved with regards to the previous architecture where each element was deployed individually.

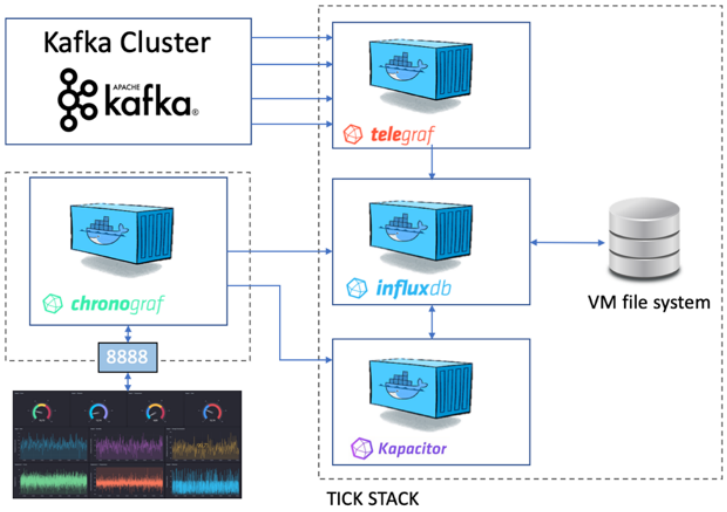


Figure 3.11: TICK Stack – ZORRO deployment.

Once again, it is important to verify and mention that the proposed system satisfies all Big Data constrains. Telegraf and influxDB are known for being able to ingest a significant amount of data when compared to other time series data systems [39]. Even if we reach the limits of these systems, the consumption from Kafka can be increased by adding Telegraf instances and multiplying the number of container instances of InfluxDB. The disadvantage of this would also be that the scalability requirements depend on our architectural and data distribution decisions. To solve this, the InfluxDB cloud offering provides cloud clustered deployment of the TICK stack capable of ingesting millions of metrics per second. This solution comes however at a cost, as the cloud offering is a commercial offering.

3.10 Data Lake Persistent Storage

Big Data systems require databases that are fault tolerant, scalable and distributed in nature. Once again inspired by the state-of-the-art study performed in this work and by the work performed on previous ZORRO internships, it was decided to maintain the choice of Apache Cassandra as the persistent storage system. As seen in section 2.3.5, Cassandra was designed with Big Data constraints and Big Data performance in mind and is therefore perfect for our use case as it satisfies the scalability, fault tolerance and Big Data constraints defined in the system requirements section (1.a, 1.b, 1.e, 1.g and 3.c).

In the context of ZORRO, Cassandra contributes to satisfying two main objectives: creating a historic record of manufacturing data for accountability and future insight extraction and ability to detect production anomalies in real-time using machine learning. Concerning anomaly detection, one way to do this is to use binary classification models. In order to train these machine learning models to do a wide variety classifications and predictions, data is needed. For these models to perform well and capture many scenarios, they need to be trained with large amounts of data which cannot be stored in a local machine or in classical relational databases. Once a suitable storage system like Cassandra is obtained, it must then be connected to an efficient data processing system, to extract insight from this

data and train machine learning models with it.

Cassandra is therefore used as the Data Lake and persistent storage system, where all data will be stored for long term retention. Apart from its performance when dealing with Big Data use cases, Cassandra is also relatively compatible with the Spark data processing engine. To ingest the data, Cassandra was connected to Kafka through python programmed connectors already discussed. To connect to Spark, for machine learning model training, an open source connector was used to connect the different Cassandra nodes to those of the Spark cluster. This connector allows us to expose Cassandra tables as Spark DataFrames and vice-versa as shown in Figure 3.12. There is no equivalent connector for InfluxDB, making the use of Cassandra with Spark even more desirable. With regards to deployment, a simple containerised Cassandra cluster was setup in one machine of Type 1, Figure 3.12. In order to satisfy the interface to the database system requirement (3.a), an Apache Zeppelin notebook application [78] was used and deployed as shown in Figure 3.12. Notebooks are used to write in code snippets (blocks) whose output is displayed on the notebook page. It can also be used to write text and other HTML objects. Notebooks are thus very popular within the academic community as they are a great way to visually and in an interactive manner share code and insights. Zeppelin provides an easy connection integration with a Cassandra cluster and an interface for CQL (Cassandra Querying Language) queries to be run on it.

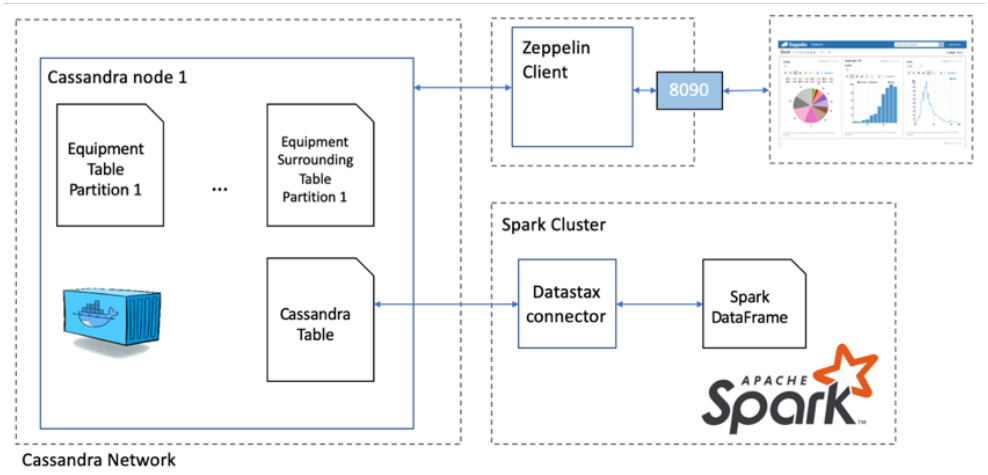


Figure 3.12: Cassandra cluster setup and Zeppelin and Spark connections.

Queries submitted on the Zeppelin notebooks are run on the raw data stored in Cassandra and the output is given on a visual (graphic or tabular) interface.

The data model chosen for Cassandra was consistent with that of Kafka's. A table for each measurement family (Kafka topic) was created. Two other tables were also created, one for the data from the equipment (on equipment + equipment surrounding) and another for the data from products (on product + product surrounding). Given that the state of a machine or product depends not only on the internal state of the machine and product but also on their surrounding properties, in order to enrich the information provided to the machine learning models, they must be trained with information from a table containing both on equipment and equipment surrounding information. A table-to-table batch join was

done using Spark both for the equipment and product information. This is exactly the same join that was done with the Kafka streams API but this time using Spark and joining Cassandra tables instead of Kafka topics. In most tables, the partition key was defined to be the product id or equipment id plus the sensor id by which the measurement (row) was taken. For the clustering key we used the timestamp to guarantee that a measurement taken by one sensor on a given system at a given time is unique.

In Cassandra, queries are restricted to the data model used. In particular, certain operations can only be performed on the columns that were used to define the partition key. Nevertheless, group by, aggregations and other classical SQL like operations can be done through this Zeppelin notebook on the partition key columns. To perform other more complex queries and data processes, the a data processing engine, like Spark, must be used. The deployment of the chosen data processing engine will be described on the next section.

3.11 Data Processing and Machine Learning

With regards to the previous internship, Spark was maintained as the data processing solution due to its exceptional performance in terms of processing speed, Big Data scalability, machine learning libraries, Python integration and Kafka integration. Another clear reason for selecting Spark is the fact that it allows us to implement the lambda Big Data architecture using a single framework: Spark. In this section we will describe the contributions done to the ZORRO data processing layer.

3.11.1 Deployment

A Spark cluster was deployed and configured in two virtual OpenStack machines. One of Type 1 and another of Type 2. One machine contained one worker and the other machine contained one master and one worker (Figure 3.13). Spark was configured to have one executor per worker, with proxy-side authentication and ZORRO specific packages installed (Kafka and Cassandra connectors). The Spark cluster was connected to both Kafka in streaming mode and to Cassandra in batch mode as described in section 3.10. This marked the completion of the desired Big Data lambda architecture for data processing. The standalone cluster manager of Spark was chosen due to its ease in deployment.

Given that the objective of containerised deployments was not satisfied here, a future objective would be to deploy Spark using Kubernetes (k8s). This would be a clear upgrade because, as described before, k8s is an excellent container cluster manager with very good support for with Spark.

3.11.2 Spark Analytics

With a Spark cluster setup, an application or script can be run on the cluster through submitting it to the cluster manager. This generally involves transferring the script to one of the machines of the cluster and submitting it to the cluster.

Data scientists and machine learning engineers however generally tend to prefer interactive tools to develop and test their applications such as notebooks (as already discuss with the Zeppelin notebook).

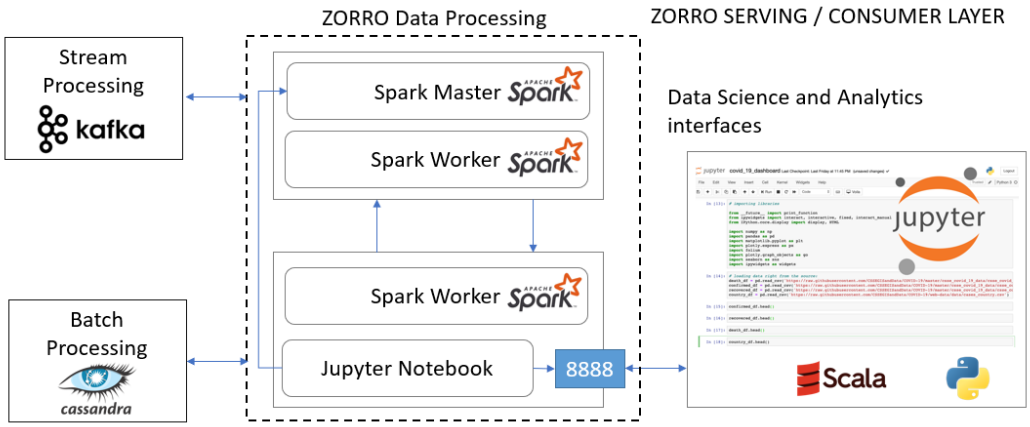


Figure 3.13: Spark cluster deployment and integration with Jupyter.

Due to the ability of notebooks to provide these interactive tools, Jupyter [79] was installed on one of the machines of the Spark cluster and configured to be able to communicate with the Spark cluster. Jupyter is a web application that allows developers to write interactive code in the form of notebooks. In the context of ZORRO, Jupyter allows for the development of Spark applications and use of machine learning libraries in a more interactive and dynamic way. The Jupyter notebook web client therefore serves as a programming interface to interact with the ZORRO data processing system. This can therefore be considered a serving layer technology as it allows data scientists and machine learning engineers, who might not be very familiar with distributed systems, to interact efficiently with the ZORRO ecosystem and be abstracted to the complexity of the system, Figure 3.13.

In terms of application deployment, if the user specifies the master url, the code written in the notebooks is uploaded to the Spark cluster and executed in the Spark distributed system. The output is then sent back to the notebook to be displayed. Developing Spark applications in Jupyter is also an excellent way to test these applications before deploying them in the cluster with a script directly.

Overall, Jupyter reduces the complexity and ease the access to the ZORRO ecosystem providing a collaborative tool for developers to share code templates or snippets that can be run easily. It is these sort of collaborative tools that allow data scientists and analysts to develop programs that allow them to make data driven decisions. These are the tools that an Industry 4.0 solution must contain as, after all, in Industry 4.0 the idea is to use data to perform data driven decisions driving optimisations, simulations, reactions and forecasting. Given that Spark is connected to Cassandra in batch mode, the Jupyter notebooks can be easily re-run with the current data that is in the Cassandra cluster therefore updating the Jupyter outputs.

