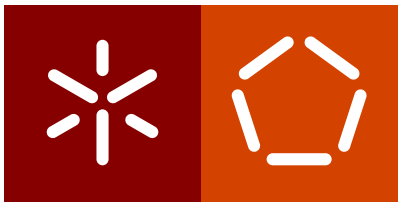
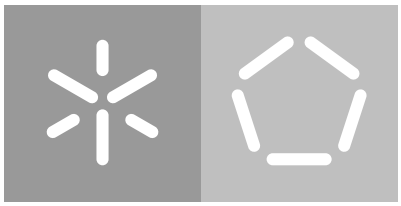


Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Pedro Nóbrega Rei

Study, Selection and Evaluation of an IoT Platform for Data Collection and Analysis for medical sensors

July 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Pedro Nóbrega Rei

Study, Selection and Evaluation of an IoT Platform for Data Collection and Analysis for medical sensors

Master dissertation
Master Degree in Medical Informatics

Dissertation supervised by
Professor Doutor António Luís Sousa
Francisco Cruz

July 2018

ACKNOWLEDGEMENTS

In the first place, I would like to express my gratitude to professor António Luís Sousa, my master thesis' adviser, for the opportunity to work in this area. I am really grateful for all the help and comprehension during the development of this dissertation.

To my lab colleagues in the HASLab group, I am really thankful for the integration and constant support in everything I needed. I would like to express my gratitude especially to Francisco Cruz, for all the help and guidance during my work in the lab.

I am also grateful to my family, for making all of this possible and always supporting me during my journey. Everything I have ever achieved, I owe it to you.

I would like to express my gratitude, in particular, to Joana, for being always by my side, in every victory and every defeat, not only during this master thesis, but almost my entire academic journey. Thank you for your patience, your comprehension and for being the most incredible person I know.

And finally, to my closest friends, thank you for always being there. I am grateful for all your patience and unconditional support.

ABSTRACT

Every day, huge amounts of data are generated in the healthcare environments from several sources, such as medical sensors, EMRs, pharmacy and medical imaging. All of this data provides a great opportunity for big data applications to discover and understand patterns or associations between data, in order to support medical decision-making processes. Big data technologies carry several benefits for the healthcare sector, including preventive care, better diagnosis, personalized treatment to each patient and even reduce medical costs. However, the storage and management of big data presents a challenge that traditional data base management systems can not fulfill. On the contrary, NoSQL databases are distributed and horizontally scalable data stores, representing a suitable solution for handling big data.

Most of medical data is generated from sensor embedded devices. The concept of IoT, in the healthcare environment, enables the connection and communication of those devices and other available resources over the Internet, to perform or help in healthcare activities such as diagnosing, monitoring or even surgeries. IoT technologies applied to the healthcare sector aim to improve the access and quality of care for every patient, as well as to reduce medical costs.

This master thesis presents the integration of both big data and IoT concepts, by developing an IoT platform designed for data collection and analysis for medical sensors. For that purpose, an open source platform, Kaa, was deployed with both HBase and Cassandra as NoSQL database solutions. Furthermore, a big data processing engine, Spark, was also implemented on the system.

From the results obtained by executing several performance experiments, it is possible to conclude that the developed platform is suitable for implementation on an healthcare environment, where huge amounts of data are rapidly generated. The results also made it possible to perform a comparison between the performance of the platform with Cassandra and HBase, showing that the last one presents slightly better results in terms of the average response time.

RESUMO

Atualmente, uma grande quantidade de dados é gerada todos os dias em ambientes hospitalares provenientes de diversas fontes, como por exemplo sensores médicos, registos eletrónicos, farmácias e imagens médicas. Todos estes dados proporcionam uma grande oportunidade para aplicações de big data, permitindo revelar e interpretar padrões ou associações entre os dados de forma a auxiliar no processo de tomada de decisão médica. As tecnologias de big data comportam diversos benefícios para o sector de saúde, incluindo a prestação de cuidados preventivos, diagnósticos mais eficientes, tratamento personalizado para cada paciente e até mesmo reduzir os custos médicos. No entanto, o armazenamento e a gestão da big data apresenta um desafio que os sistemas de gestão de base de dados tradicionais não são capazes de ultrapassar. Não obstante, as bases de dados NoSQL representam uma solução de armazenamento de dados distribuída e escalável horizontalmente, sendo, portanto, apropriadas para lidar com big data.

Uma grande parte dos dados médicos é gerada através de dispositivos embebidos com sensores. O conceito de IoT, no ambiente das unidades de saúde, permite a conexão e comunicação desses dispositivos e outros recursos disponíveis através da Internet, de forma a realizar ou auxiliar nas atividades de saúde, como por exemplo o diagnóstico, a monitorização ou até mesmo em cirurgias. As tecnologias IoT visam melhorar o acesso e qualidade dos cuidados de saúde para todos os pacientes, bem como reduzir os custos na prestação dos mesmos.

Esta tese de mestrado apresenta, assim, a integração de ambos os conceitos de big data e IoT, propondo o desenvolvimento de uma plataforma projetada para a recolha e análise de dados de sensores médicos. Para essa finalidade, foi utilizada uma plataforma IoT de código aberto, Kaa, juntamente com duas bases de dados NoSQL, HBase e Cassandra. Adicionalmente, foi também implementado um mecanismo de processamento de dados, também de código aberto, o Spark.

Com base nos resultados obtidos através da realização de diversas experiências de avaliação de desempenho, foi possível concluir que a plataforma desenvolvida é adequada para a implementação em ambientes de prestação de cuidados de saúde, onde grandes quantidades de dados são rapidamente geradas. Os resultados permitiram também realizar uma comparação entre o desempenho da plataforma com Cassandra e com HBase, realçando que esta última apresenta resultados ligeiramente melhores em termos do tempo médio de resposta.

CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

6LoWPAN IPv6-based Low-power Wireless Personal Area Network

AAL Ambient Assisted Living
API Application Programming Interface

CAP Consistency, Availability and Partition Tolerance
CQL Cassandra Query Language

DSR Design Science Research

EMRs Eletronic Medical Records

GFS Google File System

HDFS Hadoop Distributed File System
HIPAA Health Insurance Portability and Accountability Act

ICU Intensive Care Unit
IoT Internet of Things
IP Internet Protocol
IT Information Technology

JSON JavaScript Object Notation

m-health mobile-health
MVCC Multi-Version Concurrency Control

NoSQL Not Only SQL

RDBMS Relational Database Management System
RDD Resilient Distributed Dataset
RFID Radio Frequency Identification

SDK Software Development Kit
SQL Structured Query Language

UI User Interface

WAL Write-Ahead-Log
WSN Wireless Sensor Network

1

INTRODUCTION

The present dissertation explores the development of a platform for data collection and analysis for medical sensors, based on the concept of the Internet of Things. Section 1.1 presents a brief contextualization and motivation of this master thesis' topic, followed by its main goals in Section 1.2. Section 1.3 presents some contributions of this project and, finally, Section 1.4 exposes the document's structure.

1.1 CONTEXT AND MOTIVATION

The presence of embedded sensors in everyday life objects is rapidly increasing. Smartphones and wearable devices are just two examples of recent technologies that enable to bridge the gap between physical and cyber worlds [1]. Nowadays, there is an increasing interest in wearable sensors and there are already several devices available on the market for fitness, activity awareness and even personal healthcare [2].

Over the years, the access, affordability and quality of healthcare have always represented a problem around the world. In fact, a large number of people do not receive the quality care they need, either by geographical disadvantages or the high costs of medical care [3]. This way, researchers have been considering the application of smartphones and wearable technologies in the healthcare sector for remote health monitoring and clinical management of patients' physiological information [2]. Through the mobile application, sensors, medical devices and remote monitoring products, it is possible to improve the quality of care and reduce medical costs, as well as connecting people with healthcare providers [3, 4].

The concept of Internet of Things (IoT) enables the integration and communication of these network-connected devices to provide information concerning the health status of a patient in real time to the healthcare professionals [4]. Any device integrated in an IoT system is uniquely addressed and identifiable at any time and anywhere through the Internet [2]. These systems provide several benefits especially in the management of chronic diseases and remote monitoring systems, as well as reducing visits to the hospital and medical costs [4–6]. Additionally, by integrating all the available data from the IoT devices in decision-support systems, it is possible to provide better prognosis and treatments, early

interventions and life-style recommendations to improve the quality of the patient's health status [2].

Healthcare data is not only produced at an exponential rate but it also grows in diversity. Most of the data generated by medical devices and sensors is unstructured and, thus, requires different data storage mechanisms than the traditional database management systems. Furthermore, in order to analyze that huge amount of data, solutions based on cloud computing are also required [5].

Although there are already some IoT solutions on the market, the ambiguity and technical challenges are still prevailing, highlighting the deployment of an intelligent IoT-based healthcare platform that involves big data management as one of the biggest challenges [7]. This thesis aims to study the features required for that purpose, as well as to deploy and evaluate an IoT platform for data collection and analysis for medical sensors, using appropriate mechanisms for data storage.

1.2 OBJECTIVES

Having in mind the context and motivation for this master thesis, presented in Section 1.1, some objectives were established in order to conduct its whole development process. In the initial stage, it was necessary to gather a great amount of information, from different and heterogeneous sources, concerning the required components and issues for the implementation of the IoT platform in the healthcare sector. Afterwards, it was fundamental to design and deploy the system architecture and, finally, proceed with the platform evaluation process. Therefore, three main goals were established for this project:

- Assess the adoption of an IoT platform in the healthcare environment for storage and analysis of data for medical sensors:
  - Collect and analyze bibliographic documents about IoT applications, big data and database solutions in the healthcare sector.
  - Identification of the components required for building a platform to collect large amounts of data from medical sensors.
  - Identification of potential applications and medical sensors to integrate into the platform.
  - Identification of potential processing and analytic open source frameworks to integrate into the platform.
  - Study and selection of an IoT open source platform suitable for healthcare environments.

- Design and implement a system architecture for an IoT platform for storage and analysis of data for medical sensors:
  - Design of the system architecture proposed, based on the IoT platform selected, database solutions suitable for medical sensors and analytic applications.
  - Implementation of the designed system.
- Evaluate the performance of the developed platform:
  - Design and configuration of the experiments for the assessment process.
  - Representation and discussion of the results obtained from the assessment of the platform performance, based on the designed experiments.

1.3 CONTRIBUTIONS

The application of IoT technologies on the healthcare sector, along with big data storage and analytic solutions, represents one of the most promising applications nowadays. By developing an IoT platform designed for data collection and analytics for medical sensors, this master thesis is contributing to the spread of this concept as a reliable and beneficial solution to take into account for the healthcare sector.

A major contribution of this master thesis was the development and integration of a component to establish the connection between HBase data store and Kaa platform. The source code of this contribution was accepted and integrated in the platform source code, through GitHub, by the development team of Kaa.

1.4 DOCUMENT STRUCTURE

This master thesis is organized in seven chapters. This introductory chapter contextualizes the reader on the subject in study, as well as its motivations and objectives. A literature review on the main concepts and technologies addressed in this project is presented on Chapter 2. Chapter 3 exposes the research methodology adopted, while Chapter 4 presents the proposed approach and the system architecture for the platform development. Chapter 5 addresses the decision making and implementation processes in the deployment of the platform. Chapter 6 is dedicated to the assessment of the platform performance, highlighting the experimental setup, the obtained results and the respective discussion. The fundamental conclusions of this master thesis and the prospects for future work are presented in Chapter 7.

2

LITERATURE REVIEW

This chapter describes the theoretical foundations and scientific concepts concerning the development of this thesis, as well as the state of the art and literature review. Firstly, in Section 2.1, it is presented a brief introduction containing the major subjects of this chapter. Section 2.2 presents the concept of nowadays big data, its challenges and opportunities in healthcare environments and, finally, the emerging big data technologies. Subsequently, Section 2.3 presents in detail the Apache Software Foundation projects relevant for this dissertation. Section 2.4 discusses the concept of Internet of Things and its relationship with big data, highlighting the applications and benefits in the healthcare industry and making reference to security requirements and existing platforms. Finally, Section 2.5 is dedicated to the Kaa platform overview, which is the IoT platform deployed in this thesis project.

2.1 INTRODUCTION

Nowadays, healthcare organizations already store their medical records in electronic databases and there are even advances towards data transparency by making that stored data usable, searchable and actionable by the healthcare sector as a whole [8]. As healthcare data is rapidly increasing, healthcare providers and data scientists now have access to promising new threads of knowledge, called "big data" as result of not only its volume but also its complexity and diversity [8, 9]. The analysis of this data, in order to obtain useful insight, allows healthcare providers to deliver higher quality care and reduce costs. [8, 10]. Many innovative companies in the private sector are already building applications and analytic tools that help patients, physicians and other healthcare stakeholders to identify value and new opportunities based on all this emerging data [8, 10].

However, there are some obstacles in healthcare systems that big data has to overcome in order to succeed [11, 12]. Patient privacy needs to be protected as more information becomes public and big data systems need to be carefully developed to satisfy storage and analytic requirements [8, 11, 13]. Thus, new data management systems have been developed to face these challenges such as Hadoop [14], an open source framework that helps solve

problems related to storage and access to data and allows very fast parallel processing; HBase [15], an open source distributed database that is built on top of Hadoop's file system; and Cassandra database [16], that can store millions of columns in a single row and does not require prior knowledge of data formatting [13, 17]. Additionally, for big data analytics purposes Spark [18] is one of the most widely used open source processing frameworks [19].

One of the major sources of big data is the Internet of Things (IoT). This concept is based on the integration of everyday objects with sensing and networking capabilities to communicate with all devices or services over the Internet [20]. This way, healthcare represents one of the most potential and attractive areas for the implementation of IoT solutions, especially in remote health monitoring, fitness programs, chronic diseases and elderly care. In addition to the possibility of applying big data analytics on the huge amount of data generated by these systems, IoT applications in healthcare are also expected to reduce medical costs, provide specialized treatments and increase the users' quality of life [21]. In order to achieve those goals, many IoT platforms integrated with cloud based storage solutions, such as Kaa [22], have been developed. [1, 23].

2.2 BIG DATA

2.2.1 *Overview*

Everyday, web and databases are overloaded with huge amounts of data in some way that by 2025 the forecast is that the Internet will exceed the brain capacity of everyone living in the whole world [17]. Data can be generated by different kinds of sources such as online transactions, emails, videos, audio, images, click streams, logs, posts, search queries, health records, social networks, science data, sensors and smartphones [24]. Advances in digital sensors, communications, high-speed network connections and computation technologies contribute to a massive increase of this data, thus creating huge data collections [17, 25]. In addiction, the development and deployment of business-related data standards, electronic data interchange formats, business databases and information systems improved the creation, storage, communication and utilization of all that data [25]. This phenomenon is known as big data [17].

In August 2012, a report delivered to the U.S. Congress defined the concept of big data as "large volumes of high velocity, complex, and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management and analysis of the information" [10]. In fact, big data is commonly characterized by the following 5 V's [11, 17]:

- **Volume:** Huge amount of data is generated every second. All this data enforces the need of advanced tools to store and process efficiently all the information [9, 17].
- **Variety:** Data does not have a fixed structure. It can be highly structured (data from relational databases), semi-structured (web, social media, sensor source) or even unstructured (audio, video, clicks) [9, 13, 17].
- **Velocity:** The term "velocity" includes not only the speed of incoming data but also the speed that data flows in the system. It involves data streaming, structured records creation, data processing speed and availability for access and delivery [12, 13, 17].
- **Value:** The value refers to the purpose and potential of big data such as human decision support, discovering needs, segmenting populations or enabling new business models, products or services [17, 26].
- **Veracity:** It is associated to the uncertainty of data, which can be caused by inconsistencies, model approximations, incompleteness or even fraud. With many forms of big data, sometimes there is a lack of quality and accuracy [9, 17, 26].

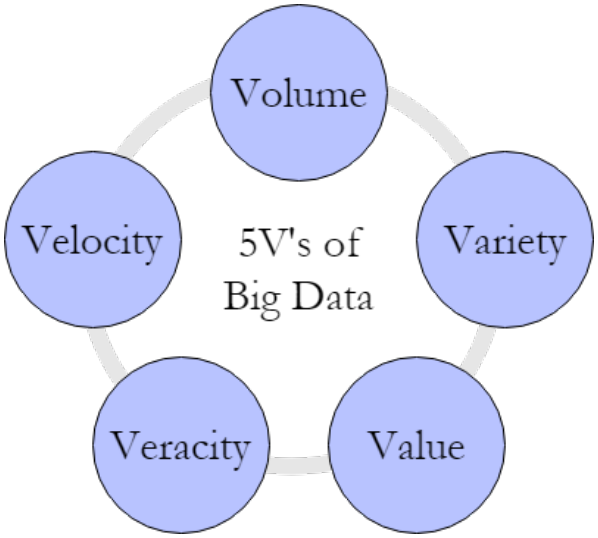


Figure 1: Representation of the 5V's characteristic of big data: Volume, Variety, Velocity, Value and Veracity [27].

In the last years, big data analytics have become of major interest and value for both academic and business communities. It is usually referred as the techniques, technologies, systems, practices, methodologies and applications that analyze critical data to obtain new discoveries and insight of relevance to the organizations [12, 25]. The analysis of big data provides useful information that help in making judgments, suggestions, supports or decisions for the stakeholders. In fact, extracting insight from big data may be useful in economic sectors, as well as improve the productivity and competitiveness of enterprises

and public sectors, creating huge benefits for both providers and costumers [28]. Several key application fields of big data such as enterprises, finances and online social-networks are introduced in [28]. In the same way, some promising areas of analytic applications such as e-commerce and market intelligence, e-government and politics, science and technology, smart health and well-being and security and public safety are presented and discussed in [25]. In [24] it is presented the potential of big data in some different sectors, highlighting the healthcare industry as one of the most important and emerging technologies for the next years [10, 12, 13, 24].

2.2.2 *Big Data in Healthcare*

The term "big data" in healthcare is related to all electronic health datasets that are difficult to process with traditional medical software and require some advanced data management tools [10, 12]. Healthcare data has increased massively not only because of the volume but also because of the data diversity and speed at which it is generated and must be recorded. In 2012, it was estimated to be equal to 500 petabytes (10^{15} bytes) and it is expected to reach 25,000 petabytes in 2020 [29]. It includes clinical information from sources like physician's written notes, medical imaging, laboratory, pharmacy, Eletronic Medical Records (EMRs), sensors (such as vital signs monitoring devices), biometric data and even smartphone health applications [10, 12].

All this amount of data creates a great opportunity for data scientists, allowing them to discover and understand patterns and associations between data to support medical decision-making [10, 26, 30]. Therefore, data analytics applications take advantage of that data in order to extract useful insight to provide several benefits including preventive care, by detecting diseases or possible complications at earlier stages when it is possible to treat them more efficiently; better diagnosis and personalized treatment to each patient; understanding and managing population health; predicting and/or estimating length of stay of patients, individuals at risk for medical complications or who will likely not benefit from surgery; and medical fraud detection [30]. By aggregating and integrating big data it is possible to extract deep knowledge about patient similarities and connections to provide personalized care for each individual patient derived not only from his EMRs information but also from his similarities with millions of other patients. This opens new opportunities for proactive medicine, actively managing diseases, empowering the patients and reduction of readmission rates [30]. With all this data-driven approach, healthcare is moving from a disease-centered model towards a patient-centered model. In the first model, physician's and healthcare professional's decision making is based on clinical expertise and medical evidence. In the patient-centered model, the patient receives services focused on individual

needs and preferences, or in other words, personalized care to benefit from accurate and better services [26, 30].

In addiction, applying advanced analytics based on healthcare data and patient-centered model also results in higher quality care at lower costs. *McKinsey*, a worldwide management consulting firm, estimated that big data analytics can save more than $300 billion per year only in U.S. healthcare [10, 31]. The list below presents some areas where applying big data analysis could help reduce waste and improve efficiency:

- **Clinical operations:** Extracting insight from data could help determine the best way for each patient diagnosis and treatment at lower costs [10, 32].
- **Research and development:** Using statistical tools and analyzing clinical trials and patient records could help improving clinical trial designs and patient recruitment in order to reduce trial failures and speed new treatments to market [10, 13].
- **Public health:** Analyzing disease patterns, outbreaks and transmission improve public health surveillance and speed response [9, 10, 13].
- **Evidence based-medicine:** Analyzing EMRs data and patient data could help predicting patients at risk for disease or readmission, and thus, providing more efficient care [9, 10, 25].
- **Remote monitoring:** Analyzing real-time large volumes of data from healthcare centers or home devices enables safety monitoring and prediction of any complication, reducing costs associated with patient readmission [9, 10, 25, 32].
- **Patient profile analytics:** Analyzing patient profile and characteristics makes it possible to identify individuals who would benefit from preventive care or lifestyle changes in order to avoid possible future complications and interventions [9, 10, 13].

Due to nowadays' large stream of data, from various sources and in many different forms, big data projects need to be carefully planned to set up the right goals and realistic expectations. Their success depends on the ability to develop efficient systems for integrating and processing all the information available [12, 17]. Furthermore, there are several challenges surrounding the concept of big data in healthcare like confidentiality and data security, storage, access to information, data reliability, interoperability and data management [11, 12, 33]. Concerning patient's privacy, in some healthcare organizations data may be shared only after de-identification, which prevents direct and indirect identification of the patient [33]. In U.S., according to the Health Insurance Portability and Accountability Act (HIPAA) there are many data elements that need to be removed to assure patient's privacy, such as names, geographic data, dates and contacts. In [33] there are listed all the restricted data elements in accordance with HIPAA. As for data storage, reliability, access and management there have been developed and tested several solutions [28].

2.2.3 *Storage and Analytics Solutions for Big Data*

In order to handle big data challenges for developing storage and analytics solutions, it is crucial to understand all its implications. For that purpose, big data management can be split in four steps [17, 34]:

- **Acquisition:** The system architecture has to acquire high speed of data from several different sources and diverse access protocols. If there is a need to filter the incoming data, this step is where it should be done.
- **Organization:** The system architecture must be able to parse data with different formats and extract the actual information that it contains. After that, data has to be integrated and stored in the right location, such as data warehouses, data marts, NoSQL databases, etc.
- **Analyze:** This step comprises running queries, modeling and building algorithms in order to extract new insights from data.
- **Decision:** By interpreting the results from the Analyze step, it is possible to make valuable and efficient decisions.

From this point of view, to be able to benefit from all the advantages of big data, it implies having an infrastructure capable of handling the steps above efficiently. That means this infrastructure should be linearly scalable, able to handle high throughput data, fault tolerant, auto recoverable and with high degree of parallelism and distributed data processing [17].

Traditional data management systems were based on the Relational Database Management System (RDBMS) [34]. However, these systems require structured data which is not suitable for handling big data, where large amounts of semi-structured and unstructured data can be rapidly generated [35]. Furthermore, RDBMSs are strictly designed with a lack of scalability and expandability, compromising the system performance when dealing with huge amounts of data. In sum, traditional RDBMSs could not handle the huge volume and heterogeneity of big data, so other solutions have been proposed [34, 35].

Cloud Computing

According to the National Institute of Standards and Technology, cloud computing is a "model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. network, servers, storage, applications and services) that can be rapidly provisioned and released with a minimal management effort or service provider interaction" [36]. This way, cloud computing provides the underlying engine needed to perform distributed data-processing of multiple datasets, characteristic of big data [37]. Overall, this model aims to provide higher scalability, elasticity, easy access through web,

lower operating and maintenance costs, as well as lower business risks [38]. These properties make cloud computing one of the most promising solutions for big data in healthcare as they provide a solution for interoperability problems and healthcare data sharing, processing and management [39].

In order to succeed in cloud environments, a cloud data management system requires [38]:

- **Scalability and high performance**, due to the continuous growth of data that need to be stored, users and data throughput.
- **Elasticity**, since cloud applications can face fluctuations in access patterns.
- **Fault tolerance**, because of the possibility of some machine's failure.
- **Security and privacy**, since data is stored in third-party resources.
- **Availability**, as applications cannot afford extended periods of downtime.

As it has been previously stated, traditional RDBMSs do not meet the requirements above. In order to address this issue, a new family of scalable databases, called NoSQL, has been developed in the last decade [36, 38].

NoSQL Databases

For permanent storage and management of large volume of data, NoSQL databases proved to be a good option [34]. NoSQL stands for "Not Only SQL", highlighting that Structured Query Language (SQL) is not a crucial point in these systems [35, 38]. In fact, applications such as big data analytics, business intelligence and social networking require too much effort from SQL-like centralized databases, pushing them over their limits. NoSQL systems are distributed, horizontally scalable, non-relational databases designed for large-scale data storage and processing [40]. The term "horizontally scalable" means that the system has the ability to distribute both the data and the load of the operations over many servers without sharing RAM or disk among them [41]. NoSQL data stores also offer flexible schemas or can even be schema-free, allowing to handle a wide variety of data structures. The horizontal scalability, the schema flexibility, the high availability and fault tolerance of these databases make them especially suitable for use as cloud management systems [38].

NoSQL database storing mechanisms follow the concept of CAP theorem, formulated by Eric Brewer in 2000, which stands for Consistency, Availability and Partition Tolerance [40, 42]. It postulates that only two of the previous three properties can be fully achieved simultaneously [36, 38, 40, 42]. This theorem is represented in Figure 2.

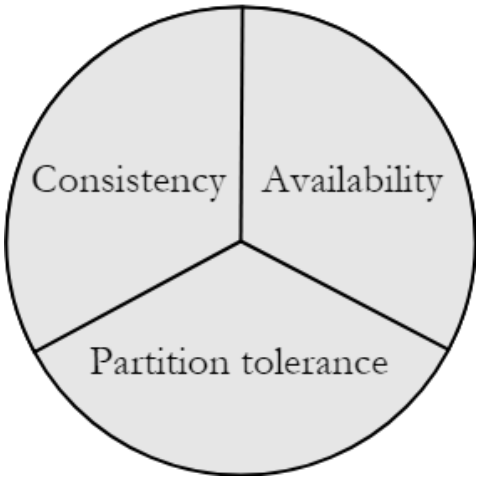


Figure 2: The CAP theorem states that only a combination of two of the following properties can be fully achieved simultaneously: consistency, availability and partition tolerance [42].

Consistency means having a single up-to-date instance of data [38]. This way, all clients who read from the database will always see the latest version of data [36, 40]. Availability implies data to be accessible at the moment that it is requested [38]. This means that all clients always find at least one copy of the requested data, even if some machines in the cluster are down [40]. Finally, partition tolerance refers to the capability of the system to tolerate network partitions [38]. This means the database still can perform read and write operations when some parts of the system are completely inaccessible [36]. Regarding the different concerns of NoSQL databases, there are three possible scenarios [42]:

- **Concerned about consistency and availability:** Mainly use of replication approach to ensure data consistency and availability. Some examples include Vertica [43], Aster Data [44] and Greenplum [45].
- **Concerned about consistency and partition tolerance:** Data is stored in distributed nodes and these systems privilege the assurance of data consistency instead of availability. Some examples include BigTable [46], HBase [15] and MongoDB [47].
- **Concerned about availability and partition tolerance:** This kind of systems privileges data availability over consistency. Some examples include Voldemort [48], CouchDB [49] and Cassandra [16].