3.11.3 Batch Processing

Following the deployment, we then proceeded to working on a batch processing pipeline. The main use of batch processing in ZORRO is to train machine learning algorithms on huge amounts of data requiring fast and parallel processing. In an industry 4.0 context, models can be trained for anomaly detection, predictive maintenance and supply chain resilience. As a starting point, and because it is the

main objective of project ZORRO, we have decided to train models for anomaly detection.

The first difficulty was using multiple sources of data and compiling relevant information into one single table so that it can be fed to the models for training. In the context of this project, if we wish to train a model to predict anomalies on equipments, then both on equipment and equipment surrounding information is required. This is because the anomaly can be a function of features from one or both entities. For example, if a machine is making too much noise, this is something that is measured on the surrounding environment of the equipment and not on the equipment itself. To combine the information from both these sources, a criterion based on the timestamp of the data was defined. The way in which the joins were done in batch mode are illustrated in Figure 3.14. Each point represents a measurement on each one of the systems (on equipment and equipment surrounding for example). The join was done following a simple equipment id and timestamp-based criteria.

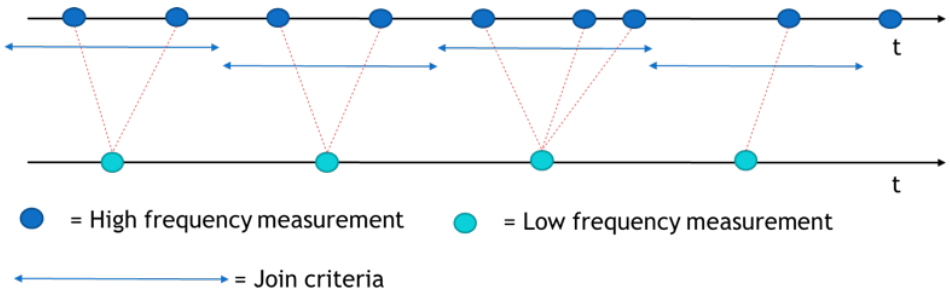


Figure 3.14: Batch join of two tables of timestamped rows.

Essentially, we wish to join two measurements that refer to the same system (same equipment) and whose values were recorded at similar times. An inner join is therefore done on the equipment id and under the condition that the timestamp columns are close to each other (we used the period of the low frequency reading to define the criterion). Normally readings are taken at constant frequencies and so the timestamps should be evenly spaced out. However, even if a reading was taken too early or too late, then it will still just join with the reading it was closest to. The result was a table containing the same number of rows as the table containing the measurements with higher frequency, but with all sensor records compiled into each row. These joins were coded using Jupyter notebooks in Pyspark (Sparks Python API). All the code that was produced in Jupyter notebooks, and submitted to the Spark cluster, can be easily converted into scripts to be deployed in the Spark cluster and be constantly running in a production environment. This could be a periodically triggered batch job deployed on the ZORRO system so that a periodically updated equipment table is available.

With the joined data and therefore a single table in Cassandra containing all relevant equipment features, before a supervised machine learning model can be trained using Spark on the data from this table, the data needs to be labelled. With regards to anomaly detection, a label column indicating if the measurement taken corresponded to a time where the machine was working well or not (non-faulty or faulty) is needed. This process is known as data annotation. Following the labelling of the data, the models were then trained. Once the model was trained, tested and tuned, it was exported so that it could be used in real-time in streaming mode. A summary of the batch process workflow is given in

Figure 3.15.

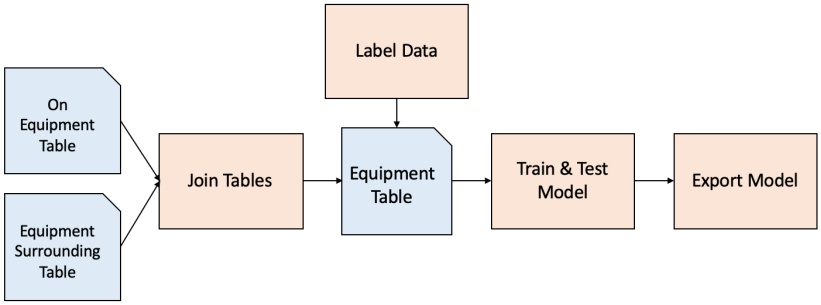


Figure 3.15: Batch machine learning job workflow.

3.11.4 Stream Processing

The second branch of the lambda architecture is the streaming layer. This is a fundamentally different layer to the batch layer and has a series of other difficulties and issues associated to it. In this work, and in the context of the ZORRO project, part of the streaming needs such as joins and virtual sensors, were already accomplished using the Kafka streams API. However, automatic anomaly detection using imported machine learning models must be done using Spark due to the available libraries and distributed processing efficiency. To connect Kafka to Spark in streaming mode, Spark offers an easy to use integration to stream data from Kafka. This is available in Spark's most recent streaming module, Structured Streaming [80]. Before the streamed data can be used for the predictions however, data from the Kafka topics must be joined (on equipment and equipment surrounding). This is because, as was described in the previous section, the models were trained using joined data. For the model to be able to make a prediction, it must be given all the equipment features that is was provided during training. As illustrated in Figure 3.16, the Spark streaming application must therefore create two independent streams, one for the on equipment Kafka topic and another for the equipment surrounding Kafka topic, join them and then apply the prediction using a model loaded at the start of the streaming application and which resulted from the model training stage.

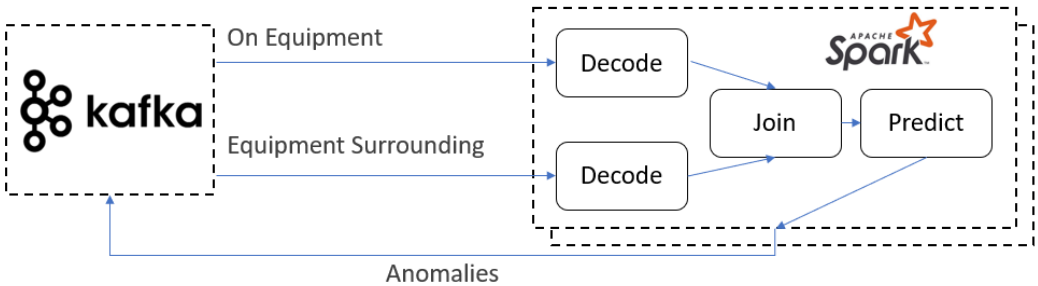


Figure 3.16: Kafka Spark streaming connection.

Given that data from Kafka is encoded in binary form, it must first be decoded to obtain the JSON string. Included in the decoding stage, along with decoding the binary data, the JSON type data obtained must then be converted into a Spark DataFrame. The method in which data is joined was not the same

as the one used and described in Figure 3.14. The stream-to-stream joins were done using a tool in Spark known as watermarking. The issue with joining streaming data is that a reading received now can possibly join with a reading received in the future. Given that the joins performed in our case depend on a timestamp, a reading received now can match with another reading that is late in its arrival. In order to respond to these situations, a state must be maintained throughout the streaming process. This triggers a question which is how long should this state be? If no limit is put, then the state will grow indefinitely. This is where Spark's watermarking tool can be used to define this length. The watermark defines a time window until which data is dropped. If a data point arrives and it is outside the time window, this data point is dropped and not used for joining. Similarly, if a data point arrives on time, it is kept for the duration of the watermark.

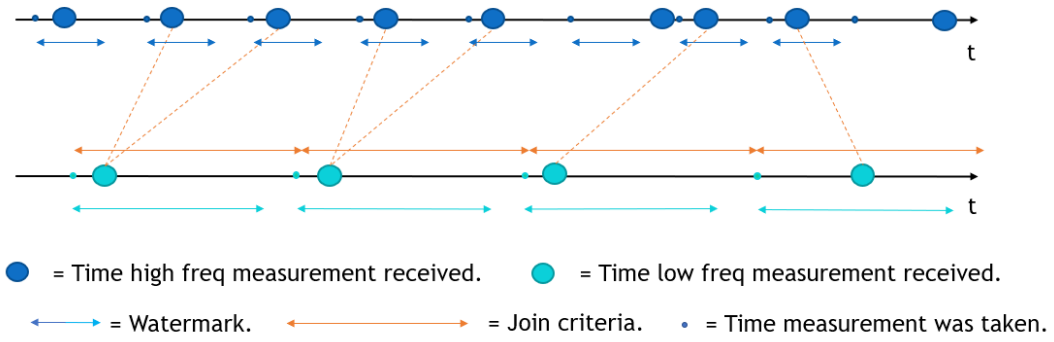


Figure 3.17: Stream to stream join in Spark using watermark.

The watermark value should be chosen based on the admissible/expected latency bearing in mind of course that the longer the watermark, the larger the streaming state and therefore the larger the impact on performance the state will have. Figure 3.17 shows a possible scenario of a Spark stream-to-stream join using watermarks. The join criterion once again is based on equipment id and on a timestamp window. For stream-to-stream joins, we have decided to join on equipment measurements with the latest value of equipment surrounding (defining the window with the period of the measurement of lower frequency). If there is no major latency, the joins work fine as illustrated in the start of Figure 3.17. If latency arises (which in the figure is the case of latency in the high frequency measurements), then some joins are never done because the measurements are no longer within the watermark. To solve this, we would need to increase the watermark. In the case of the figure, increasing a little bit the watermark seems to be the rational thing to do. However, increasing too much the watermark will lead to increased retained state and therefore increased latency associated to the joins.

Once the predictions are done, if Spark identifies any anomaly, it writes back to the Anomalies Kafka topic. By writing the anomalies into a Kafka topic rather than directly into a database, other endpoint systems can react or learn about the fact that an anomaly was identified by subscribing to the Anomalies Kafka topic. This is particularly interesting in an Industry 4.0 solution where some automatic responses could be triggered based on the anomalies identified. The proposed solution for streaming predictions is scalable as both technologies can be horizontally scaled and are known to be so (Kafka and Spark [32]).

Chapter 4

Developed System Outputs and Performance Analysis

The results of the developed work in the context of the internship this dissertation was carried out in can be divided into two main contributions. Firstly, the creation of know-how for AKKA Technologies regarding Big Data Industry 4.0 systems. Secondly, the development of a Big Data Industry 4.0 meta solution with quantitative and qualitative proof of concept outputs and performance indicators. In this chapter we will focus on the results obtained from this develop solution both in terms of the outputs it can produce and its performance.

4.1 Real-time Data Visualisation

In this section the results obtained from the real-time data visualisation and asset monitoring tool will be presented.

4.1.1 Factory Dashboards and Monitoring

The main objective of the data-visualisation system, is to be able to monitor the status of a factory-floor. Figure 4.1 is an example of one of the dashboards that was produced displaying measurements from an equipment that was simulated. As can be seen in this Figure, Chronograf offers different visualisation elements which can be configured by the user and are updated in real-time. Due to InfluxDB's querying capabilities, multiple features of different sensors can be monitored. We can, for example, choose to display sensor instantaneous values, as is illustrated in the bottom two graphs of Figure 4.1, or we can choose to follow the moving average of a sensor (top left measurement gauges).

Dashboards like these mark the completion of the real-time data visualisation and asset monitoring objectives which can clearly be applied in use cases such as remote analysis (maintenance) and visual insight extraction. In order to not be locked into the Chronograf tool and to demonstrate that having InfluxDB as our choice for time series database is not restricting the data visualisation offers, other

Figure 4.1: Example of a ZORRO Chronograf dashboard produced.

dashboard tools such as the very popular Grafana tool [81] were also used and produced very similar results.