NoSQL databases can also be classified as key-value databases, document databases and column-oriented databases [42, 50]. In key-value databases, data is stored as key-value pairs [50]. The key identifies uniquely the value, and it is used to store and retrieve the value into and out the data store. The value is opaque to the data store, so it can be used to store any arbitrary data. However, due to this property, these data stores cannot perform data-level querying [38]. Some examples of these databases include Dynamo, Voldemort and Redis

[40]. Document databases are especially suitable for applications where the input data can be represented in a document format [38]. Data is stored and organized as a collection of documents with any number of fields and length [50]. Those documents are usually represented using JavaScript Object Notation (JSON) or some derived format [38]. These data stores are not particularly concerned about high performance write operations, but to ensure big data storage and good query performance [42]. Typical document databases are CouchDB and MongoDB [41, 42]. Finally, most of column-oriented databases derived from Google Bigtable. In Bigtable, datasets consist of several rows where each row is addressed by a unique key. Each row is comprised by a set of column families and each column family is comprised by one or more columns [38]. The full description of Bigtable is presented in [46]. The deployment of column-oriented databases in an architecture with data compression and parallel processing can provide high performance of data analysis and business intelligence processing [42]. Some examples of these data stores include Google's Bigtable, HBase (which is the direct open source implementation of Bigtable concepts) and Cassandra [38, 41, 42].

Big Data Technologies

Nowadays, most of big data platforms and frameworks are based on distributed storage of data, which, unlike traditional systems, stores blocks of very large files across multiple nodes. These storage systems are designed to run on low-cost hardware and provide high availability and streaming access to data [35]. In this perspective, Hadoop has been a revolutionary technology in the realm of computer science [51].

Apache Hadoop is an open source framework that allows distributed processing of huge datasets across clusters [31]. It has not only gained significant importance in business intelligence and analytics [25], but also has become common in healthcare industry [9, 31]. In fact, it is the most widely applied technology in big data systems as it helps to overcome important challenges like storage and access, management of overheads (associated with the large datasets) and fast parallel processing [13]. Hadoop's main infrastructure consists in its default distributed file system and MapReduce [35, 52].

MapReduce is a programming paradigm introduced by Google in order to process huge amounts of data [35]. Some use cases of MapReduce framework in healthcare industry include finding optimal parameters for lung texture classification using machine learning algorithms, content-based medical imaging indexing and wavelet analysis for solid texture classification [31].

Despite providing high scalability across many servers in Hadoop cluster, MapReduce does not perform very well with intensive input-output tasks [31]. Apache Spark is another computing engine that can be integrated in Hadoop cluster for data processing. It supports in-memory computing, which allows to query data much faster than disk-based engines

like Hadoop's MapReduce [53]. In fact, for performing analytics on continuous streams of data, Spark proves to be very useful due to its capabilities to compute on streaming data with machine learning algorithms and graphic tools for visualization. This way, it is possible to perform both real-time and retrospective analysis of incoming data [31].

Currently, software and hardware vendors like IBM [54], Cloudera [55] and Dell EMC [56] offer tools, platforms and services for big data analytics based on Hadoop architecture [57]. Other big data technologies, including the previous mentioned NoSQL databases, are presented and described in [17] and [10]. Some big data tools such as Apache Zookeeper [58] for managing resources and Apache Flume [59] for data integration are presented in [35]. Actually, many Apache Software Foundation projects are developed to meet the requirements of big data systems.

2.3 Apache Software Foundation Projects

2.3.1 *Apache Hadoop*

The first version of Hadoop was created in 2004 by Doug Cutting, who named the project after his son's stuffed elephant. In January 2008, it became a top-level Apache Software Foundation project [60]. The Apache Hadoop project is responsible for developing open-source software for reliable, scalable and distributed computing. Hadoop software library is a framework that provides distributed storage and processing of large datasets across clusters of computers using simple programming models. It is designed to scale up from one to thousands servers and to detect and handle failures [14]. The storage is provided by Hadoop Distributed File System (HDFS) and the processing and analysis by MapReduce. There are other components that can be integrated in Hadoop, but these two are considered its kernel [61].

Hadoop Distributed File System

HDFS is an open-source fault-tolerant distributed file system, inspired by Google File System (GFS), that supports large data storage and management [60, 62]. It is designed to be deployed on low-cost hardware and to provide high throughput access to application data [14]. HDFS stores data in blocks, typically with size of 64MB, and each block is stored as a separate file in the local file system [35, 63]. The reason for this large size of blocks is to reduce the number of disk seeks [52, 61].

The HDFS cluster has a master/slave architecture [62, 64]. It consists of a single *NameNode*, which is considered the master server, and several *DataNodes*, usually one per node in the cluster [64]. The *NameNode* is a centralized service operating on one node in the cluster and it is responsible for maintaining the file system directory tree and managing the file

system namespace. It deals with clients requests to perform file system operations, such as open, close, rename and delete [63]. *NameNode* does not store HDFS data. Instead, it maintains a mapping between the HDFS file name, a list of blocks in the file and the *DataNodes* that store those blocks [63]. This way, *NameNode* acts as the arbitrator and repository of HDFS metadata [14]. *DataNodes* are responsible for handling read and write requests from clients and perform operations such as block creation, deletion, and replication when the *NameNode* requests. They retrieve blocks not only when they are told to, but they also report back to the *NameNode* the lists of blocks they are storing, periodically, in order to keep it up to date on their current status [65]. Note that although it is the *NameNode* that manages the namespace and block operations, clients are addressed to *DataNodes* in order to read or write data [63]. The HDFS architecture is represented in Figure 3.

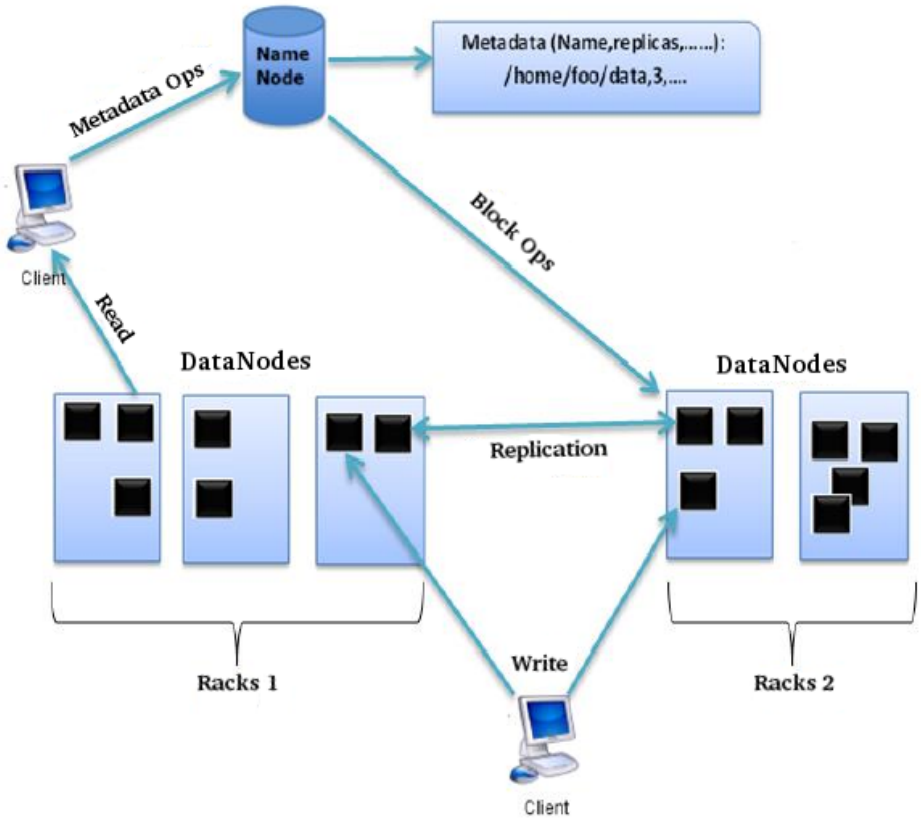


Figure 3: Architecture and workflow of Hadoop Distributed File System (HDFS). The *NameNode* manages metadata operations and gives instructions on block operations to *DataNodes*. When clients send operations requests to HDFS, the *NameNode* addresses them to the respective *DataNode* in order to perform read or write operations. Adapted from [35].

Without the *NameNode*, the file system cannot be used because there would be no way of knowing how to reconstruct the files from the blocks on the *DataNodes*. Thus, it is extremely

important to make sure the *NameNode* is resilient to failure. Hadoop provides two ways for achieving that. First, Hadoop can be configured so that the *NameNode* writes its persistent state to multiple file systems in order to backup all metadata information. Second, Hadoop can also run a secondary *NameNode* that is responsible for merging the namespace image periodically. It keeps a copy of the merged namespace which can be used in the event of the *NameNode* failing [61].

HDFS is designed to store huge amounts of data across machines in a large cluster in a reliable way [14]. Thus, it implements an automatic replication system for granting fault-tolerance [63]. The number of replicas is configurable through the replication factor property on HDFS configuration. Each file is stored as a sequence of blocks, which are replicated under that replication factor [14]. HDFS replication is also transparent to the client application. When a block is written in one *DataNode*, it echos the data to a second one, and so on until the desired number of replicas have been achieved. When the *NameNode* receives the list of stored blocks from each *DataNode*, it will verify if each block is sufficiently replicated. In case of failure, the *NameNode* instructs *DataNodes* to make additional replicas [63].

Data corruption can occur in any system due to faults in storage devices, network faults or even buggy software. In order to prevent that, HDFS implements checksum verification mechanisms on its contents for granting data integrity. When a client creates an HDFS file, it computes a checksum for each block and stores them in a separate hidden file in the same HDFS namespace. Then, when a client retrieves file contents, it verifies that the data received matches the checksum previously stored [14].

MapReduce

Hadoop MapReduce is the most popular open-source implementation of the MapReduce framework introduced by Google [66]. MapReduce provides a powerful parallel programming technique for distributed processing of data on clusters. The fundamental method of this programming paradigm is to break down the complex big data problems into small units of work and process them in parallel [24]. MapReduce, as the name suggests, expresses the fact that two different functions are performed: Map and Reduce [17, 52]. The Map function is applied on the input data in order to produce a set of intermediate results in the form of $< key, value >$ pairs [17, 63]. Then, the records for any given key (possibly spread across many nodes in the cluster) are aggregated at the node running the *reducer* for that key. This step involves data transfer between machines, so the Reduce stage is blocked from progressing until all the data from the Map stage has been transferred to the appropriate machine [60]. Finally, the Reduce function merges all the intermediate values associated with the same key [17]. Those two functions can run independently on each $< key, value >$ pair, exposing large amount of parallelism [63]. Usually, user applications only have to

define the Map and Reduce functions and the framework takes care of everything else, such as failover and parallelism mechanisms [66].

In Hadoop cluster, a job corresponds to a MapReduce program and it is executed by subsequently breaking it down into small parts called tasks. When a node in the cluster receives a job, it divides it and runs it in parallel with other nodes. This procedure is led by *JobTracker* [17]. The *JobTracker* receives the MapReduce job from the user application and splits it into tasks. Subsequently, it assigns those tasks to *TaskTrackers* on *DataNodes*. Lastly, the *TaskTracker* of each *DataNode* executes the assigned task on its data. *JobTracker* is responsible for the coordination of the process and, thus, it is in continuous communication with *TaskTrackers*, which periodically report the task status [35, 63]. When Hadoop detects task failures, it restarts the task in another healthy node, granting the fault-tolerance of the system [60].

Despite being a simple programming model, restricted to the usage of key-value pairs, there is a surprising number of tasks and algorithms that fits into this framework [60].

2.3.2 *Apache HBase*

HBase is an Apache Hadoop-based project modeled on Google's BigTable database [60]. The architecture of BigTable can be found on [46]. HBase can be defined as a distributed, sparse, persistent and multidimensional sorted map [15, 67]:

- **Distributed:** Data is stored in multiple nodes across the cluster.
- **Sparse:** Usually a record has many columns and some of them may have null data. HBase can efficiently save the space in sparse data.
- **Persistent:** Data is written and saved in the cluster.
- **Multidimensional:** A row can have multiple columns.
- **Map:** Data is stored in form of key-value pairs.
- **Sorted:** The key is stored in a sorted way for faster read and write optimization.

HBase is a column-oriented database built on top of the HDFS [68]. Therefore, the issues surrounding data replication, node failure and data distribution across the nodes are handled by the HDFS [50, 69]. HBase is designed to provide random real-time read/write access to very large datasets [60, 61].

Data Model

HBase data model is presented in Figure 4 and comprises [15, 70, 71]:

- **Row:** Each table comprises a set of rows where each row is identified through a unique row key. This is just a logical representation since the data is not stored physically in row, but in columns. In HBase a row consists of its row key and one or more columns associated with their values.
- **Column:** A column consists of a column family and a column qualifier delimited by a ":" character (*family*:*qualifier*).
- **Column family:** Column families physically stores a set of columns and their respective values. Each column family has a set of storage properties that can be configured, such as compression, cache in memory, etc. Every row in a table has the same column families, however a row might not store anything in a given column family.
- **Column qualifier:** A column qualifier addresses the data on the column family. Column qualifiers are mutable and different rows can have different sets of column qualifiers.
- **Cell:** A cell is the basic quantum of information in HBase. It is addressed by a row key, column family and column qualifier and contains a value and a timestamp, which represents the value's version. Each cell is associated with a timestamp, allowing to manage different versions of data for the same row key.

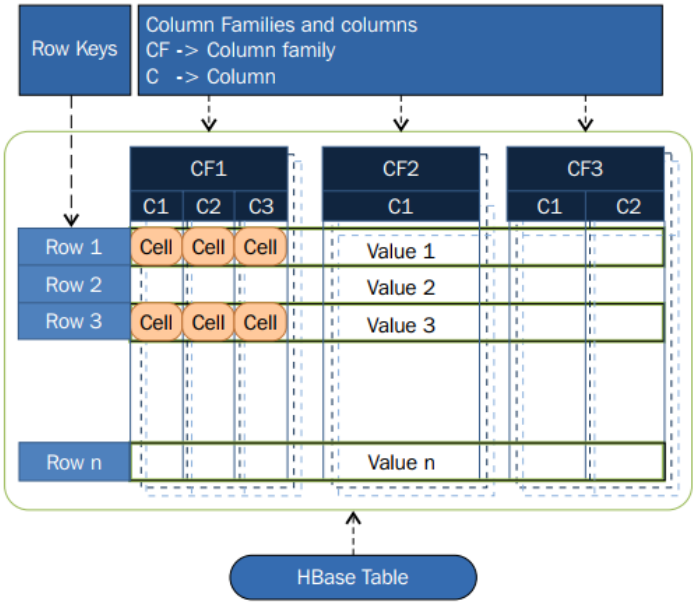


Figure 4: Logical representation of HBase data model. An HBase table consists in a set of rows where each row comprises a row key and one or more column families. Each column family groups one or more column qualifiers (simply referred as columns). The cell represents the basic unit of information, as it is addressed by a row key, a column family and a column qualifier [67].

All table accesses are performed using row keys [60]. Rows are sorted alphabetically by row keys, so lexicographically adjacent keys will be stored in the same area of physical storage [72]. For this reason, the design of the row key is crucial [15]. HBase stores everything as uninterpreted bytes. This means that any arbitrary array of bytes (not necessarily human-readable) can be used as a name for a column family or even a key for a row [50]. Both column families and qualifiers may be arbitrary strings, but column families are usually defined statically at the table schema creation, while column qualifiers can be added at any time [60, 72].

Architecture

Column family's data are stored in multiple files and in multiple regions, where each region comprises a particular range of a table's rows [61]. In order to manage them, HBase's master server assigns multiple regions to a region server. In addition, HBase uses Zookeeper to manage the coordination and resources needed to be highly available in a distributed environment [67]. In sum, HBase system architecture consists in a Zookeeper cluster, a master server and region servers. [71, 73].

The master server, designated as *HMaster*, is the administrator of the HBase cluster. It is responsible for cluster monitoring and management, assigning regions to the regions servers and handling failover and load balancing by re-assigning those regions [67, 74].

The region servers run on *DataNodes* of the HDFS and are responsible for managing the regions, splitting data in regions and coordinating and serving read and write requests [67, 74]. Regions have a previously configured maximum size. When the data grows more than that size, region servers split that region in two equal regions, maintaining the high velocity of read and write operations over data [67].

Apache Zookeeper is a distributed, open-source coordination service for distributed applications that is also part of the Apache Software Foundation [58]. It offers filesystem-like access with directories and files (called *znodes*) that distributed systems can use to negotiate ownerships, register servers and other coordination services [73]. The *znodes* are not designed for general data storage but to map the metadata used for coordination purposes. However, Zookeeper allows clients to store some useful information for metadata or configuration in distributed systems [75]. Every region server creates its own *znode*, which the *HMaster* uses to discover available servers [73]. At the creation time, a session with an associated timeout is initialized. If Zookeeper does not receive anything from its session for more than that timeout, the session is closed and the client is considered faulty [75]. With this mechanism, HBase uses Zookeeper to monitor all region servers and recover them in case of failure [67].

As for concurrency control, HBase implements Multi-Version Concurrency Control (MVCC). In MVCC, when data is updated, it does not overwrite the old data, but instead it adds a

new version for the update and marks it as the current. With this approach, a read operation can retrieve not only the current version data but also data from previous versions [38]. The maximum number of versions allowed can be configured in column families' properties.

HBase provides strong consistency for both read and write, which means that when write requests are confirmed, the same data is visible to all subsequent read requests [38]. For reads, the clients cannot read any record that is inconsistent, until that inconsistency is fixed. As for writes, HBase does not write updates to disk instantly [76]. It first saves all the updates in a Write-Ahead-Log (WAL) stored in hard drive and then it performs in-memory data replication across the nodes, increasing this way the write throughput. Periodically it flushes the in-memory structure to disk creating an immutable index-organized data file, designated *HFile* [69, 76]. For this reason, WAL proves to be extremely important to recover data in case of any failure [67].

2.3.3 *Apache Cassandra*

Cassandra was integrated in Apache Software Foundation projects in March 2010, after being open-sourced by Facebook in July 2008 [77, 78]. Since its early deployments involved storing social network data such as user activity updates, reviews and application statistics, it is optimized to provide high performance writes [78]. Cassandra is defined as a distributed, scalable, highly available, fault tolerant column-oriented database that is influenced by Amazon's Dynamo infrastructure and Google's BigTable data model [77, 78].

Data Model

In Cassandra, a database schema is represented by a keyspace, which is the namespace that contains all data objects, such as tables [79]. A keyspace is defined by a name and a set of attributes, like replication factor, replica placement strategy and column families [78].

Cassandra tables are represented by a distributed multidimensional map indexed by a key. It contains the following dimensions [78, 80, 81]:

- **Row:** Each row is identified by a string key of arbitrary length and contains one or more column families.
- **Column family:** A column family consists of columns and super columns. Cassandra stores column families in separate files on disk, so it is important to define related columns in the same column family.
- **Column:** A column is the most basic unit of data structure and it consists in a tuple containing a name, a value and a timestamp. Cassandra uses the column timestamp

to determine the most recent data. It is also possible to have different number of columns for different rows.

- **Super column:** Super columns provide another level of nesting to the regular column family structure. Each super column consists in a name and a map of sub-columns.

Cassandra is considered to be schema-free. Despite the keyspace and the column families being fixed at the database creation, the columns can be added to a family at any time [78, 81]. Similarly to HBase, Cassandra represents its data structures in sparse multidimensional hashtables, meaning that different keys can have different numbers of columns [78].

In order to perform queries over the tables, Cassandra makes use of the Cassandra Query Language (CQL), which has a SQL-like syntax [79].

Architecture

Cassandra has a peer-to-peer distributed architecture [81]. Contrary to the master/slave architecture of HBase, it is a decentralized system, meaning that all nodes of the cluster are structurally identical and function the same way [78]. Thus, Cassandra has no single point of failure (such as the master node failure), which contributes to the high availability of the system [81].

Since there is no master node, Cassandra uses a gossip protocol so that each node can have state information about the other nodes in the cluster [78]. Gossip is a peer-to-peer communication protocol where nodes exchange messages about their state and the state of other nodes they know about. This way, each node quickly obtains information about the state of all other nodes in the cluster. Every gossip message has a version associated with it, allowing only the old state information to be updated [81]. This process runs every second and it is also responsible for detecting failures. Cassandra uses an accrual detection mechanism to calculate a per-node threshold based on network conditions, workload and other conditions that might affect the node's response [78, 81]. Through the gossip mechanism, this method can determine if one node is up or down, allowing Cassandra to avoid routing client requests to unresponsive nodes [80, 81]. When a node failure occurs, the other nodes will keep trying to gossip with it to see if it is recovered, unless the administrator explicitly removes the failing node [81].

Similarly to HBase, Cassandra provides a replication mechanism to ensure reliability and fault-tolerance [81]. The number of times data is replicated across the cluster is defined by the replication factor [80]. The way that those replicas are distributed across the cluster is determined by the replica placement strategy. Both replication factor and replica placement strategy are set at the keyspace definition [81].

Despite being focused on availability, it does not mean that consistency is dismissed. In fact, Cassandra has a "tuneable consistency", which means it is possible to decide the

level of consistency required, in balance with the level of availability [78]. The consistency level can be set by the client and specifies how many replicas in the cluster must handle the requested operation in order to be considered successful [78]. Three well known consistency levels in Cassandra are [76, 78, 81]:

- ONE: This is the default value. It only needs a response from one replica for both read and write requests. For read, the response may not always contain the most recent data. A background thread is created to check the same data on the other replicas, and if there is any outdated, a read repair is performed. As for write, the operation is considered successful when at least one replica is written.
- ALL: This level perform writes and reads on all replicas and fail if there is any unresponsive replica. It provides stronger consistency at the cost of availability, due to synchronous blocking operations that wait for all nodes to be successfully updated.
- QUORUM: For write operations, it must be performed successfully on more than a half of the replicas. For read operations, it must return data with the most recent timestamp after more than a half replicas have responded.

Cassandra architecture allows any node in the cluster to be read or written [81]. When data is written to the cluster, it will be first written to a commit log, which provides durability of data in case of some unexpected shutdown [16]. A write is not considered successful until it is written in that commit log [78]. Afterwards, the data is written to an in-memory structure called *memtable*. When the memory usage of *memtables* exceeds a configured threshold or the commit log approaches its maximum size, the contents of the *memtable* are flushed onto disk in an immutable file called *SSTable* [16, 78]. As for reads, Cassandra will check the requested data in the *memtables* first [78]. All this process is presented in Figure 5.

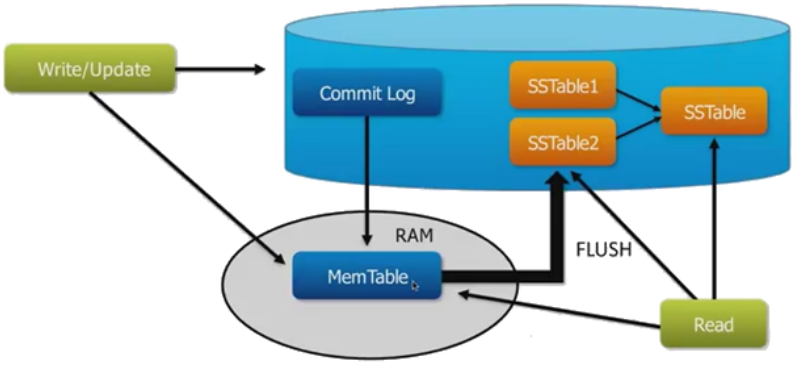


Figure 5: Cassandra workflow for read and write operations. Data is written first in the commit log to ensure durability, and then in the *memtable*. When the *memtable* reaches its maximum capacity, data is flushed into *SSTables*. As for read operations, data is sought first in the *memtables* and then in the *SSTables* [82].

2.3.4 *Apache Spark*

Apache Spark can be defined as a "fast and general-purpose cluster computing system" [18]. It provides high-level Application Programming Interface (API) in several programming languages, such as Java, Scala, Python and R [18, 53]. Spark is by far one of the most widely used open source processing frameworks for big data [19, 83].

This computing system has a batch processing model similar to MapReduce, which consists in dividing a large dataset into several small sets for subsequent parallel processing [84]. However, Spark engine supports in-memory computing, which provides faster data processing compared to traditional disk-based engines [53]. In fact, this in-memory processing capability proved to be much faster than Hadoop's MapReduce that was mentioned in Section 2.3.1 [53, 84]. Thus, this framework is commonly designed for scalable MapReduce computing across distributed systems [85].

Apache Spark also provides a set of high-level tools (Figure 6) such as Spark SQL for SQL and structured data processing, Spark Streaming for stream processing, MLlib for machine learning and GraphX for graph processing [18, 53].

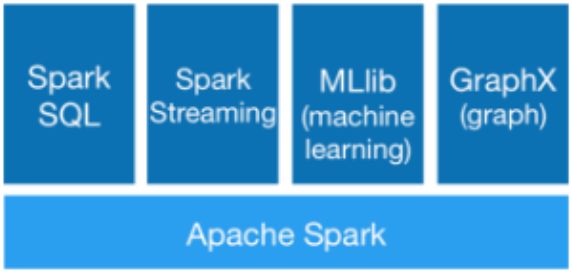


Figure 6: Apache Spark supported tools include SparkSQL, Spark Streaming, MLib and GraphX. The documentation for each element can be found in its website [18].

The Spark framework consists in a spark master node and one or more workers. The master is responsible for initializing the spark driver that manages all the workers, which, on its turn, are responsible for the execution of parallel processing operations [85]. The main abstraction that Spark provides is a Resilient Distributed Dataset (RDD), which is a collection of objects partitioned across the cluster that can be manipulated and processed in parallel [18, 86]. RDDs are fault tolerant and able to automatically recover from failures [86].

2.4 INTERNET OF THINGS

2.4.1 *Overview*

The first steps of the Internet revolution led to the interconnection between people all over the world. Nowadays, the next revolution aims to create smart environments by establishing network connections between everyday objects. Mark Weiser, who is considered to be the father of ubiquitous computing, defined a smart environment as "the physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network" [87]. To bring this concept to reality, a new technology has been developed over the years: the Internet of Things (IoT) [87].