4.1.2 Deployment Metric Collection

In addition to real-time data visualisation from the factory floor, the dashboard tool was also used for system and deployment monitoring. As described in section 2.3.4, Telegraf is the element of the TICK stack that is responsible for the ingestion of data into InfluxDB (database of the TICK stack). Apart from having support for Kafka data ingestion, and therefore, in our case, being able to obtain data from the factory floor, Telegraf is also capable of ingesting metrics from other systems, such as docker and hardware system metrics. These other metric collection plugins were added to the ZORRO project in order to monitor the state of the deployments of the various sub-systems. For small-scale systems, this metric monitoring might not be very useful, however, for Big Data scale solutions, like ZORRO, state monitoring of the deployment is essential. Figure 4.2 is an example of such ZORRO metric collections where two dashboards indicating the docker deployment CPU and Network loads are shown. This was yet another advantage from the previous Apache Nifi deployment which was not configured for this deployment metric collection.

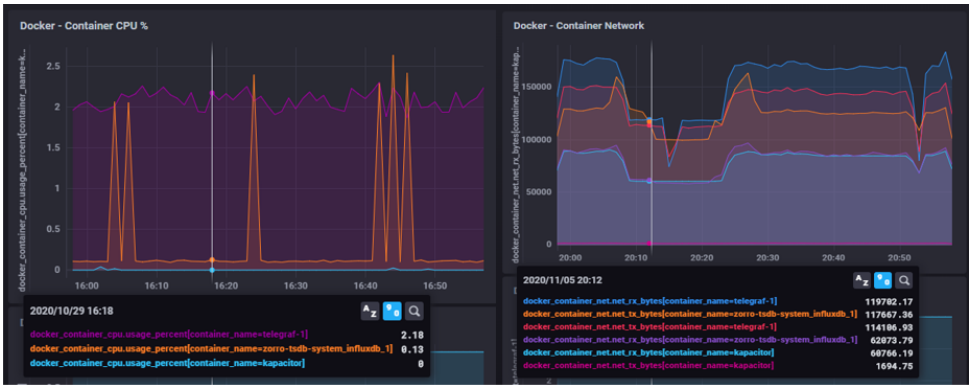
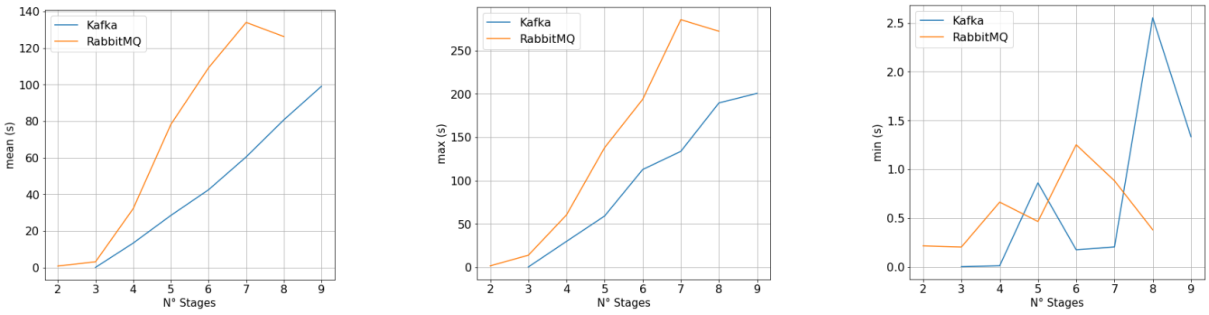


Figure 4.2: Kafka docker deployment monitoring with the TICK stack (CPU).

4.1.3 Latency Performance

A latency analysis was performed on this manufacturing asset monitoring service to understand the extent to which the visualisation was real-time. The latency here measured refers to the time taken for the data to be written into influxDB in comparison to when it was produced by the factory simulator. In our case, this corresponds to the latency obtained from the factory, in other words, the moment the timestamp was inserted, to Kafka to Telegraf and finally to InfluxDB. The latency tests carried out consisted in progressively increasing the data load on the system. The load was increased by increasing the number of production stages being simulated. Each stage contained two equipments. For each test, an extra factory stage containing two equipments was added. Each equipment contained 4 data producers (on equipment, equipment surrounding on product and product surrounding), each producing 5 messages per second. Each group of 5 messages, encoded in utf-8 holds, around 6 KB of data. Therefore, this corresponds to around $6 \times 4 \times 2 \times nb_{stages}$ KB/s of data. For 9 stages for example, the amount of data being transferred is 432 KB/s or 360 messages/s. The simulated messages are quite large for some IoT scenarios. Nevertheless, their sizes are consistent with complex message types with larger aggregation of values and meta information [25]. Initially, the simulator was run on a single machine of Type 1 (table 3.2). A comparison between the previous system (using RabbitMQ) and the current one was made. The results comparing the mean, maximum and minimum latencies obtained are illustrated in Figure 4.3.



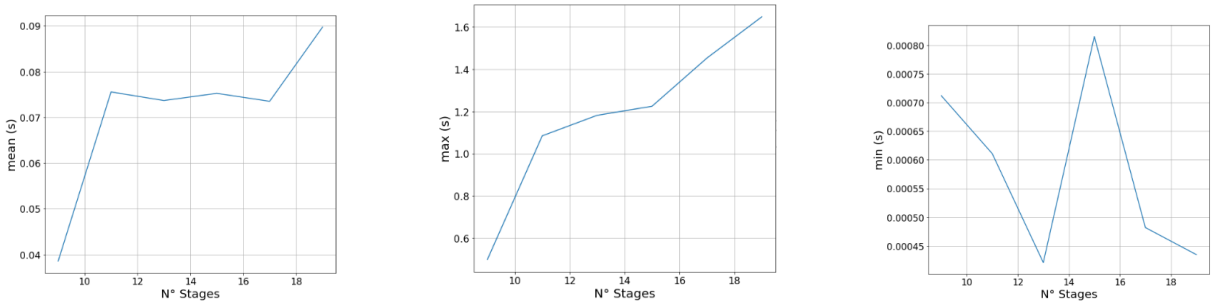
(a) Mean latency. (b) Maximum latency. (c) Minimum latency.

Figure 4.3: Kafka vs RabbitMQ latency analysis.

By analysing the mean and maximum latency values obtained in Figure 4.3.a and 4.3.b respectively, it is clear that the latency obtained was much larger than normal and not consistent with the state-of-the-art-findings [70], [39]. It was therefore attempted to find the true source of latency. It was discovered that the latency mainly came from the factory simulator as it was running slowly when the data load was increased. This motivated an optimisation of the factory simulator. This optimisation involved code changes and timestamp insertion changes. Although the results may not be representative of the nominal performance of both technologies, they allow of the comparison between Kafka and RabbitMQ as they were simulated under the same conditions. Under the simulated conditions, the Kafka message broker obtains significantly smaller mean and maximum latencies in comparison with RabbitMQ.

Despite the optimisation of the simulator, the dynamics of the latency were still significantly impacted by the load on the simulator. The simulator was thus parallelised, reducing the load on the machines

hosting them. This is what can be seen in Figure 3.6, where the factory simulator is being hosted in two different machines (both of Type 1). All these optimisations gave way to a second latency campaign measuring the mean, maximum and minimum latencies of the Kafka broker. The results obtained are available in Figure 4.4.



(a) Mean latency. (b) Maximum latency. (c) Minimum latency.

Figure 4.4: Kafka latencies obtained after simulator optimisations.

Overall, the values obtained for the latency are much more admissible with the mean, Figure 4.4.a, being always inferior to 0.1 s. Likewise, the minimum values of latency (Figure 4.4.c) confirm Kafka's capacity to have very small latencies. The results obtained are interesting as they demonstrate that the developed Kafka platform, which is architecturally stable but was only simulated with a single small machine (machine Type 1), respects latency levels required for condition monitoring and is close to the latencies required for augmented reality [20]. With regards to motion control, these latencies are too large and therefore, as expected, an edge computing solution is more suitable for machine and equipment automatic control. Despite the admissible latencies obtained after the optimisation and parallelisation of the simulator, it was discovered that the dynamics of the latencies were still impacted by the simulator. Indeed, large changes in the latency were seen precisely when there was a significant change in the load on a machine. Given that the simulators were coded using the multi-threading and queue libraries of Python, this latency was probably due to RAM memory depletion of the machines hosting the factory simulator.

In conclusion, a more efficient and parallelised simulator is required to truly measure the latency and throughput performance of the current ZORRO system. Notice however, in Figure 4.4, that up to 19 stages were tested, which correspond to around 912 KB/s and 760 messages per second. Despite this load, the latency obtained was still only of 90 ms. We stopped here because the simulator once again demonstrated its limit. In a real IoT system, the latency could be higher due to weak network connections and physical distance. Nevertheless, we can conclude that once this data is on the TCP layer, communication is very fast. Overall, we can confidently say that we have the basis for what appears to be a system capable of handling large amounts of data in real-time rather quickly.

All the graphs and analysis above were produced using Jupyter Notebooks. These allowed for visual and interactive graphing and analysis of the results obtained. Likewise, these notebooks can relatively easily be picked and understood up by someone new to the project as along with the code, HTML text snippets were added to comment on the results.

4.2 Data Science and Exploration Tool

In this section, the results obtained from the data science, analytics and data exploration tools deployed in will be presented.

4.2.1 Zeppelin Notebook

As described in section 3.10, a Zeppelin notebook was deployed as a tool to interact with the raw data in the Cassandra database and extract insights and perform analytics. Figure 4.5 shows an example of some outputs produced. As a notebook, one advantage of using a Zeppelin to query the data is that the queries that are run can be saved and re-run at any time with the current data that is present on the database.

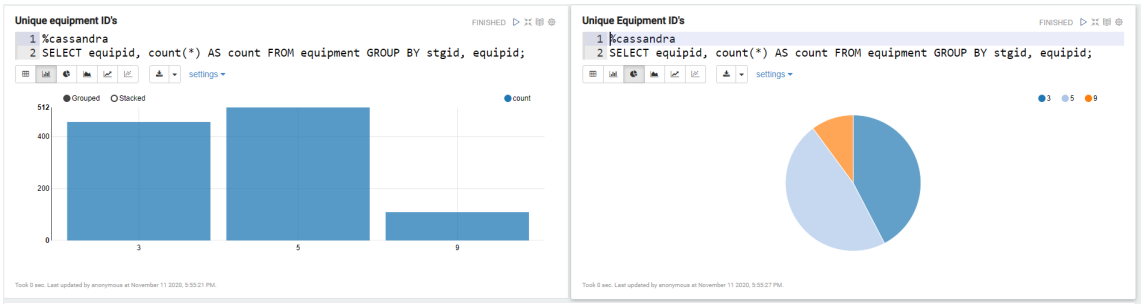


Figure 4.5: Zeppelin notebook for querying Cassandra DB and dashboarding output.

4.2.2 Jupyter Notebook

As seen in Figure 3.13, a Jupyter notebook was added to allow for analytics and data exploration. To test this tool, and also demonstrate its potential, a series of data exploration methods were used. Python libraries were used to analyse a fraction of the produced data in the Cassandra database. The interest of this tool lies here, as it allows for Data Sciensits to connect to a Big Data scale processing platform (ZORRO) using tools that they are familiar with. Some of the outputs from the data exploration obtained can be seen in Figures 4.6 and 4.7.

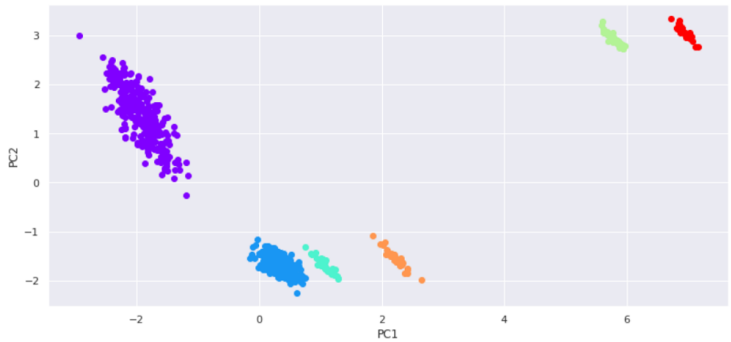
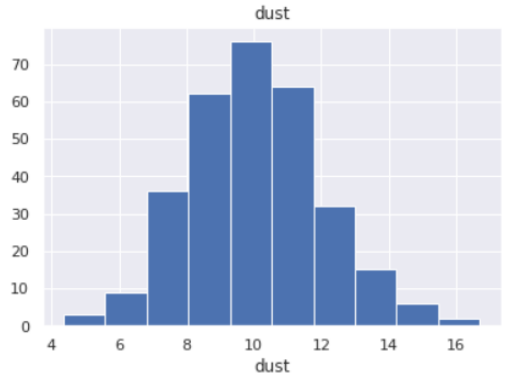


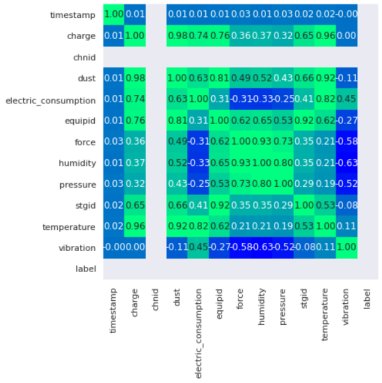
Figure 4.6: Results from data visualisation - Colour coded 2D PCA representation of 6 equipments.

Illustrated in Figure 4.6 are the produced output of a 2 dimensional Primary Component Analysis

(PCA) showing colour coded clusters corresponding to six different equipments (2 dimensional meaning using primary component one and primary component 2). PCA is a dimension reduction tool which allows for the visualisation of a high dimensional system, like an equipment containing temperature, humidity, vibration and other measurements, in a smaller dimension (2 dimensions in this case), whilst attempting to minimise the amount of data lost in this reduced representation. It achieves this by representing the data in the directions of largest variance (primary components). It is considered an unsupervised machine learning algorithm and can be used, for example, to visually identify anomalies.



(a) Histogram of the dust measurement on a equipment.



(b) Correlation matrix of features of an equipment.

Figure 4.7: Results from data visualisation - histogram and correlation matrix.

Figures 4.7.a and 4.7.b show respectively a histogram of the dust measurement on one of the equipments and correlation matrix between measured variables from one equipment. The results obtained from these diagrams allow for the understanding, in a data driven fashion, of what nominal and non-nominal factory behaviour physically looks like. These outputs can also be used to understand where optimisations can be carried out. For example, using the correlation matrix, if we find that temperature for a machine is highly correlated with electric consumption, we can attempt to move this machine to a cooler area or reduce the temperature of the section it is in. Essentially, our platform is capable of using and manipulating tools which allow for data driven insights to be extracted and data driven decisions to be made.

The code produced in these outputs was written in Python, on Jupyter notebooks, deployed on the ZORRO platform, and can be easily shared as a template code or snippet to be reproduced and re-run against the current data present on the persistent data storage system (Cassandra). This was an important addition to the ZORRO system as most data scientists and machine learning engineers need to perform a data exploration phase in order to generate descriptive statistic summaries to understand the characteristic of the data and identify certain correlations or behaviours.

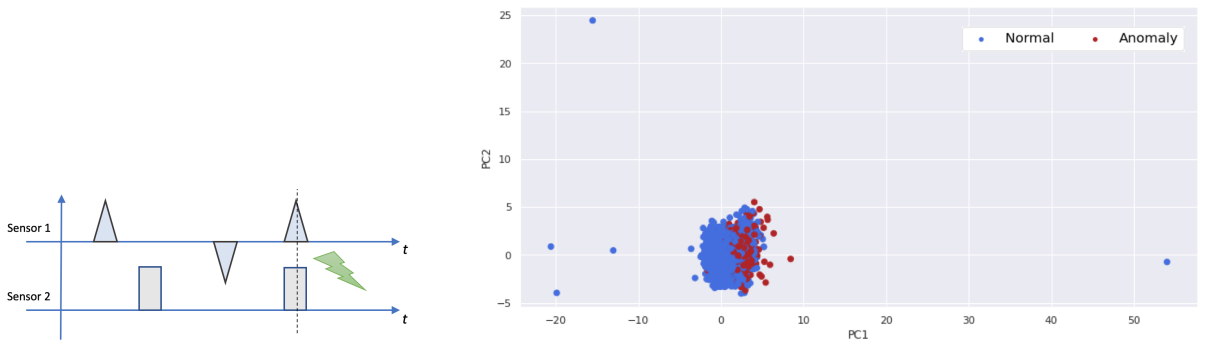
4.3 Automatic Anomaly Detection

In this section, the results obtained from the developed models for automatic anomaly detection are described. Both the training processes and the deployment of automatic anomaly detection machine

learning models are described.

4.3.1 Model Training

Once the data was joined, the next step was to train some models to detect equipment anomalies. As a starting point, four supervised learning models offered by Spark were used: Logistic Regression, Support Vector Machine (SVM), Decision Tree and Random Forest. The problem was therefore approached as a binary classification problem (faulty 1 vs non-faulty 0). In order to train a supervised learning model, data needs to be labelled beforehand. This means that the model must be given information regarding which data points correspond to anomalies and which correspond to correct behaviour of the system (equipment). A criterion must therefore be defined in order to label an anomaly. In this work, two anomaly criteria were chosen. In the first criterion, a reading was considered an anomaly if, at the same time (in one row), both values of two different sensors were above or below a certain threshold. The thresholds were defined depending on the mean, the standard deviation and the distribution of the sensor reading. For normally distributed sensor readings, the thresholds were defined at three sigmas (3-sigma) to the left and to the right of the mean (99.7% rule). For uniformly distributed sensor readings, the thresholds were simply defined by the extremities of the domain of the distribution. As seen in Figure 4.8.a, an anomaly (green lightening symbol) is defined by a combination of two anomaly sensor readings. Consider in the Figure 4.8.a that each triangle corresponds to an anomaly reading in sensor 1, say temperature, and each rectangle corresponds to an anomaly reading in sensor 2, say humidity. We only consider that there is an anomaly when, for instance, both temperature and humidity act strangely at the same time. The idea behind this criterion is that anomalies are related not to individual features but to a combination and relationship between features. In Figure 4.8.b, a 2D PCA plot is illustrated to get a rough idea of the distribution and proportion of anomalies during testing. In this case, approximately 5% of readings were labelled as anomalies (red points). The data plotted in Figure 4.8 corresponds to 20 minutes worth of simulation.



(a) Anomaly (green lightening) every time two sensors behave strangely (triangle for sensor 1 and rectangle for sensor 2) at the same time.

(b) PCA illustrating anomaly to non-anomaly data distribution for first criterion (first two primary components).

Figure 4.8: First anomaly criterion - sensor combination.

The classical machine learning process was then followed. The data was divided into training and testing data sets (75% to 25% respectively). The models were trained with the same training data set

and tested on the same testing data set. The results obtained for the model scores are available in Figure 4.9. A score of 1 is a very strong prediction score, whilst a score of zero is a very weak prediction score. These model results correspond to the prediction accuracy, given three criteria, on one equipment.

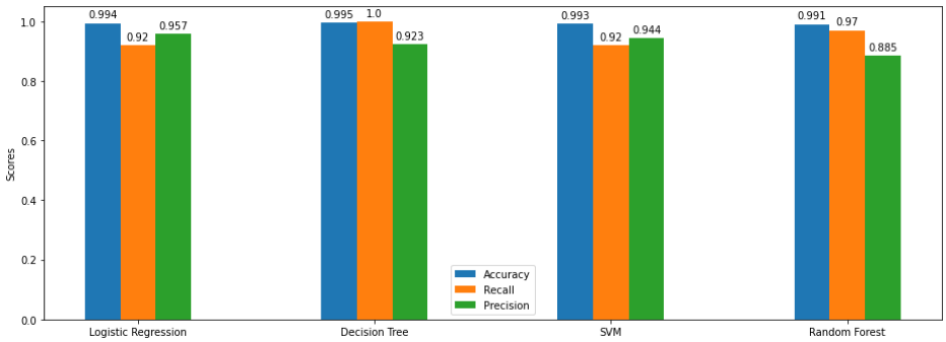


Figure 4.9: First anomaly criterion - model scores.

Initially, three metrics were used to determine model performance: precision, recall and overall accuracy, Figure 4.10. Recall refers to the ability of finding anomalies when they exist (true positives) [82]. It is also known as sensitivity. Notice that in our case, a positive refers to an anomaly. A low value for recall means that the model is unable to detect an anomaly when it appears. Precision refers to the quality of the model in finding the positive class (anomaly). It is thus also known as the positive predictive value [82]. A small precision implies that the model very easily labels a point as an anomaly, even when it is not. Finally, overall accuracy simply refers to the proportion of correct predictions with regards to all predictions. Naturally, the overall accuracy is very much related to both the precision and the recall.

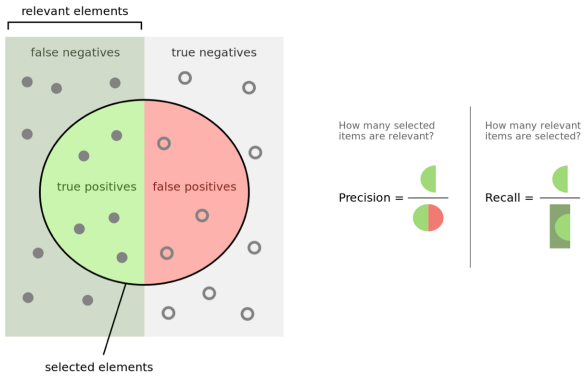


Figure 4.10: Precision and recall - binary classification.

In order to obtain a more precise evaluation of the models, precision vs recall and ROC curves were plotted. The models studied, apart from SVM, can predict a probability that a certain measurement is an anomaly, rather than the prediction of anomaly or not anomaly itself. This provides the machine learning engineer with flexibility, as the threshold of probability with which we consider a measurement an anomaly can be varied. The ROC curve plots the false positive rate against the true positive rate (recall) as a function of this threshold and the precision vs recall curve (PvR) does the same but naturally with the precision and recall values. For this anomaly criterion, the ROC curves obtained are illustrated in Figure 4.11. The area under the ROC curve (AUC) measures how well a binary classification model

separates the two classes. A good model maximises this area. The precision vs recall curves, are illustrated in Figure 4.12. This curve is similar to the ROC curve but takes more into consideration the imbalance between the two classes. This is particularly useful in data sets where there are many negative observations (class 0) and few positive observations (class 1), which is usually the case in anomaly detection scenarios. The key of the precision vs recall curve is that it does not use the true negative values.

In both cases, the ideal model maximises the area under each curve and is very different to the no skill model curve (blue). The curves confirm the satisfactory results obtained and demonstrate that for this criterion, none of the models are very sensitive to the value of the threshold.