IoT can be simply defined as the integration of all networked devices and sensors to provide information in real-time and allow interaction with people who use them [4]. It is based on the integration of Information Technology (IT), which is related to the storage and processing of data, and Communication Technology, which includes communication protocols and technologies [6]. The term "Internet of Things" was used for the first time by Kevin Ashton in the supply chain management context, in 1999 [87, 88]. Nowadays, with the present capability of Internet, IoT aims to create a new paradigm where computer systems interact with sensors and devices that see, hear and sense the surrounding environment, without the intervention of humans, in order to make real use of that information [88].

The IoT concept is based on "things", also designated as smart objects, that are built and developed in order to be identifiable, communicate and interact with every entity of the system [89]. In fact, they are considered the active participants of the system, since they are responsible to communicate with each other and with the surrounding environment by sharing data and reacting to the physical world events with or without user intervention [87]. Hence, IoT entities can be seen as providers and/or consumers of data related to the real world [89]. They can be grouped in three main classes: devices attached to objects for identification purposes; sensors and actuators for external access and control of objects' properties and functions; and finally, sensor-enabled devices such as wearables and smartphones. IoT platforms should support these different classes in order to assure the interoperability of the system, integrating every device or sensor in their network infrastructures [1].

One of the major breakthroughs concerning embedded communication in the IoT systems was the Radio Frequency Identification (RFID) technology [87]. This concept is based on microchips attached to antennas for wireless communication of data that allows automatic identification of the object they are attached to, similar to electronic barcodes [87, 90].

In its first steps, IoT only considered objects tagged with this technology. Nowadays, it covers many other technologies such as distributed sensor networks, sensor-enabled devices and other smart objects capable to interact with physical world [1]. Communication technologies are generally divided in long-range and short-range technologies. The first group refers mainly to the regular long distance communication solutions like Internet and mobile phones while the second group includes wireless technologies such as Bluetooth, RFID, Wi-Fi, Infrared and ZigBee [7, 21]. The comparison of these different short distance communication techniques is presented in [7]. Another technology for short-range communication that is worth mentioning in the IoT context is the Wireless Sensor Network (WSN) [87]. The WSNs consist of a large number of efficient, low cost and low power sensors that enables the collection, processing and analysis of valuable information gathered in several environments [87, 88]. Usually there is one or more base stations, designated by gateways, that act like a data sink to the network sensors and connect them to the physical world [88]. Sensor networks are useful specially for monitoring the status of things, as they provide a better awareness of the surrounding environment [90]. At last, an ongoing trend is to adopt IP-based sensor networks, using IPv6-based Low-power Wireless Personal Area Network (6LoWPAN) [21].

In sum, sensors and devices are responsible to interface with the physical world, either by sensing data or triggering actions on the physical world [89]. An IBM report estimated that by 2020 there will be 212 billion sensor-enabled objects available and 30 billion of them will be connected to networks [1]. According to Cisco, that number is even greater, having estimated 50 billion of network-connected devices by 2020 [6]. By making all these devices and sensors communicate and sharing information with each other, IoT platforms make it possible to collect and analyze new data streams faster and more accurately [91]. In this point of view, it is generated a massive amount of data, usually semi-structured or unstructured, that is useful only when it is analyzed, for example with big data analytic tools. In fact, IoT and big data are two inter-dependent technologies. The widespread deployment of IoT leads to a fast increasing in volume and diversity of data, thereby providing great opportunities for the development of big data applications [28].

Similarly to big data applications, one of the most critical challenges in the widespread adoption of IoT applications is the security issue. IoT solutions should provide mechanisms of confidentiality, privacy and authenticity in order to be adopted by stakeholders. Data represents one of the fundamental features in these systems, so data confidentiality is needed to guarantee that only authorized entities are able to access and modify them [20, 89]. This is achieved mainly by implementing an access control mechanism, such as Role-Based Access Control, and an authentication process (with an identity management system) [89]. Furthermore, not all data should be available. Privacy rules aim to define which data referring to individual users can be accessed. This is important especially in

healthcare applications where the privacy of personal and sensitive information must be ensured [89]. Finally, data encryption plays a key role in the information security, especially in wireless communications. It is also important to highlight that not every device is powerful enough to support robust encryption, so security solutions with low-computational complexity are required [20, 88].

Nowadays, software and hardware for sensing, communication and decision making solutions have become more versatile and affordable, which promotes the development of IoT applications in several areas [7]. The potentialities of IoT applications extend to every aspect of daily life. With networked devices and sensors equipped in the environment it is possible to improve the quality of life at home, at work, when sick, when doing physical activity or even while traveling [90]. In a more general way, the adoption of IoT solutions can provide a competitive advantage over the current solutions in several fields, such as smart homes, smart cities, environmental monitoring, smart business and inventory management, security and surveillance and healthcare. Some application examples and its benefits in those sectors are described in [89].

2.4.2 *IoT in Healthcare*

It is a fact that medical costs can affect people's quality of life, especially in the case of chronic diseases. On the other hand, the number of elder people is continuously increasing, which, apart from the costs, is putting pressure on social and health services. In addition, most patients that are subject to continuous monitoring prefer the comfort of their home instead of being confined in healthcare facilities. The development of IoT applications in the healthcare sector is one of the most promising technologies to provide a solution for those issues [6].

An IoT-based healthcare system can be defined as a network of all the available resources connected over the Internet to perform or help in healthcare activities, such as diagnosing, monitoring or even surgeries [7]. It can cover several fields like pediatric and elderly care, supervision of chronic diseases and management of private health and fitness [21]. The adoption of these solutions is seen with good eyes since it aims to improve the access and quality of care as well as reduce healthcare costs [21, 91]. In the same way as big data technologies in this sector, IoT follows the basic principle of "the right care for the right person at the right time" and, thus, it improves patient satisfaction and makes healthcare more cost-effective [91].

The sensing technologies are crucial in this field, since the acquisition of physiological parameters should be as accurate as possible in order to provide better diagnosis and treatments [7]. In this perspective, medical devices and sensors for diagnosis and imaging can be adapted as smart objects in the IoT context [21]. The progress on these technologies

enables continuous data acquisition which provides a better awareness on the diagnosis or the treatment outcomes. Some device examples for data acquisition are presented in [7].

IoT-based healthcare applications generate huge amounts of health data from several medical devices, thus providing a great opportunity for big data analyzing and processing tools in order to increase the efficiency of relevant health diagnosis and monitoring services [21]. By operating on data collected from IoT devices, analytic tools enable a better understanding on diseases and treatments as well as manage the population health [92]. Furthermore, it would be possible to spread information with medical practitioners around the world to discuss critical health cases and decide the best treatment in real-time [88].

In sum, healthcare presents one of the most potential and attractive areas for the implementation of IoT solutions, specially in remote health monitoring, chronic diseases and elderly care [21]. In a general point of view, the focus of IoT technologies in healthcare is based on the prevention, early pathology detection and home care [91]. In addition to the possibility of applying big data analytics on the huge amount of data generated by these systems, IoT applications in healthcare are expected to reduce medical costs, provide specialized treatments and increase the users' quality of life [21].

A list of healthcare applications and their required technologies and sensors in an IoT environment is presented in [21]. Some of the main healthcare services and applications are described below.

2.4.3 *Ambient Assisted Living (AAL)*

IoT systems are especially useful in assisted living scenarios. The patient resorts to sensors and devices for health monitoring to collect and made available useful information for doctors or other members of the family, in order to improve treatment and responsiveness [20]. An Ambient Assisted Living (AAL) can be implemented through an IoT platform empowered by artificial intelligence and addressed to aging and incapacitated individuals. An AAL is designed to provide better quality of life and access to healthcare services for those patients in order to extend their independent life in the comfort of their homes [21]. By allowing patients to carry medical sensors to monitor different parameters such as body temperature, blood pressure and breathing activity, it enables to perform personalized healthcare, by tracking users daily activities and making suggestions to enhance their lifestyle and prevent health problems [89].

There are several studies discussing and proposing AAL solutions based on IoT applications [21]. For example, the smart medicine box offers a good solution for patients with non-compliance problems, since it enables to track the number of pills and schedule of patient's medication and act as a reminder [23].

Remote Patient Monitoring

In healthcare, medical sensors are placed on a patient in order to monitor several parameters such as blood pressure and heart beat. With IoT systems, these sensors will be part of a sensor network, enabling the remote monitoring of the patient [88]. On one hand, in an hospital environment, beds are connected to analytical dashboards that can send vital signs of the user in real time. On the other hand, it is also possible to monitor patients remotely in the comfort of their homes, allowing the supervision and communication between healthcare professionals and patients when there is some change in the patient's condition or, simply, when they have a medical question [23]. Both applications are shown in Figure 7.

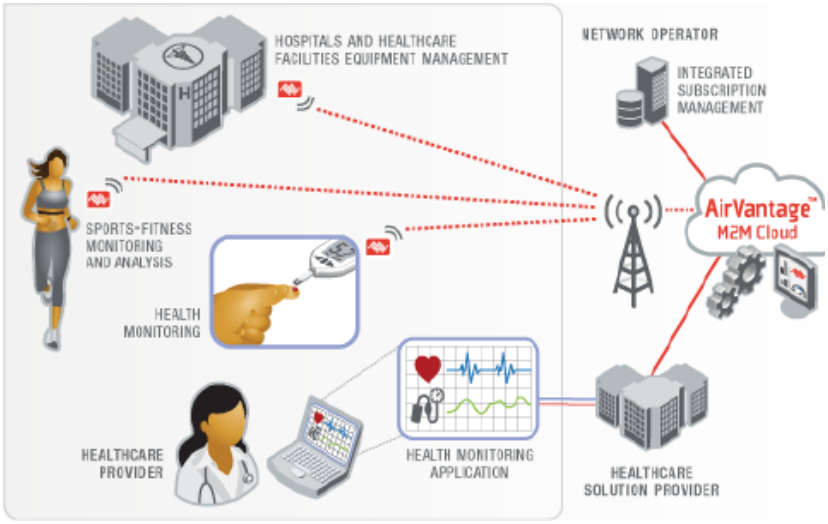


Figure 7: Remote patient monitoring schematic showing the possible scenarios of application. The healthcare solution provider collects health data from both healthcare facilities and patients being monitored outside them in order to make it available for the healthcare providers in real time. Adapted from [6].

In the first scenario, IoT systems are useful for hospitalized patients that require close observation and constant attention [91]. That is the case of patients in the Intensive Care Unit (ICU) who may have serious injuries or suffered a complex surgery. Generally, patients have different sensors and devices attached to their body to monitor vital parameters and display them on a bedside monitor. Applying the IoT concept on this scenario, it is possible to store all the health records and measurements into a database where it can be accessed by healthcare professionals through a user-friendly interface. Furthermore, an health monitoring application could process and analyze data from the ICU devices and inform in (almost) real-time the healthcare professionals about vital parameters changes and the need of any intervention [6]. An architecture for an IoT monitoring system in the ICU is proposed in [6].

In the second scenario, it is included the monitoring of elder people, rehabilitation patients and chronic diseases. The need of providing healthcare services at home tends to

increase due to the aging of the population and the prevalence of chronic diseases. Since employing more medical personnel would result in a huge increase of medical costs, remote monitoring technologies present an attractive solution [6]. Chronic diseases need to be monitored and treated early in order to prevent serious complications [4]. Patients with chronic diseases such as chronic obstructive pulmonary disease and heart failure represent a significant expenditure on healthcare services, like hospitalization and re-admissions. In [93] it is presented a study on the remote monitoring of patients with those chronic diseases that shows the potential of this technology on decreasing significantly patient re-admission and even death. Another patient remote monitoring system based on IoT to support chronic disease patients is proposed in [4].

With the technological advances, patients dispose of a great variety of sensors to track their vital signs, such as heart rate monitors, blood pressure cuffs and glucometers [4]. Some IoT-based applications in the context of remote patient monitoring include [21]:

- **Glucose level sensing:** By monitoring blood glucose it is possible to detect changing patterns that helps in the planning of meals, activities and medication times.
- **Electrocardiogram monitoring:** Monitoring the electrocardiogram provides useful information for several diagnosis such as multifaceted arrhythmias, myocardial ischemia and prolonged QT intervals.
- **Blood pressure monitoring:** It allows to keep blood pressure regularly controlled remotely.
- **Body temperature monitoring:** Body temperature is a vital sign in the maintenance of homeostasis.
- **Oxygen saturation monitoring:** Pulse oximeter wearables are suitable for non-invasive constant monitoring of blood oxygen saturation.

Data analysis and visualization are also considered critical components in remote patient monitoring systems [2]. Applying pattern recognition and analytics tools in real time across a large set of data enables to predict some anomalies such as heart strokes in cardiovascular patients [5]. Thus, it is possible to provide timely and personalized treatment. Another benefit of remote monitoring technologies is the possibility of recording a complete patient's medical history [88].

An architecture for remote health monitoring systems based on wearable sensors and partitioned into data acquisition, data transmission and cloud processing layers is described in [2]. In [5] it is proposed a remote patient monitoring solution based on web services and cloud computing, using a microcontroller board and electrocardiogram electrodes.

M-health

Due to advances in mobile communications, the use of smart mobile devices that support 3G and 4G mobile networks for data transport and mobile computing has become a source of great opportunities for every sector. The application of those mobile technologies in the healthcare sector is known as mobile-health, or in its abbreviated form, m-health, reinforcing the concept of delivering healthcare services anytime and anywhere. This way, both physicians and patients can access the same medical records through personal computers, tablets or smartphones and establish contact in case of any emergency. The term "m-health" was defined in 2003 as an interconnection between mobile communication and network technologies for healthcare purposes. Its applications can have a strong impact in healthcare monitoring and alerting systems, clinical and administrative data collection, healthcare delivery programs and detection and prevention systems [94].

Recently, the massive development of smartphones and tablets along with smartphone-controlled sensors contributed to the widespread of m-health applications, representing a big step towards a person-centered approach in healthcare sector [95]. Many smartphone applications have been developed for several purposes, such as diagnostic, which are used to access diagnostic and treatment information; drug reference, providing information about the name of the drugs, as well as their indications, dosages and side effects; and medical education applications, with tutorials and surgical demonstrations [21]. A list of various smartphone healthcare applications is presented in [21] and [94].

In addition, the wearable devices that have been recently developed allow to extend the capabilities of mobile devices, especially in remote monitoring of human body parameters [95]. Another improvement on this field is the suitability of many health and fitness accessories with smartphones, such as Apple watches [96], Fitbit Flex [97, 98] and Garmin Vivosmart [99] bands, allowing to track heart rate, steps, distance, calories burned and sleep [21, 23, 92].

An IoT architecture based on wearable devices is described in [95]. In [100] it is proposed a smartphone-centric platform for remote monitoring of patients with heart failure. The potential and implementation of an IoT architecture based on the m-health concept for non-invasive sensing of glucose level is described on [101].

Security Requirements

The security requirements concerning the IoT-based healthcare systems are identical to the standard IoT systems. Confidentiality needs to be ensured in order to make data accessible only to authorized users, requiring authentication mechanisms to ensure their identity [7, 21]. Another important requirement is data integrity, to ensure that medical data is not altered or compromised. Finally, the system should also guarantee two important properties: availability and fault tolerance. The first one makes sure that IoT healthcare services

are always available to authorized parties, even under denial-of-service attacks, while the second one ensures that in the presence of a system fault, the same services are still being provided [21].

2.4.4 *IoT Platforms*

The evolution of IoT systems along with the expansion of network-connected sensors and devices resulted in the generation of enormous amounts of data and, consequently, the need for huge data storage and computing power [1, 87]. Therefore, IoT cloud based platforms integrated with cloud based storage solutions have been developed in order to address those issues [1]. An IoT platform should be able to collect data from devices and send it to cloud servers for processing and storage for future utilization. This way, an IoT system must be equipped with communication services to interact with those devices and provide data management and application development services [102].

There are numerous commercial IoT platforms available, where four of the big players include IBM Watson IoT Platform [103], Amazon Web Services IoT Platform [104], Cisco's IoT cloud connected [105] and Microsoft Azure IoT Platform [106, 107]. In terms of open source platforms, there are some attractive solutions such as Kaa [22], DeviceHive [108], OpenIoT [109] and ThingSpeak [106, 110, 111]. In [110] it is presented a comparison between different IoT platforms both paid and open source, including Kaa, in terms of their basic features. In a general way, paid platforms offer more services than open source platforms, especially regarding the management of events from things, analytics and security. However, despite being open source, Kaa platform supports many of those features and functionalities, while offering a large variety of application and communication protocols, highlighting Kaa as a promising open source IoT platform.

A middleware is considered a key technology in the development of IoT systems since it behaves as an intermediary between IoT devices and applications [112]. It can be defined as a software layer between the technological and the application levels. By abstracting the technological details and complexities of the system, it allows the developer to focus directly on the development of new applications without having to write a different code for each device [20, 90, 112]. Kaa is included in this category. In [112] it is presented a comprehensive review of several middleware systems and tools for IoT.

2.5 KAA IOT PLATFORM

2.5.1 *Overview*

Kaa is an open source middleware platform for IoT that enables building complete end-to-end IoT solutions and applications, as schematized in Figure 8 [22]. It is supported by CyberVision [113] and licensed under Apache 2.0 [114], which is a free license that provides full ownership over developed solutions [22, 112]. The Kaa platform offers an open and feature-rich toolkit for development of IoT applications, that consequently allows to reduce associated costs, risks and time-to-market [22, 115]. Some of its capabilities include [22]:

- Managing an unlimited number of sensors.
- Setting up interoperability across devices.
- Provisioning and configuration of remote devices.
- Performing real time monitoring.
- Distributing over-the-air firmware updates.
- Creating cloud services for smart products and applications.
- Collecting and analyzing data from devices and sensors.

In order to start using this features, Kaa offers a preconfigured virtual environment, known as Kaa Sandbox, to install in a VirtualBox Virtual Machine for small developments or proof of concepts. Alternatively, it is also possible to deploy Kaa in a cluster environment [22, 115].

Regarding Figure 8, Kaa provides Software Development Kit (SDK) components to the objects in the hardware layer in order to establish connection with the back-end infrastructure. Kaa SDKs can be integrated with virtually any type of device, allowing the exchange of data between the connected device and the server. The Kaa server is responsible for back-end functionality, such as communication between connected objects, data consistency and security, device interoperability and failure-proof connectivity. Finally, the Kaa server also provides interfaces for the integration with data management and analytic systems or other applications, as represented on top of Figure 8 [22].

Figure 8: Overview of the Kaa platform as a middleware between the hardware devices and the application layer. The devices are provided with Kaa SDKs to securely communicate with the server, which integrates several applications, such as data management and analytic systems [22].

Some of the key features that highlight the potential of Kaa are described below.

Hardware and Data Model

Kaa platform is hardware-agnostic, meaning it can integrate any device, from functional operating systems to simple microcontrollers. As for data models, both structured and unstructured data are supported, however it favors the first one. Well-structured data not only represent good material for analytics, but also help in the abstraction of any low-level implementation on the hardware. In addiction, Kaa can manage and operate identically different technologies and devices as long as they have the same data schema. Schemas are Apache Avro [116] compatible, which enables efficient data serialization [22].

Endpoint SDK

An endpoint SDK is a library that provides client-side APIs for implementing Kaa platform features. SDKs are designed to be embedded in the client application and to handle

several features, such as client-server communication, authentication, data marshalling, encryption and persistence. They are available in Java, C++ and C languages. Each SDK is generated for a specific server instance and contains the data schema for the IoT application [22].

Connectivity and IoT Gateways

Kaa platform is also transport-agnostic, meaning it allows the communication over any type of network connection. IoT gateways and routers can improve the performance of Kaa applications, since they simplify data collection from sensors and devices that use simple communication protocols, such as Bluetooth and ZigBee [22].

Data Processing

Kaa server provides APIs to instruct the transmission of data collected from sensor devices to the back-end data processing and/or warehousing system. Additionally, it supports a framework of pluggable log appenders that enables to load data into a specific database, stream processing or even a custom data processing module via REST or Flume. Therefore, Kaa allows the seamlessly integration of data processing and analytics tools, ensuring an end-to-end structured data flow [22].

2.5.2 *Architecture*

Kaa platform comprises a Kaa server, Kaa extensions and Kaa endpoints. The Kaa server represents the back-end layer of the platform and it is responsible for managing users, applications and devices. Kaa extensions include third-party software that improves the platform functionality. Finally, a Kaa endpoint is a registered client application that resides on a particular connected device embedded with a Kaa SDK [22].

An high-level perspective of Kaa architecture is presented in Figure 9. A Kaa cluster consists in several interconnected Kaa server nodes, which are coordinated with the help of Apache Zookeeper services. The cluster requires NoSQL and SQL databases to store endpoint data and metadata, respectively. Each Kaa node in the cluster performs a combination of Control, Operations and Bootstrap services [22, 115].

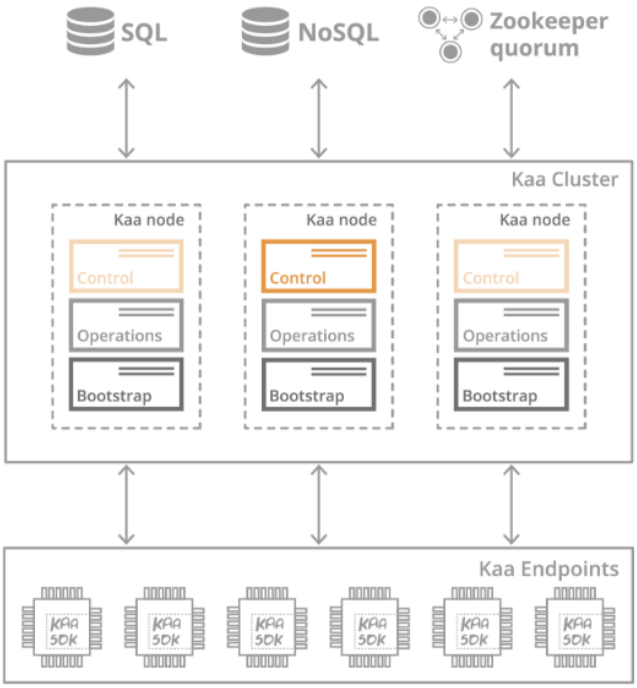


Figure 9: High-level overview of Kaa architecture. Kaa cluster is comprised of several Kaa nodes coordinated by a Zookeeper quorum and requires SQL and NoSQL databases to store endpoint data and metadata, respectively. Endpoints communicate with the cluster through the Bootstrap services of the Kaa nodes while the Control service manages the available Operations services [22].

The Kaa Control service is responsible for managing the entire system and processing API calls from both web User Interface (UI) and external integrated systems. In order to maintain an updated list of all available Operations services, Zookeeper sends continuously information to the Control service. Additionally, it provides an web UI that enables the users to perform administrative operations, such as creating applications, registering and configuring endpoints and managing accounts. To guarantee the high availability of the system, there should be at least two nodes with Control service enabled, one active and the other in standby mode. Thereby, in the presence of a failure, Zookeeper notifies the standby service and promotes it to the active Control service [22, 115].

The Kaa Operations service is responsible for concurrently handling the endpoints requests. Thus, the cluster should have this service enabled in every node in order to provide horizontal scaling to the system. In case of an outage of this service on one node, the previously connected endpoints switch automatically to another available node. In addiction, Kaa server perform re-balance routines to route endpoints to the less loaded nodes in the cluster [22, 115].

The Bootstrap service is responsible for redirecting endpoints to the Operations service, by sending the information about connection parameters [22, 115]. When the SDKs are

generated in the platform, they contain a list of the Bootstrap services available in the cluster, which is used later by the endpoints to connect to those services. On the other hand, Bootstrap services also keep an updated list of the available Operation services, by coordinating with Zookeeper. The endpoints connect to the Bootstrap services in order to retrieve the connection parameters, such as IP (Internet Protocol) address and security credentials, from the list of the available Operation services [22].

In terms of SQL databases, Kaa supports MariaDB and PostgreSQL, which are used to store metadata such as users, applications and endpoint groups. NoSQL databases provide storage for endpoint-related data that grows linearly with the number of endpoints. Kaa officially supports Apache Cassandra and MongoDB as NoSQL solutions [22].

2.5.3 *Applications*

The concept of IoT brings numerous opportunities for smart applications and technical innovations in different sectors. Several IoT use cases suitable for the deployment of Kaa platform are presented on Kaa project website, such as agriculture, automotive, consumer electronics, industry, logistics, smart city, smart energy, sports and fitness, wearables and healthcare. Regarding the healthcare industry, Kaa enables some important features such as the ability to manage virtually any medical device, remote monitoring of patient's health, remote device configuration and data analytics applications for clinicians and patients [22].

2.6 SUMMARY

The generation of data is rapidly increasing in every sector due to the advances in communication technologies, sensors, electronic data and high-speed networks, with healthcare being no exception. In the healthcare sector, a huge volume and variety of medical data comes from different sources, such as sensors and medical records. The concept of big data arises in order to explore all the potentialities of these massive amounts of data, by using analytics and processing technologies to extract useful information. The application of these technologies provides several advantages, including better diagnosis, preventive care and personalized treatment to each patient, which enables to change the healthcare paradigm to a patient-centered approach.

In order to benefit from big data solutions, several challenges concerning data management and storage need to be addressed. For that cause, many cloud storage solutions have been developed, highlighting Apache HBase and Apache Cassandra as two major databases widely adopted in big data applications. Apache foundation also addresses analytic solutions, such as Apache Spark, that can be easily integrated with the mentioned databases and also enables stream processing in real time.

IoT presents another attractive technology in the healthcare field. It is an emerging technology that enables the communication and interaction between everyday objects through a continuous network. IoT and big data are inter-dependent technologies, since IoT provides an important source of data for big data solutions to process and analyze. There are several areas of application in the healthcare sector for these systems including ambient assisted livings, remote monitoring and m-health.

Nowadays, despite having several IoT solutions on the market, building an IoT platform capable of efficiently handling all interoperability, storage, communication, security and management issues represents one of the greatest challenges. An IoT solution using the open source middleware Kaa integrated with both HBase and Cassandra databases is proposed and tested on this thesis project, focusing on data collection features, in order to reinforce the potential of IoT and cloud solutions in the healthcare environment.

3

RESEARCH METHODOLOGY

This chapter is addressed to the research methodology followed along the development of this master thesis. Section 3.1 presents a brief introduction concerning the concept of research methodologies, while Sections 3.2 and 3.3 expose the methodology selected and its practical application, respectively.

3.1 INTRODUCTION

A research methodology can be defined as a guideline that aims to find a solution for a specific problem or contribute to the understanding of a phenomenon [117]. It comprises a set of principles and practices that leads the researcher to choose a certain set of methods over another [118]. This way, some methodologies may be more efficient than others for a specific problem and, thus, it is crucial to follow a well established research methodology in order to obtain the best results.

The Design Science Research (DSR) methodology represents one of the most famous and commonly used research methodologies in the fields of engineering and science, including computer science and IT [119–121]. Furthermore, this methodology is based on the identification of a problem, the design of a solution and its validation, which fits the purpose of this master thesis. Therefore, this was the approach followed during the development of this project.