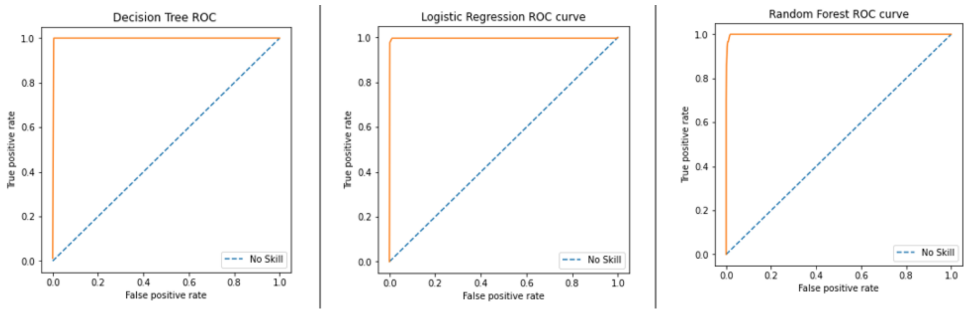


Figure 4.11: First anomaly criterion - model ROC curves.

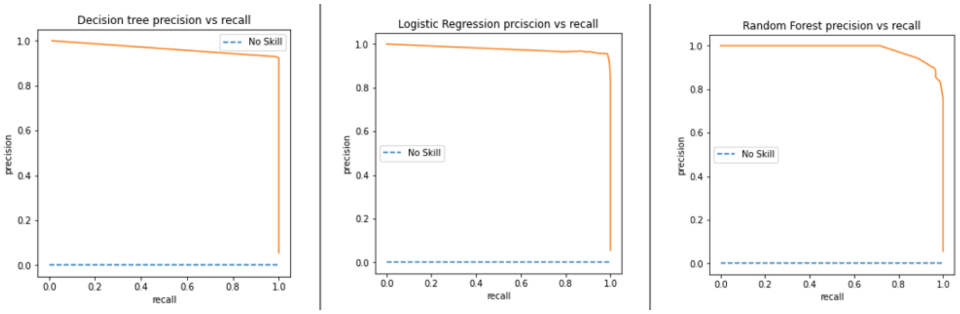
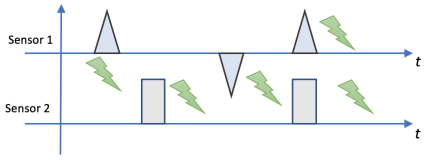


Figure 4.12: First anomaly criterion - Precision vs recall curves.

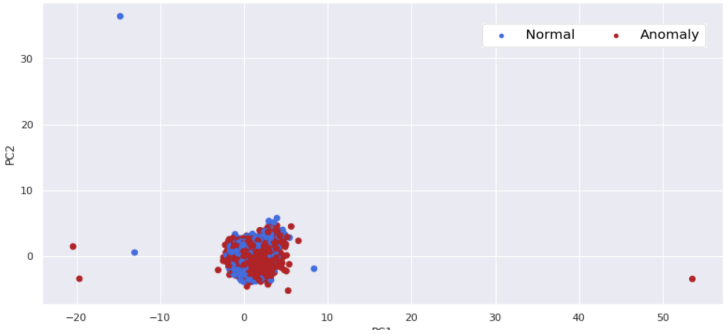
The results obtained show that tree-based models, especially Random Forest seem to very slightly underperform. It is worth mentioning however, that the parameters of none of the models were really optimised, therefore we cannot yet conclude which model is ideal for this type of situation. To optimise the models, the cross validation method could have been used especially because Spark offers support for this. The ROC and PvR curves obtained can provide the machine learning engineer with an idea of the ideal or adequate threshold for each model. Nevertheless, the choice for this parameter is generally more dependent on the use case of the model. In a critical production scenario for example, we are perhaps more willing to stop production if we believe that one of the machines is malfunctioning. In this case, it would be reasonable to define the threshold as smaller than 50%, to make sure we get all anomalies. In a mass production environment on the other hand, stopping production can be very expensive, and therefore we would only want to stop it if we were sure there was a serious anomaly in one of the machines. It is worth mentioning however that overall, for this scenario, the results obtained

for these models were very satisfactory.

In the second anomaly criterion, we identify an anomaly every time one of its features displays a value that is above or below the defined thresholds. This scenario is illustrated in Figure 4.13.



(a) Anomaly (green lightening) every time any sensor behaves strangely.



(b) PCA illustrating anomaly to non-anomaly data distribution for second criterion (first two primary components).

Figure 4.13: Second anomaly criterion – sensor anomaly.

In this case, the proportion of anomalies to non-anomalies has changed significantly. Around 33% of points are anomalies (red points). Figure 4.14 shows the model scores obtained following the same process and using the same model parameters.

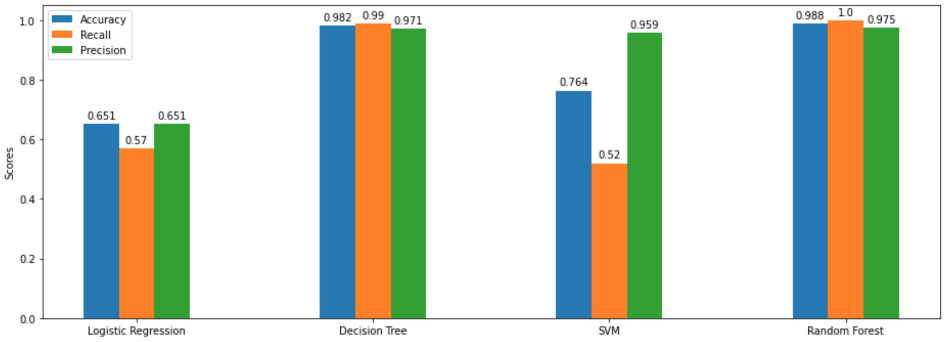


Figure 4.14: Second anomaly criterion - model scores.

In this second anomaly scenario, tree-based algorithms shine (Random Forest and Decision Tree). From this initial approach, we can already conclude that tree based algorithms seem to behave better in data sets where the ratio between classes approaches 50%. Similarly, when the number of features that contribute to an anomaly increases, SVM and logistic regression, using the same parameters, seem to lose their ability to identify anomalies. It is also interesting to notice the scores obtained for SVM. In Figure 4.14 we see that the SVM model has a good precision but poor accuracy and recall demonstrating the interest behind using multiple metrics to evaluate the skill of models. With regards to the Logistic Regression model, its poor skill can also be confirmed in the ROC curve, Figure 4.15. Following a certain threshold, the logistic regression model becomes worse than a no skill model.

All the curves, results and notebooks in which they were obtained, can serve as an important tool for ZORRO PoC purposes but also for future work to be done on the machine learning and anomaly detection aspects of ZORRO. This contribution paves the way for a new ZORRO ML Toolbox.

In conclusion, through these two anomaly scenarios we have shown that the developed platform is

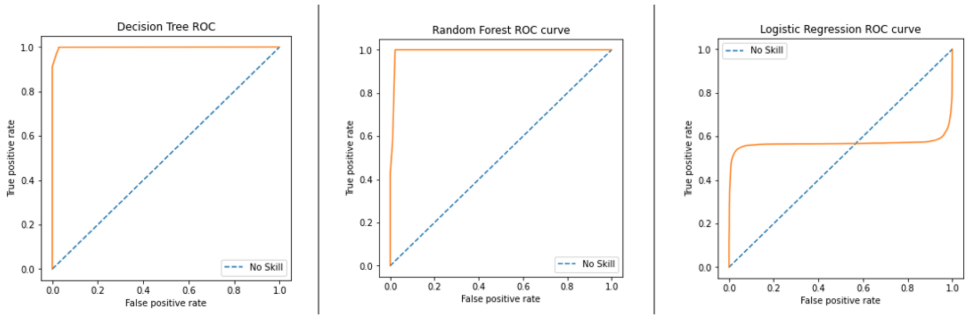


Figure 4.15: Second anomaly criterion - model ROC curves.

capable of performing a classical machine learning workflow essential for the development of effective models. Interesting model scores were also obtained, particularly when the anomalies depended on the relationship between features.

4.3.2 Anomaly Pipeline and Visualisation

As seen in section 3.11.4, a model that identified an anomaly would then signal that anomaly by sending a record to the Kafka Anomalies topic with all the information about the anomaly. Given that the predictions are written into Kafka, it is easy to then insert them into the TICK stack via Telegraf and visualise them in real-time.

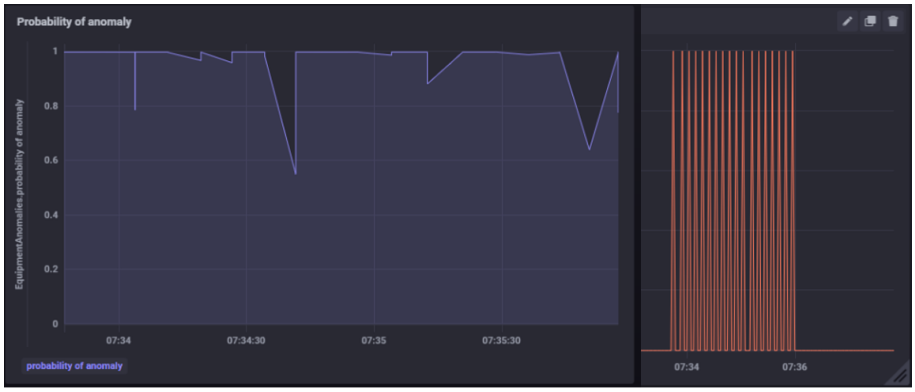


Figure 4.16: Visualisation of anomaly detection with a Chronograf dashboard.

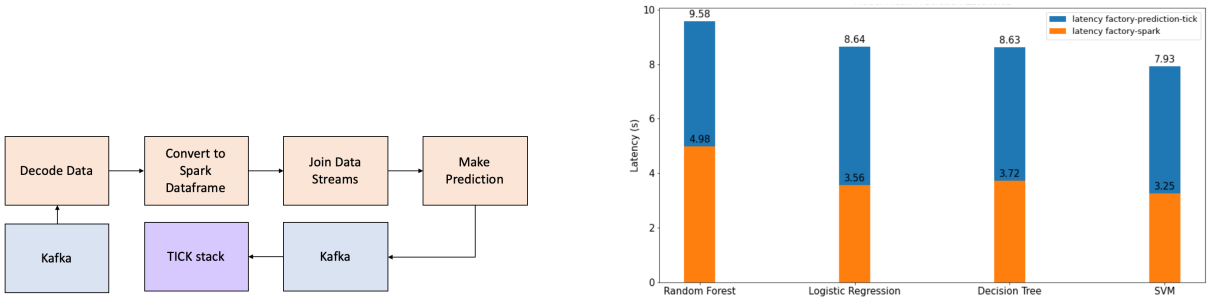
In Figure 4.16, the Chronograf dashboard on the right peaks every time an anomaly is identified. The graph on the left shows the probability with which the decision of labelling the anomaly was made. This dashboard was produced while simulating a factory floor with one equipment having anomalies. Other dashboards concerning the anomalies were created containing the values of the features of the equipment at the time of the anomaly. This provided a transparent and full analysis of the nature of the anomalies. These dashboards marked the completion of an important milestone in the ZORRO project as it is a real-time visual representation of the machine learning algorithms working to adding insights on the production status. Most importantly, the anomaly information and alerts are display in a human interpretable fashion. This was the original main objective of Project ZORRO. The architecture can now also be considered event-driven, which was another feature that was identified in the solutions of the

cloud providers, as certain events trigger certain actions. Another important consequence of achieving this milestone is the ability to now test the developed machine learning models in an environment that closely simulates a real-production environment.