3.2 DESIGN SCIENCE RESEARCH

The DSR methodology is essentially a paradigm for problem solving. As stated in [117], "It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished".

This methodology comprises two main activities: building and evaluation. Building is related to the creative process that leads to new artefacts, while evaluation assesses their

utility [117]. According to [122], a DSR methodology approach follows the steps described bellow:

- **Awareness of the problem:** In this step the researcher becomes conscious about the problem. This awareness may have different sources, such as new developments in technology or simply reading related areas. After that, the researcher formulates a proposal to solve the problem, and thus, begins the investigation.
- **Suggestion:** During this phase, the researcher must come with at least one *tentative design*, with the goal to solve the problem from the previous step.
- **Development:** In this step the researcher presents one or more artefacts, which can be of different natures such as algorithms, software or expert systems. The methods used on its development do not necessarily need to be innovative, the innovation is in the design.
- **Evaluation:** This step comprises the assessment of the artefacts using the criteria stated on the proposal. The results must be in conformity with the expectations about the product behaviour during its design, otherwise the researcher must attempt to understand any divergence.
- **Conclusion:** Every results, either positive or less positive, must be consolidated and written down in order to apply the acquired knowledge and/or perform further research.

By following these concepts it was possible to efficiently design and organize the development of this master thesis. The practical application of the DSR methodology is described in the next section.

3.3 Practical Application

As previously stated, the research methodology adopted for the development of this master thesis was the DSR methodology. Therefore, it was required to identify and establish the different phases of the development of this project.

At the beginning, it was made a brief research about the emerging technologies and their potential applications in the healthcare sector. After some research, it was possible to identify IoT as one of the most promising technologies, however the implementation of these systems in healthcare environment is still under research due to storage, data management and security issues. After being *aware of the problem*, the next step was the *suggestion* to solve it, which introduces the proposal of two architectures for an IoT platform - Kaa, integrated with two different NoSQL databases - Cassandra and HBase. In addiction, the

integration with a framework for analytics applications, namely Spark, was also suggested. The *development* phase comprises the deployment of Kaa cluster and its integration with both databases and Spark. Kaa platform already officially supports Cassandra, however, developing the integration with HBase was required. Having both architectures fully working, it was the time to test them and *evaluate* their behaviour, comparing both results. All the results were analyzed and the *conclusions* are discussed later in this master thesis.

4

THE PROBLEM AND ITS CHALLENGES

This chapter exposes the fundamental problem addressed by this master thesis and its challenges . In Section 4.1 it is presented an introduction regarding the main issues of this project, while in Section 4.2 it is proposed an approach to answer their requirements. Finally, Section 4.3 presents the system architecture of the approach.

4.1 INTRODUCTION

As it was previously stated in Section 2.2.2, healthcare data is massively increasing not only in volume, but also in variety and in the speed at which it is generated [29]. This huge amount of data provides a great opportunity for big data tools to implement analytic applications, allowing to extract useful insights on healthcare data and bringing several benefits, especially in preventive care, better diagnosis and personalized treatments [30]. However, in order to deploy big data analytic applications with success, it is crucial to store all that data efficiently.

Several areas, inside and outside healthcare facilities, may take advantage of big data systems to improve their quality of care. By integrating these systems with IoT platforms, all the available resources will be connected with each other over the Internet to perform or help in healthcare activities, such as diagnosing and monitoring, improving the quality of care and also reducing medical costs [7]. Some of the areas outside medical facilities that benefit from these applications include the monitoring of elder people and chronic disease patients. Inside healthcare facilities, these systems might be especially useful for monitoring hospitalized patients that require close observation and constant attention, such as patients in the ICU [6]. These patients usually wear several sensors to constantly monitor vital signs like electrocardiogram, blood pressure, body temperature and oxygen saturation. Additionally, the inpatient room also has sensors to monitor the environment, such as the air quality. The capability of IoT systems embedded with big data technologies to integrate, connect all these sensors and process their data improves the quality of care, allowing to perform better diagnosis and personalized treatments for each patient. Nevertheless, the required quick storage of wide amount and variety of data and the constant demand of

data processing tasks represent some obstacles to overcome in order to achieve the success of these systems.

4.2 PROPOSED APPROACH

This master thesis focus on the scenario where data is collected and processed inside an healthcare facility, mostly from hospitalized patients. To develop an IoT system capable of handling healthcare big data, four main feature blocks were considered: Data Source, IoT Platform, Data Collection and Data Processing. Each of these blocks represent the required feature (or features) that the system has to support in order to be fully operational. In Figure 10, it is schematized the integration of these four blocks.

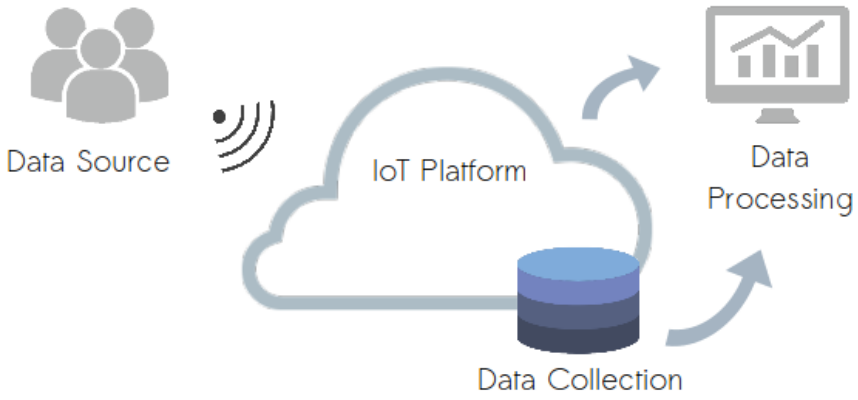


Figure 10: Proposed architecture represented by four feature blocks: Data Source, IoT Platform, Data Collection and Data Processing. Data Source represents the block responsible for sending data to the IoT Platform, which can forward that data to storage solutions in the Data Collection block or to processing tools in the Data Processing block. The Data Processing block should also be able to retrieve data from Data Collection directly.

Data Source

This block concerns the provenance of the healthcare data and its integration into the IoT platform. Data can be generated from multiple sources, such as EMRs, medical imaging and medical sensors. Then, data can be sent directly to the platform or through a gateway, where it can be pre-processed in order to apply some necessary actions or trigger any inconsistency warning. Gateways are especially used in WSN environments, as it was referred in Section 2.4. Thus, it is crucial to ensure the interoperability between the source of data and the platform, as well as the establishment of efficient communication protocols used for exchanging information. It is important to mention that security measures should also be taken into account to ensure the authenticity of data.

IoT Platform

The IoT platform is the center block of all the system, since it establishes the connection between the other three blocks and it is responsible to manage and monitor every resource and every task in the system. In terms of data flow, this block handles the authentication and communication processes of the endpoints in the Data Source block and, then, forwards the information received from them to their final destination, being either the Data Collection block or the Data Processing block. It also manages all users and applications deployed in the system.

Data Collection

The Data Collection block is responsible for storing the incoming data from the Data Source block and comprises database solutions to fulfill that purpose. Since this system is designed to deal with big data, it requires fault tolerant storage solutions capable of handling large amounts of data with different data structures (or even not structured at all). As it was presented in Section 2.2.3, NoSQL databases are distributed and horizontal scalable databases designed for large-scale data storage, representing, therefore, the best candidates for the development of this block.

Data Processing

The Data Processing block not only represents data processing mechanisms, but also data visualization and analytic applications. The term "processing" in this block is related to any consumer or end user of the platform data. On the one hand, for visualization and simple processing purposes, data generated in Data Source block can be forwarded directly through the IoT platform to the Data Processing block, in almost real-time. This mechanism is designated as data streaming. On the other hand, for analytic and machine learning applications, the Data Processing block should also be able to retrieve data directly from the Data Collection block.

4.3 System Architecture

The proposed approach was based on the four blocks detailed in the previous section. In order to achieve a concrete architecture for the proposed system, it was necessary to design and develop each of those blocks with appropriate frameworks and software and, finally, integrate them in a fully operational platform.

Since the IoT Platform is the center block of the system, it was defined as the starting point to design the system architecture. The fundamental requirements for the selection of the platform were the open source nature and the capability of handling a wide amount of

users and sensors, as well as providing data collection features. Kaa platform is not only suitable for those demands, but also provides important additional features, such as notifications, authentication and encryption mechanisms, devices management, administrative capabilities and an intuitive web UI. As it was stated in 2.4.4, Kaa is a promising middleware IoT platform in the realms of the open source software that allows the developer to focus on the development of IoT applications without having to write a different code for each device. One particularity of Kaa, is that it resorts to SDKs to handle client-server communication, authentication, data marshalling, encryption and persistence. Thus, by integrating those SDKs in the Data Source devices it is possible to obtain Kaa endpoints, which are responsible to communicate and send data to the Kaa platform.

Regarding the Data Collection block, Kaa platform is already integrated with Cassandra database to store data from Kaa endpoints. As it was stated in Section 2.2.3, Cassandra meets the requirements for big data applications. Nevertheless, HBase represents a good alternative to have into account, since it is built on top of the HDFS, which has been widely used in the analytics area.

Finally, as it was referred in Section 2.3.4, Apache Spark is one of the most widely used open source frameworks for big data processing. The possibility to implement several tools, such as Spark Streaming and MLib, makes this framework highly suitable to be deployed in the Data Processing block. Additionally, it was also important to integrate an alternative component that fetches data directly from the storage solutions, making it available for large-scale analytic or machine learning applications.

Figure 11 presents the integration of all these components, in a distributed system designed for data collection, processing and analytics, highlighting Kaa as the IoT platform, HBase and Cassandra as the storage solutions for big data and Spark as the processing framework.

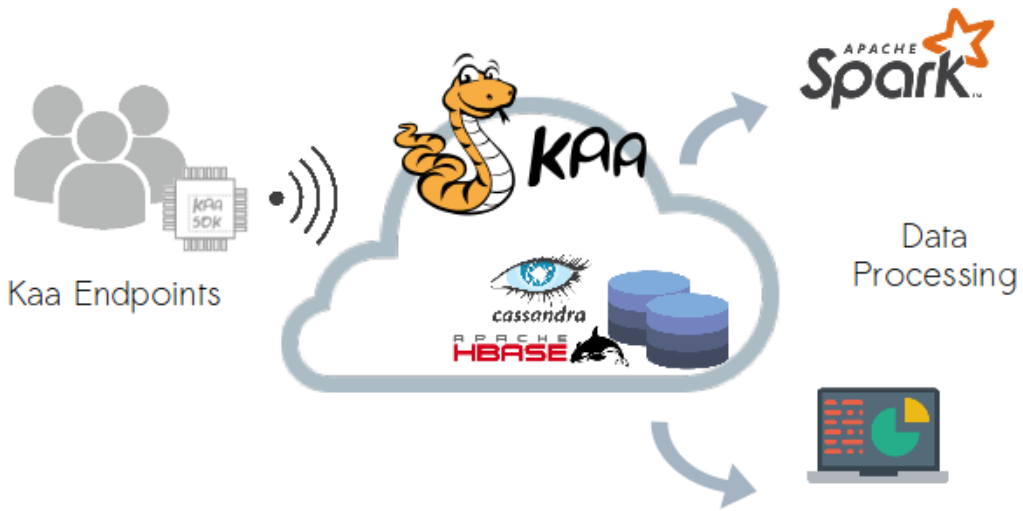


Figure 11: System architecture for data collection, processing and analytics. It comprises: Kaa endpoints as data source; Kaa platform as the IoT Platform; Cassandra and HBase as data collection solutions; Spark and an analytical application as data processing mechanisms.

5

PLATFORM DEVELOPMENT

This chapter addresses the development and implementation of the system architecture described in the previous chapter. Firstly, Section 5.1 presents the main assumptions and decisions that lead the whole development process. Section 5.2 explains in detail the implementation process of all system components, while Section 5.3 presents a brief summary focusing on the major outcomes of this chapter.

5.1 DECISIONS

The main subject of this master thesis concerns the development of an IoT platform for medical sensors with emphasis on data collection and analytic features. Thus, it is crucial to design a system capable of handling large amounts of data efficiently to achieve those purposes. The system architecture schema was already described in Section 4.3. Nevertheless, there are still important questions to be addressed.

The first issue is related to the deployment of the IoT platform. As it was referred in Section 2.5, Kaa platform can be either used in single mode, using a pre-configured environment designated by Kaa sandbox, or deployed in cluster mode. The first option is suitable for small applications and proof of concept scenarios. However, for production and real case scenarios the cluster mode represents the best option. Since this project focus on data collection applications, it was decided to particularly explore that feature among the several features provided by Kaa.

The platform was designed to integrate all data from healthcare environments. However, for the system proof of concept, it was only considered data generated from medical sensors. Thus, it was decided to develop Kaa endpoint applications that simulate medical sensors for temperature, blood pressure, electrocardiogram and oxygen saturation.

Another critical decision to address was related to the storage mechanism. As it was stated in Section 2.5, Kaa platform already officially supports Cassandra database as the NoSQL solution for data storage. The connection between Kaa IoT platform and another third party software is established due to a driver component designated as log appender. The log appender, as the name suggests, collects the logs of data sent to the platform and

appends it to a selected third party software destination. The Cassandra log appender is the component responsible to append data to the Cassandra database. However, there is no log appender for HBase (Figure 12).