Another interesting aspect of this result that was not yet explored was the possibility of automatic responses on the IoT layer. Given that Kafka is the central data pipeline of our architecture, once the anomaly data is in this pipeline, any of the ZORRO sub-systems can respond or react to the anomaly. This broadens the reach of the ZORRO event-driven architecture. In order to demonstrate that the developed platform is capable of automatically enriching customer databases, Kafka was connected to a PostgreSQL database using a Python Kafka consumer and PostgreSQL connection APIs.

4.3.3 Anomaly Prediction Latency

With the anomaly detection streaming pipeline tested and functioning correctly, the next aspect studied was the latency of this pipeline. Given that anomalies need to be identified in real-time so that the responses to these events can also be done in real-time, the levels of latency of this pipeline must be understood and minimised. Considering that different models operate in different ways, the latency in the prediction for each model will be different. The simulator was run for each model separately, with one equipment only (as each model should be trained for one equipment only). The objective was to be able to study the latency associated with each individual model. For each model, the simulator was run for 20 minutes and the results were calculated on the final 15-minute window (to make sure the system latency could stabilise at a certain value). Given that only one equipment was run, the amount of data the system was exposed to was around 32 KB/s or 20 messages/s. The simulator was run using the exact same anomaly configuration that the model was trained with. The model scenario chosen for the latency study was the sensor combination model (first anomaly criterion). The results are available in Figure 4.17.b. Figure 4.17.a represents the processing work-flow around the anomaly detection streaming process. It is the same process as described in section 3.11.4.



(a) Streaming work-flow.

(b) Mean latencies obtained for each model.

Figure 4.17: Prediction latency analysis results.

In figure 4.17.a, each column represents a latency when compared with the moment the timestamp of the reading was created in the factory simulator. The orange column represents the time it took for the data to arrive to Spark from the factory. This therefore involves communication from the factory to Kafka and then from Kafka to Spark. The blue column represents the time it took for an anomaly to

be written into the TICK stack (displayed in the dashboard). This is therefore the total time that it takes to identify an anomaly and display it in a human readable fashion. Other columns were calculated as an attempt to measure the latency of the inner operations represented in Figure 4.17.a (decoding, join and prediction). These latencies were found to be very close in proximity to each other. The latencies of these inner operations were calculated using timestamps inserted in the Spark streaming scripts that were developed. Given that Spark is a distributed system and that the operations in Spark are not necessarily followed in the order that they were programmed in (due to the Spark optimiser), it is believed that Spark inserted the timestamp at a time that did not really reflect the desired insertion of the timestamp. Therefore, the correct measurement of these inner operations could not be guaranteed and are therefore not presented.

From Figure 4.17, the first conclusion is that stream-to-stream joins along with a machine learning predictions clearly take time. We are no longer at the small communication latencies we identified initially with the TICK stack in section 4.1.3. The second conclusion is that this latency depends rather significantly on the model chosen and therefore the prediction stage, as the data load is the same in all other stages regardless of the model used. As expected, the Random Forest model is the one that takes the most time given that it must consult the predictions of multiple trees (300 were used). For all the other models, similar latencies were obtained with SVM having the lowest prediction latency. To understand if the system is stable and latency does not diverge, the latency evolution for these predictions was analysed. The results are illustrated in Figure 4.18.

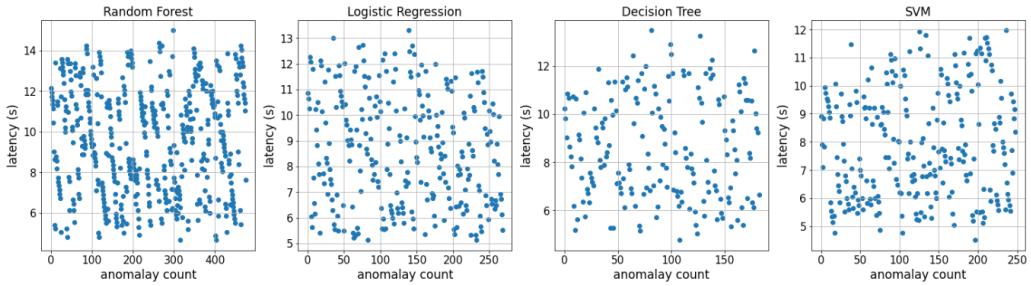


Figure 4.18: Latency evolution for each model.

For all models, the system seems to be stable as the latency oscillates around the mean that was calculated for each model. Likewise, it is important to remember that these latencies were calculated during a 15-minute window after the system was already given 5 minutes to reach a steady state. Figure 4.18 also shows how many anomalies each algorithm found. We can see that these values vary quite significantly as not only each model behaves differently but also the data simulated was different.

With these results, another question that is posed is the possibility of being able to reduce or tune this latency as a function of the use case or client requirements. In the case of our architecture, this latency could be reduced by attempting three things: scaling the Spark cluster thus adding processing capacity, Scaling the Kafka cluster or reducing the frequency (load) of the data production in the factory. It was concluded that scaling Kafka would not be very beneficial in this case as the communication latency obtained from it was already significantly small. This was seen in the TICK latency analysis, where Kafka was exposed to higher data loads. Nevertheless, increasing the number of partitions to 2

and adding another machine running a Kafka cluster could help in the parallelisation of the consumption. Scaling Spark would be a viable option, however, this cannot be done as a limited number of machines for testing purposes were available. It was therefore decided to reduce the message frequency per producer from 5 messages to 2 messages per second. This therefore corresponds to 8 messages per second being sent. The watermark for each reading was also slightly reduced. The results obtained with the same model as above are illustrated in Figure 4.19.

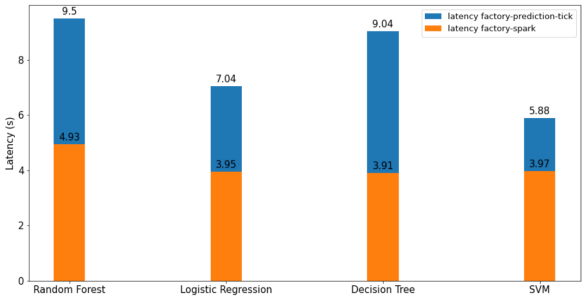


Figure 4.19: Anomaly detection latency with reduced simulator frequency.

From the results obtained we see that the latency has decreased for Logistic Regression and SVM but not at all for the tree-based algorithms. It appears that the Spark cluster is not necessarily overwhelmed with the amount of data but rather, there is a processing threshold that Spark is not being able to compensate for even with this reduced data load. To truly understand if the objective of 5 seconds for a prediction can be obtained, larger Spark and Kafka clusters are needed to increase the processing capacity. Another interesting solution could be to have Kafka assume some of the processing load. As seen in section 3.8, the Kafka streams java API can be used to perform topic stream-to-stream joins. This developed application could be used to perform the joins and write the joined data to another Kafka topic. Spark would then only have to consume the data from this topic, decode it, convert the JSON to a Spark DataFrame and then perform the prediction. Given that Kafka streams API is optimised to work with data from Kafka, this could be advantageous.

Even though the platform does not have the desired latency objective at the desired scale, we now have the platform and architecture setup and ready to be optimised so that the objectives can be achieved. The watermarking tool has proved to be an excellent out of the box tool offered by Spark. It appears to be however a heavy tool in terms of processing capacity. Nevertheless, the decoding and JSON conversion tasks should also not be forgotten as they too could explain part of the observed latency.

In conclusion, we have developed Spark structured streaming applications capable of detecting anomalies in real-time and writing them back to Kafka. A lambda architecture capable of using machine learning to provide online insights (anomaly detection) about the production, which was our original goal, was built. This is therefore the basis for a more optimised and robust Industrial IoT architecture ready to be used as a reference and as a Meta solution to guide AKKA's consultants in advising and implementing Industry 4.0 to its clients.

Chapter 5

Conclusions

In this chapter we will summarise the work and implementations done on the ZORRO system, the results obtained and some of their implications on the larger context of the project. Following this, the extent to which the developed ZORRO system can be considered a robust meta solution is discussed. The future work to be done will then be discussed considering the difficulties encountered and limitations identified. Considering the Masters degree in which this Thesis was carried out, a final discussion of its application on the Aerospace sector is also done.

5.1 Overview of Deployed System and Work Developed

As shown in Figure 3.6, in this work we have developed an Industrial IoT data collection and processing system, respecting Big Data constraints and adopting a lambda architecture allowing for both batch and streaming data processing. The developed system was used to visualise manufacturing data in real-time, perform data exploration and analytics, and to detect manufacturing anomalies in real-time using machine learning algorithms trained and deployed on the same platform. The system requirements were satisfied and the desired architecture produced. Returning to the three main layers introduced in the introduction of this work, in Figure 1.2, the following paragraphs sum up the work done on each of these layers.

On the IoT layer (factory simulator), optimisations were performed on the simulator code, the parallelisation of the simulator was done and the integration with a Kafka cluster was enabled. In addition, the simulator was adapted to have the data being communicated in JSON format.

On the ZORRO data pipeline and processing (TCP) layer, a significant number of implementations were done. A Kafka message broker cluster, acting as a central data pipeline, was deployed allowing for messages to be communicated in real-time (with very admissible latencies) to all systems that require it. In addition to this, the Kafka streams API was used to perform fast and scalable data processing tasks. A data lake, enabling historic storage of huge data volumes, consisting of a Cassandra database and an influxDB time series database was deployed. This gave way for batch persistent storage that can be read in parallel by the Spark distributed data processing system, and for time series specific opera-

tions. To ingest data into InfluxDB from Kafka, Telegraf was deployed. To ingest data into Cassandra, connectors were programmed. On the artificial intelligence and machine learning (ML) side of things, a Spark cluster was deployed and configured for data processing and implementation of ML models for automatic anomaly detection in real-time. This involved batch data processing jobs interacting with the data lake (Cassandra) such as joins, data labelling and ML model training, and speed layer (streaming) jobs, for the online predictions. An initial campaign was carried out to test different binary classifiers and their performance both from a prediction (score) and speed (latency) perspective. The models utilised data from different data sources, therefore time constraint table-to-table and stream-to-stream joins were done. Being more specific to anomaly detection and ML, this project has performed a preliminary analysis on unsupervised learning algorithms for anomaly detection (auto-encoder and others) and initial practical analysis on the supervised learning models available in Apache Spark (Decision Tree, Random Forest, SVM and Logistic Regression). Tools used to measure the performance of these algorithms were also studied and developed (ROC and precision vs recall curves). In terms of model skill (scores), some interesting results were already obtained as models demonstrated being capable of identifying anomalies dependant on relationships between variables successfully. All these aspects contributed to the creation of a ZORRO ML toolbox. Performance tests were carried out to quantify the general message latency and the prediction latency of the developed system. Despite not being integrated with a real IoT layer, the performance results obtained are valuable as they offer an idea of the performance and latency that data communication and processing would have once the data is already on the TCP layer.

On the serving layer, Chronograf was used for real-time dashboarding of both the anomalies and the general production / equipment status (asset monitoring). As a data pipeline, Kafka also allowed for automatic enrichment of a client specific (SQL) knowledge database through a python coded Kafka-PostgreSQL connector. A Jupyter notebook client was deployed creating an interactive and collaborative programming interface for data scientists and ML engineers to perform data exploration, batch processing and ML model training and testing using Spark's python API, Pyspark. Similarly, but on the data lake side of things, a Cassandra Zeppelin client was deployed for interaction with the raw data stored in Cassandra.

With regards to deployment, most of the elements of the ZORRO architecture were dockerised except for Spark, Jupyter and Zeppelin. Deployment states were defined using docker-compose files, where containers, networks and volumes were defined and created, rendering the architecture one step closer to being deployed on a cluster manager such as Kubernetes or docker-swarm. Given that the platform was also deployed on an internal OpenStack cloud, referring back to section 2.3.1, the developed ZORRO platform can be more easily deployed on a similar PaaS or even IaaS offering of any cloud provider.

Overall it can be concluded that a strong basis for a mature Industrial IoT data platform capable of satisfying the defined constraints related to Big Data and Cyber Physical systems has been developed.

5.2 Generation of Know-how for AKKA Technologies

As referred multiple times in this dissertation, one of the major objectives of this work was to generate know-how for AKKA regarding Industry 4.0, Big Data and stream-processing data systems. In addition to performing an extensive and documented state-of-the-art study, this objective was achieved through a consistent and detailed documentation that was developed parallel to the development and design of the ZORRO system. In particular, throughout the construction of the system, every two weeks, presentations were prepared to document the evolution of the system, the setbacks encountered and the additions made.

The configuration files used to build each each of the specific sub-systems and have them communicate with each other, the code produced to run each one of the data processing, table creation, data transferring and data visualisation / exploration jobs were stored in the functioning deployment environment and on a GitLab [6] code repository for future ease of instancing and use.

Having the system illustrated in figure 3.6 documented, built, deployed and functioning, we then proceeded in utilising it to perform a series of data exploration, data processing and machine learning campaigns in order to demonstrate its potential as discussed in chapter 4. Likewise, the system was tested with various data loads in order to understand its performance. The tools developed to test the system and analyse the results were documented and designed to be easily picked up and reproducible on other results.

Overall, all of the above mentioned elements contributed to the completion of the generation of internal know-how objective not only by providing a function proof of concept, but primarily through all the documentation that was produced around it. As a result, AKKA can use the developed ZORRO solution to not only understand the nature and potential o Industrial Big Data systems, but also the difficulties associated to these system.

5.3 System and Work Evaluation

The developed system has proved to be a functioning and architecturally stable solution respecting the various system requirements. An imperative next question to answer is the extent and ease to which it can be migrated towards customers operations. Put differently and simply, how plug and play is the developed solution? This is an important question to tackle as ease in integration and deployment is an important requirement of Industry 4.0 systems [20]. It is hard to correctly answer this question as it requires on terrain and industrial experience. Nevertheless, as it stands, the solution requires IT system engineers to implement it. Similarly, the computational capacity associated to the solution must be outsourced towards an external cloud provider. This, however, is not a problem as this ZORRO solution was already deployed and tested on AKKA's internal Openstack PaaS, built on AKKA's IT infrastructure, and can therefore be relatively easily transferred. Some aspects of the ZORRO serving layer are easily displayed to the user and do not require previous expertise. An example of these are the dashboards produced. On the other hand, the development and data science tools are targeted naturally towards

data scientists and the customers should have dedicated teams to receive these tools. Perhaps one of the most challenging aspects which the ZORRO project has not yet dealt with is data security and data ownership and how that will collide with industrial experts that might not have any experience with complex data and cloud systems. In conclusion, the ZORRO solution is becoming more stable technically and architecturally, but some eventual integration and partnership with an industrial company is desired, in order to better understand the onsite implementation challenges of ZORRO as an Industry 4.0 solution.

5.4 Future Work

Despite the advances made to the system, there is a significant amount of work to be done for the ZORRO system to be fully validated and ready to be demonstrated as a Proof of Concept to potential future clients and partners of AKKA.

As seen in section 5.3, one of the most important future aspects to work on in this project is the integration of the developed stack with a real IoT layer (network). The factory simulator proved essential and useful to build the current system, but it does not allow for reliable and efficient latency testing, as most of the times, significant latency amounts came from the factory simulator itself. Given that the simulator was coded using Python scripts, many aspects inherent to a real IoT layer were not at all simulated. In a real IoT layer, network connection may be limited, and sensors may have unexpected behaviour that was not simulated in our anomalies. Apache Kafka proved to be a viable technology for the creation of a unified data pipeline. We saw, however, that it was not the most suitable solution for direct integration with an IoT layer. For this reason, another essential step in the IoT layer would be to integrate Kafka with an MQTT broker such as HiveMQ, Mosquito or even the existing RabbitMQ broker. As an alternative, data could be collected in IoT gateways and communicated to the Kafka cluster directly through an MQTT proxy aggregating batches of data to be communicated to the Kafka cluster. These gateways that are generally used in large scale IoT systems [19], form part of the edge-computing domain of Industry 4.0 which has also not been explored in this work due to its proximity to the IoT layer. Understanding how edge computing could be used in an industrial context to aggregate and send Industrial IoT data to the developed ZORRO platform would also be an interesting aspect to explore.

Another aspect to work on is the data and communication security of the ZORRO platform. Although Kafka, the central piece of the developed data pipeline, allows for Transport Layer Security, this was not configured. Configuring it would securely encrypt all the inter-machine communications with the exception of communication of the elements from within the TICK stack and between Spark and Cassandra.

The deployment of almost all elements for the architecture was done using docker containers, for reasons discussed previously. To push the containerised deployment one step further, all elements of the architecture should be containerised and deployed using Kubernetes. This would allow for a real micro-service, fault tolerant and scalable Big Data architecture. It would also extend the know-how of the project further towards the popular Kubernetes cluster manager. Once this is done, a Continous

Integration, Continous Development Pipleine (CI-CD) could be implemented in order to further the ease in deployment of the ZORRO system as a Meta solution. It would also further increase AKKA's know-how regarding how these data systems are deployed, managed and maintained. All the code developed in this project was deployed in a GitLab repository which itself contains CI-CD tools that could be explored.

Regarding automatic anomaly detection, the optimisation of the existing developed models, using Spark's cross validation class, and the introduction of new models could be explored. Likewise, it would be interesting to test the sensitivity of the developed models to distributions that are not normal or uniform. This could include Poisson or exponential sensor distributions. Some new models include models that were explored in the state-of-the-art but were never implemented due to time restrictions. The most interesting ones to explore would be the auto-encoder artificial neural network (explained in section 2.4.5, and geometric distance-based algorithms using the Mahalanobis distance. It would also be important to understand how many false positives our trained algorithms produce in a situation of nominal equipment behaviour. Finally, the project could also go beyond automatic anomaly detection and into the realm of using machine learning for predictive maintenance, time series forecasting and supply chain resilience. In particular it could also be interesting to consider other classes in the classification process such as ”replacement soon” or ”high probability of future anomaly”.

To render the PoC architecture truly equal to a lambda architecture, automatic Spark jobs for table joining, data labelling and model training could be coded. This essentially would consist in automating the batch processing illustrated in Figure 3.15. In addition, an automatic interaction between the ZORRO speed and batch layer could be done, where the speed layer selects the most recent and the most high performing model trained in the batch jobs for predictions. Code has been written for all these tasks in both script and notebook form. Nevertheless, the automation of all these tasks is missing.

Finally, general more particular performance results need to be obtained for certain stages of the ZORRO system. For example, the speed of the table-to-table batch joins was not tested, nor was Spark's speed of decoding the the data from Kafka or JSON string to Spark DataFrame conversion. It would be important to begin by understanding how computationally expensive these tasks are before attempting to optimise the architecture.

5.5 Applications on the Aerospace Sector

Industry 4.0 can be used in a series of production contexts and the objectives and gains are rather transversal for all. Nevertheless, on the aerospace sector where very accurate and faultless manufacturing is required to meet customer and regulator critical norms, Industry 4.0 can be particularly interesting to help manufacturers in increasing the speed of production whilst still improving the quality. This was seen in works like [13] where an Industry 4.0 solution was used to train a machine learning algorithm to perform the deburring processes on aerospace parts automatically. This work proposed to perform this critical task, which is traditionally carried out manually by experts and thus with subjective judgement, using a deburring machine controlled with a machine learning algorithm making data based decisions. Another example was seen in [83], where an automatic blade compressor re-manufacturing Industry 4.0

system was proposed to improve the quality of blade re-manufacturing which previously was done by hand and only 45% of the blades actually were able to be reused. In these examples, Industry 4.0 is used to increase the quality and speed of aerospace manufacturing. These examples demonstrate that the use of data acquisition and intelligent information systems is essential to automate and produce with the quality and accuracy that aerospace manufacturing requires.

Quoting Jean-Brice Dumont, Executive Vice President of Engineering, Airbus [14] - ”Our challenge in the coming years is to manufacture more aircraft faster, and for that, we need to enable our workers to be much better equipped and to be much more effective in what they do. We need to raise the bar.” Here M. Dumont raises an interesting point concerning Industry 4.0 which should be thought of as a tool to help manufactures become more efficient. Real-time asset monitoring, for example, can help manufacturing workers on the production line identify anomalies quickly and visually before the parts or assemblies reach the final tests and can no longer be changed. Remote analysis of potential manufacturing problems can also save time and therefore making the manufacturing process faster and giving manufacturers more flexibility. This quote by M. Dumont is consistent with Airbus's wish to double its production of Aircraft in the next 20 years and at a speed that is twice as fast as discussed in section 2.2. With the current COVID-19 pandemic, this objective must be naturally adapted, however, the need for more efficient production processes continues to be present and has perhaps now grown.

The work developed in this dissertation is a step towards a clearer and more efficient manufacturing process. All of the tools developed aim in increasing the quality and capacity of decision making of production experts. From real-time asset monitoring and statistical visualisation of the production data to real-time anomaly detection and alerting, the insights that the developed system can provide are data driven, allowing for production experts to perform objective analysis. This tool therefore fits well with M. Dumont's objective of creating Industry 4.0 systems capable of making the Aerospace manufacturers more efficient, better equipped and more effective.

Bibliography

[1] Momentum. The evolution of industry 1.0 to 4.0. [Online] https://www.seekmomentum.com/blog/manufacturing/the-evolution-of-industry-from-1-to-4, 2019. Accessed 20 10 2020.

[2] V. Koch R. Geissbauer, S. Schrauf and S. Kuge. Industry 4.0 – opportunities and challenges of the industrial internet. PricewaterhouseCoopers, December 2014.

[3] McKinsey. Industry 4.0 after the initial hype. McKinsey Digital, 2016.

[4] Nancy Velásquez, Elsa Estevez, and Patricia Pesado. Cloud computing, big data and the industry 4.0 reference architectures. *Journal of Computer Science and Technology*, 18(03):e29–e29, 2018.

[5] AKKA. Akka technologies - we accelerate innovation and business for the digital industrial world. [Online] https://www.akka-technologies.com/. Accessed: 7 12 2020.

[6] GitLab. The complete devops platform. [Online] https://about.gitlab.com/. Accessed: 11 12 2020.

[7] AKKA. Autopilot. [Online] https://www.akka-technologies.com/case-study/platoon-traffic-light-assist-using-iot/. Accessed: 18 12 2020.

[8] Link & fly - solving future mobility challenges. [Online] https://www.akka-technologies.com/case-study/linkfly-air-transport-of-the-future/. Accessed: 21 12 2020.

[9] Euro Geo Surveys. Ingeoclouds. [Online] https://www.eurogeosurveys.org/projects/ingeoclouds/. Accessed: 19 12 2020.

[10] 5g-mobix – new 5g cross-border corridors for connected and automated driving. [Online] https://www.akka-technologies.com/press-release/5g-mobix-new-5g-cross-border-corridors-for-connected-and-automated-driving/. Accessed: 20 12 2020.

[11] Andreja Rojko. Industry 4.0 concept: background and overview. *International Journal of Interactive Mobile Technologies (iJIM)*, 11(5):77–90, 2017.

[12] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23, 2015.

[13] Wahyu Caesarendra, Tomi Wijaya, Bobby K Pappachan, and Tegoeh Tjahjowidodo. Adaptation to industry 4.0 using machine learning and cloud computing to improve the conventional method of deburring in aerospace manufacturing industry. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, pages 120–124. IEEE, 2019.

[14] Microsoft. Airbus reaches new heights with the help of microsoft mixed reality technology. [Online] https://news.microsoft.com/en-my/2019/06/17/airbus-reaches-new-heights-with-the-help-of-microsoft-mixed-reality-technology/. Accessed: 10 12 2020.

[15] Airbus. Skywise, the leading data platform for the aviation industry. [Online] https://skywise.airbus.com/. Accessed: 13 12 2020.

[16] SIEMENS. Unique automation portfolio. [Online] https://new.siemens.com/global/en/products/automation.html. Accessed: 19 12 2020.

[17] The leading industry 4.0 companies 2019. [Online] https://iot-analytics.com/the-leading-industry-4-0-companies-2019/, 2019. Accessed 06 12 2020.

[18] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[19] Khaled Al-Gumaei, Kornelia Schuba, Andrej Friesen, Sascha Heymann, Carsten Pieper, Florian Pethig, and Sebastian Schriegel. A survey of internet of things and big data integrated solutions for industrie 4.0. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1417–1424. IEEE, 2018.

[20] Waqas Ali Khan, Lukasz Wisniewski, Dorota Lang, and Jürgen Jasperneite. Analysis of the requirements for offering industrie 4.0 applications as a cloud service. In *2017 IEEE 26th international symposium on industrial electronics (ISIE)*, pages 1181–1188. IEEE, 2017.

[21] AWS. Aws iot. [Online] https://aws.amazon.com/iot/. Accessed 29 10 2020.

[22] Microsoft. Azure iot. [Online] https://azure.microsoft.com/en-us/overview/iot/. Accessed 29 10 2020.

[23] OpenStack. What is azure digital twins? [Online] https://docs.microsoft.com/pt-pt/azure/digital-twins/overview, 2020. Accessed 07 12 2020.

[24] Google. Google cloud iot. [Online] https://cloud.google.com/solutions/iot. Accessed 2 12 2020.

[25] P Sommer, F Schellroth, M Fischer, and Jan Schlechtendahl. Message-oriented middleware for industrial production systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, 2018.

[26] Emanuel Trunzer, Pedro Prata, Susana Vieira, and Birgit Vogel-Heuser. Concept and evaluation of a technology-independent data collection architecture for industrial automation. In *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 2830–2836. IEEE, 2019.

[27] Hyun-Seong Lee, Seoung-Hyeon Lee, Jae-Gwang Lee, and Jae-Kwang Lee. Designing a method of data transfer using dual message queue brokers in an iot environment. In *3rd IEEE/ACIS International Conference on Big Data, Cloud Computing, and Data Science Engineering*, pages 1–11. Springer, 2018.

[28] Peter Heller, Dee Piziak, and Jeff Knudse. An enterprise architect's guide to big data: Reference architecture overview. *Unpublished paper. Oracle Enterprise Architecture White Paper*, 2015.

[29] The Apache Software Foundation. Apache kafka more than 80% of all fortune 100 companies trust, and use kafka. [Online] `https://kafka.apache.org/`. Accessed: 11 12 2020.

[30] Neha Narkhede, Gwen Shapira, and Todd Palino. *Kafka: the definitive guide: real-time data and stream processing at scale*. " O'Reilly Media, Inc.", 2017.

[31] The Apache Software Foundation. Zookeeper. [Online] `https://zookeeper.apache.org/`. Accessed: 20 12 2020.

[32] Ayae Ichinose, Atsuko Takefusa, Hidemoto Nakada, and Masato Oguchi. A study of a video analysis framework using kafka and spark streaming. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2396–2401. IEEE, 2017.

[33] Somphop Chanthakit, Phongsak Keeratiwintakorn, and Choopan Rattanapoka. An iot system design with real-time stream processing and data flow integration. In *2019 Research, Invention, and Innovation Congress (RI2C)*, pages 1–5. IEEE, 2019.

[34] Khaled Al-Gumaei, Kornelia Schuba, Andrej Friesen, Sascha Heymann, Carsten Pieper, Florian Pethig, and Sebastian Schriegel. A survey of internet of things and big data integrated solutions for industrie 4.0. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1417–1424. IEEE, 2018.

[35] Spark. Apache spark. [Online] `https://spark.apache.org/`. Accessed 1 11 2020.

[36] Spark. Apache spark - documentation. [Online] `http://spark.apache.org/docs/latest/index.html`. Accessed: 1 11 2020.

[37] DB-Engines. Popularity changes per category, december 2020. [Online] `https://db-engines.com/en/ranking_categories`. Accessed: 20 12 2020.

[38] InfluxData. Tick stack. [Online] `https://www.influxdata.com/`. Accessed: 5 11 2020.

[39] Rui Liu and Jun Yuan. Benchmarking time series databases with iotdb-benchmark for iot scenarios. *arXiv preprint arXiv:1901.08304*, 2019.

[40] InfluxData. Influxdb. [Online] https://www.influxdata.com/products/influxdb/. Accessed: 20 12 2020.

[41] InfluxData. Telegraf. [Online] https://www.influxdata.com/time-series-platform/telegraf/. Accessed: 20 12 2020.

[42] Aleksey Kychkin and Aleksadr Nikolaev. Iot-based mine ventilation control system architecture with digital twin. In *2020 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pages 1–5. IEEE, 2020.

[43] Aisha Siddiqa, Ahmad Karim, and Abdullah Gani. Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18(8):1040–1070, 2017.

[44] Apache cassandra - manage massive amounts of data, fast, without losing sleep. [Online] https://cassandra.apache.org/. Accessed 07 12 2020.

[45] DataStax. Benchmarking top nosql databases apache cassandra, couchbase, hbase, and mongodb. eBook, April 2014.

[46] Katalin Ferencz and József Domokos. Iot sensor data acquisition and storage system using raspberry pi and apache cassandra. In *2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE)*, pages 000143–000146. IEEE, 2018.

[47] Dharavath Ramesh, Ashay Sinha, and Suraj Singh. Data modelling for discrete time series data using cassandra and mongodb. In *2016 3rd international conference on recent advances in information technology (RAIT)*, pages 598–601. IEEE, 2016.

[48] Docker. Apache spark - documentation. [Online] https://www.docker.com/. Accessed: 7 12 2020.

[49] Ruffin White and Henrik Christensen. Ros and docker. In *Robot Operating System (ROS)*, pages 285–307. Springer, 2017.

[50] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. " O'Reilly Media, Inc.", 2015.

[51] Docker. Docker docs - swarm mode key concepts. [Online] https://docs.docker.com/engine/swarm/key-concepts/. Accessed: 11 12 2020.

[52] Kubernetes. Kubernetes - production-grade container orchestration. [Online] https://kubernetes.io/. Accessed: 7 12 2020.

[53] Lambda architecture. [Online] http://lambda-architecture.net/. Accessed: 11 12 2020.

[54] hazelcast. What is lambda architecture? [Online] https://hazelcast.com/glossary/lambda-architecture/. Accessed: 20 12 2020.

[55] Repository dedicated to kappa architecture. [Online] https://milinda.pathirage.org/kappa-architecture.com/. Accessed: 20 12 2020.

[56] Jay Kreps. Questioning the lambda architecture. *Online article, July*, page 205, 2014.

[57] The smack stack is the new lamp stack. [Online] https://d2iq.com/blog/smack-stack-new-lamp-stack. Accessed: 20 12 2020.

[58] McKinsey. Digital manufacturing capturing sustainable impact at scale. June 2017.

[59] Tom M Mitchell et al. Machine learning. 1997.

[60] Toulouse Tech. Arbres binares de décision. [Online] http://wikistat.fr/pdf/st-m-app-cart.pdf, 2016.

[61] L.Fernandes. Understand the logistic regression algorithm. [Online] https://openclassrooms.com/en/courses/6389626-train-a-supervised-machine-learning-model/6405876-understand-the-logistic-regression-algorithm. Accessed 10 10 2020.

[62] Toulouse Tech. Machines à vecteurs supports. [Online] http://wikistat.fr/pdf/st-m-app-svm.pdf, 2016.

[63] Oleksandr I Provotar, Yaroslav M Linder, and Maksym M Veres. Unsupervised anomaly detection in time series using lstm-based autoencoders. In *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pages 513–517. IEEE, 2019.

[64] Di Hu, Chen Zhang, Tao Yang, and Gang Chen. Anomaly detection of power plant equipment using long short-term memory based autoencoder neural network. *Sensors*, 20(21):6164, 2020.

[65] Stanislav Kolenikov, Gustavo Angeles, et al. The use of discrete data in pca: theory, simulations, and applications to socioeconomic indices. *Chapel Hill: Carolina Population Center, University of North Carolina*, 20:1–59, 2004.

[66] MC.AI. Auto-encoder in biology. [Online] https://mc.ai/auto-encoder-in-biology/, 2019. Accessed 10 10 2020.

[67] Docker. Docker docs - overview of docker compose. [Online] https://docs.docker.com/compose/. Accessed: 11 12 2020.

[68] OpenStack. The most widely deployed open source cloud software in the world. [Online] https://www.openstack.org/. Accessed: 11 12 2020.

[69] RabbitMQ. Rabbitmq is the most widely deployed open source message broker. [Online] https://www.rabbitmq.com/. Accessed: 11 12 2020.

[70] Philippe Dobbelaere and K Sheykh Esmaili. Industry paper: Kafka versus rabbitmq. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 227–238, 2017.

[71] P Sommer, F Schellroth, M Fischer, and Jan Schlechtendahl. Message-oriented middleware for industrial production systems. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1217–1223. IEEE, 2018.

[72] Confluent. How to maintain message ordering and no message duplication. [Online] `https://kafka-tutorials.confluent.io/message-ordering/kafka.html`. Accessed: 14 12 2020.

[73] Confluent. The complete solution for cloud-native event streaming. [Online] `https://www.confluent.io/`. Accessed: 11 12 2020.

[74] Confluent. Internet of things (iot) and event streaming at scale with apache kafka and mqtt. [Online] `https://www.confluent.io/blog/iot-with-kafka-connect-mqtt-and-rest-proxy/`. Accessed: 20 10 2020.

[75] HiveMQ. Hivemq and apache kafka - streaming iot data and mqtt messages. [Online] `https://www.hivemq.com/blog/streaming-iot-data-and-mqtt-messages-to-apache-kafka//`. Accessed 2 10 2020.

[76] Lambros Sarakis, Theodore Zahariadis, Helen-Catherine Leligou, and Mischa Dohler. A framework for service provisioning in virtual sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):1–19, 2012.

[77] Wolfgang Thronicke Atos. Industry 4.0 aspects – machine interference and virtual sensors. [Online] `https://atos.net/en/blog/industry-4-0-aspects-machine-interference-virtual-sensors`, April 2015. Accessed: 14 12 2020.

[78] Apache Zeppelin. Web-based notebook that enables data-driven, interactive data analytics and collaborative documents with sql, scala and more. [Online] `https://zeppelin.apache.org/`. Accessed: 14 12 2020.

[79] Jupyter. Project jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages. [Online] `https://jupyter.org/`. Accessed: 14 12 2020.

[80] Apache Spark. Structured streaming + kafka integration guide. [Online] `https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html`. Accessed: 14 12 2020.

[81] Grafana. Metrics and logs at scale. [Online] `https://grafana.com/`. Accessed: 13 12 2020.

[82] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[83] Richard French, Michalis Benakis, and Hector Marin-Reyes. Intelligent sensing for robotic remanufacturing in aerospace—an industry 4.0 design based prototype. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pages 272–277. IEEE, 2017.