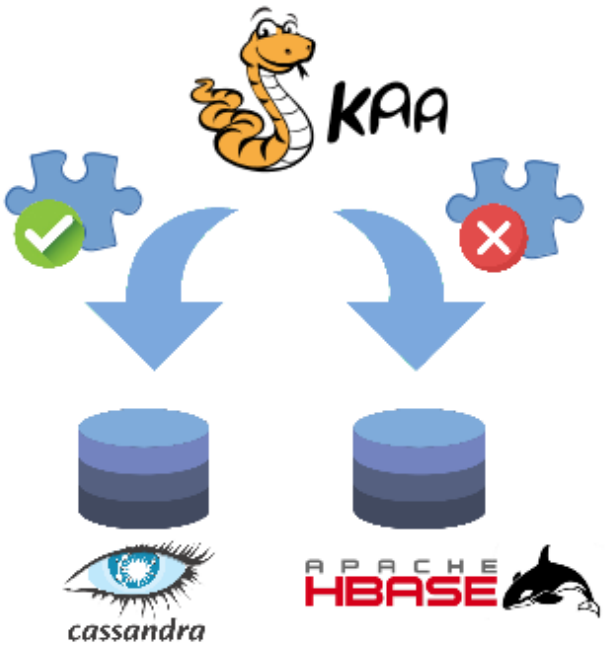


Figure 12: Data storage solutions overview in default Kaa platform. Kaa officially supports Cassandra, through Cassandra log appender, while there is no support developed for HBase.

Therefore, in order to integrate HBase in Kaa platform it was necessary to develop a new log appender. Hence, not only it provides the whole system with an alternative NoSQL database that is widely used, especially for analytic purposes, but also makes it possible to perform a quantitative comparison between the two NoSQL solutions, HBase and Cassandra.

Finally, the approach for the last block of the system, the Data Processing block, represents the last major issue on the platform development. The first decision was to develop all of its components outside the main IoT platform, in order to avoid unnecessary load into the platform. The second decision was to implement Spark and a generic component for analytical applications. Despite providing several different tools, Spark framework was specifically integrated for (almost) real-time streaming purposes. In an healthcare environment, the availability of collected data is as important as the data collection process itself. In this perspective, data streaming applications are extremely relevant since they are responsible to drive the collected data to the end user or even to an interactive dashboard. The capability of Spark Streaming library to perform data processing in (almost) real-time and display that data to the end users, makes Spark the chosen framework for the implementation of streaming applications. In order to implement more complex analytic applications, it

was also set another component in the platform that communicates directly to data storage solutions. Thus, it is possible to access all data from medical sensors and apply machine learning or other analytic algorithms, creating a good opportunity for big data scientists. It is important to notice that, for this master thesis purpose, this new component is generic and only performs reads from the storage solutions. Nevertheless, its impact on the whole platform is the same as it would be with any of those analytical applications.

5.2 IMPLEMENTATION

Based on the planning and decision making process over the platform development, it was possible to break the implementation process into four main steps: central platform development, HBase log appender development, client application development and data processing application development. The central platform development stage comprises the core of the whole system. It concerns the deployment of the IoT platform and storage databases, including all the required components and configurations. The HBase log appender development stage focus on the driver component to connect HBase data storage with Kaa platform. Finally, in the application area, the client application development step aims to develop medical sensors for data integration on the platform, while data processing application development stage considers the deployment of Spark and the generic data analytic component.

5.2.1 *Central Platform Development*

This stage represents the implementation of IoT Platform and Data Collection blocks, described in Section 4.2. Firstly, it was deployed the Kaa platform in single mode for each node, along with the required third party software, such as Zookeeper for coordination purposes; Cassandra and MariaDB databases for storing endpoint data and Kaa metadata, respectively. Secondly, all those components were configured for cluster environment. Both deployment and configuration processes were fulfilled based on the official documentation of each software component. Afterwards, it was integrated the HDFS and HBase to the whole system, also in cluster mode, using the official documentation for their deployment.

5.2.2 *HBase Log Appender Development*

This stage comprises the development of the log appender responsible for establishing connection between Kaa platform and HBase. The official Kaa platform documentation provides a guide to develop a custom log appender, which was followed in order to create the HBase log appender. The process comprehends four main steps: design a configuration

schema, implement the log appender, develop the log appender descriptor and provide the log appender. The project involving the development and code of HBase log appender is published on GitHub [123].

Design a configuration schema

The configuration schema is an Apache Avro compatible schema that defines the structure and configuration of the custom log appender. In order to create a log appender instance for HBase, it was necessary to specify the parameters required for establishing the connection with the database itself. Thus, the configuration schema for HBase log appender contains a zookeeper quorum, an HBase table, column families and the respective column qualifiers. The result of the compiled schema with those parameters in Kaa web UI is presented in Figure 13.

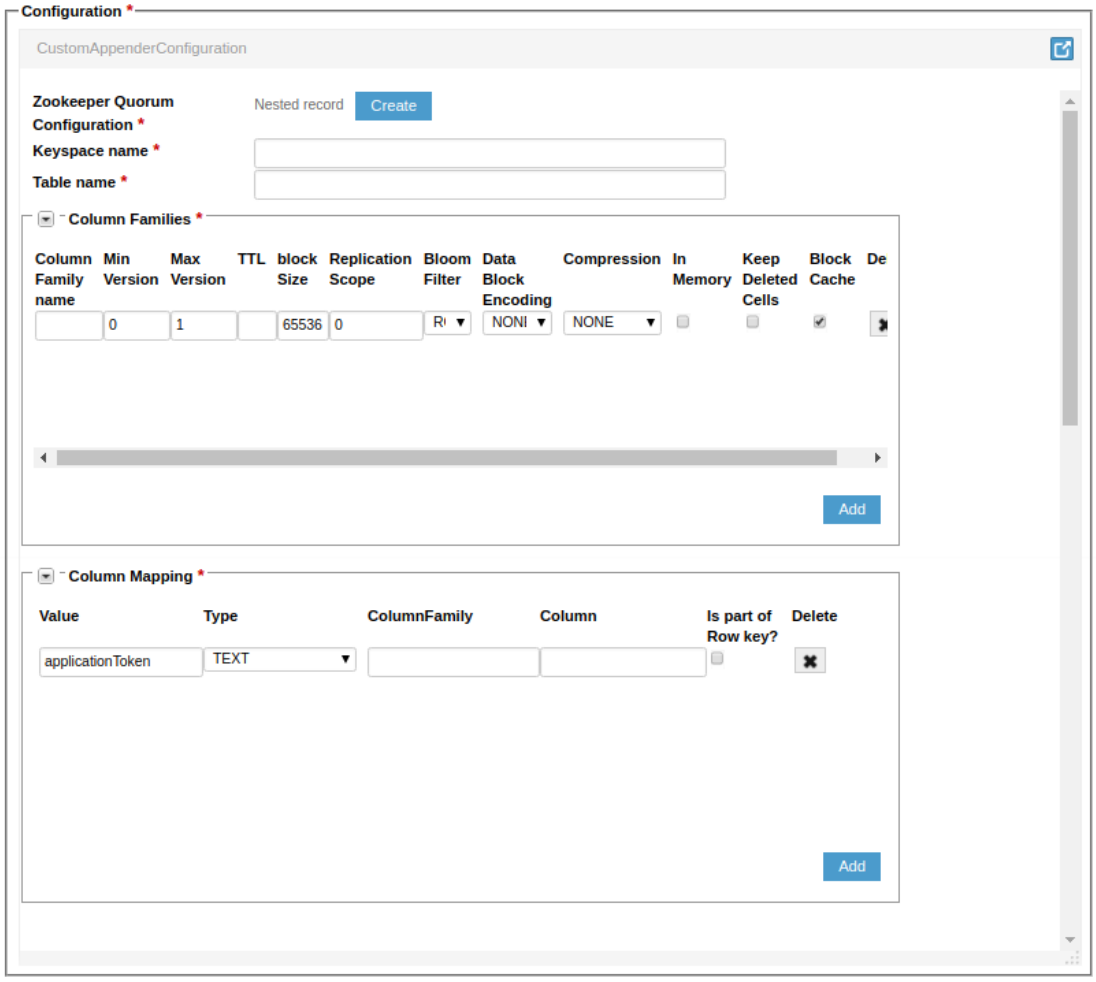


Figure 13: HBase log appender configuration schema. In order to instantiate an HBase log appender, it is necessary to specify the zookeeper quorum, the HBase table, the column families and the respective column qualifiers.

The zookeeper quorum comprises the IP addresses of all system nodes containing zookeeper, as well as the port at which they are listening. The HBase table is defined by the table name and the corresponding keyspace. The column families can be set with several properties, such as data minimum and maximum versions, time-to-live, block size, replication factor, bloom filters, data block encoding, compression algorithms, in memory column family, keep deleted cells and block cache. Finally, in the column mapping area, the value and type fields represent the source of data from the Kaa SDK integrated in medical sensors; the column family and column fields represent the HBase column where data will be stored. In this sector, it is also possible to select one or more values to be part of the row key.

Implement HBase log appender

Every log appender in Kaa platform extends a generic log appender abstract class, which implements three main methods: initialize, append and close. In order to create the HBase log appender, the development of those three methods was required.

The initialization method comprises all the instructions to start the log appender. Firstly, it establishes the connection with the HBase database through the zookeeper quorum specified in the log appender configuration. Afterwards, it connects to the HBase table, using the keyspace and table name parameters from the configuration schema. If the table does not exist, it is created along with the column families specified in the log appender configuration.

The append method is responsible for consuming logs and data from Kaa endpoints, or in this particular case, medical sensors. When data is sent to the log appender, it creates a row key based on the configuration schema and then proceeds to the storage into the HBase table. Since HBase stores data as uninterpreted bytes, data is converted before the storage process. Finally, a successful message is sent to the client application.

The close method is responsible for closing the appender and releasing any resources associated with it. Thus, both connection with HBase table and HBase cluster through zookeeper quorum are closed in this step.

Develop HBase log appender descriptor

The log appender descriptor provides the Kaa server with the information about the location and configuration of the log appender. Therefore, in order to enable Kaa to find and implement the HBase log appender, the respective descriptor needs to be developed. Some important features to configure the HBase log appender descriptor include the name presented in the web UI, the directory of the HBase log appender implementation class and the configuration schema.

Provide HBase log appender

The HBase log appender is a Maven project. In order to provide it to Kaa server, it is necessary to build the project first and, afterwards, place it in Kaa library directory. The instructions of this process are detailed on the official Kaa documentation.

5.2.3 *Client Application Development*

This stage is related to the development of Kaa endpoints to simulate medical sensors. It represents the Data Source block from the proposed approach in section 4.2.

In the Kaa environment, client applications are integrated with SDKs to handle data marshalling, encryption and communication with server. To generate a SDK in Kaa platform for data collection purposes, it is necessary to create an application and specify two different schemas: a log schema and a configuration schema. The first one concerns the data schema of the client application, while the second one is related to the configuration features, like the sample period. After defining the two schemas, it is possible to generate the corresponding SDK in several programming languages. It is important to notice that this process is explained in the official Kaa documentation. In this master thesis, Java was selected as the programming language.

According to Section 5.1, the medical devices selected for data integration into the platform include thermometers, pulse oximeters, invasive blood pressure and eletrocardiogram sensors. With a view to simulate a real clinical scenario, those medical sensors were developed based on simulated sensors from an Open Source Integrated Clinical Environment [124]. This open source project provides a framework for connecting medical devices and clinical applications through the concept of medical IoT. The adaptation of those simulated sensors with Kaa SDKs enabled their immediate integration with Kaa platform and, thereby, with the whole system.

5.2.4 *Data Processing Application Development*

This stage concerns the Data Processing block presented in Section 4.2. As it was previously stated, this block consists of two different components, Spark and a generic analytical application, deployed outside the central platform to avoid unnecessary load.

Spark was implemented in standalone mode, following the official documentation. Afterwards, it was developed a Spark streaming application in order to allow real time streaming of data from medical sensors. Kaa platform does not provide a direct integration with Spark. However, it provides a log appender for Apache Flume, which is supported by Spark as a data source for streaming purposes. After develop a Spark streaming application listening

to Flume data source and establish the Flume log appender in Kaa web UI, it is possible to achieve real time streaming of data from medical sensors in order to perform some processing or dashboard visualization. For the proof of concept, data sent from medical sensors was streamed by Spark application to an online dashboard named Freeboard [125].

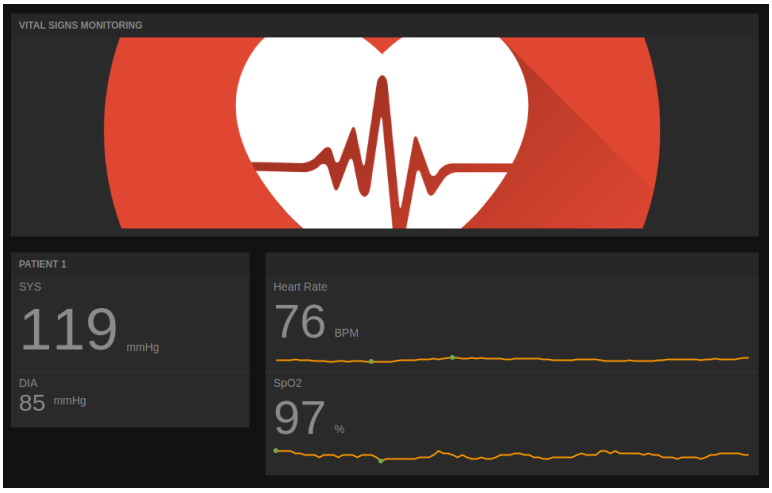


Figure 14: Vital signs dashboard showing blood pressure, heart rate and oxygen saturation values in real time. For heart rate and oxygen saturation it is also shown the history chart.

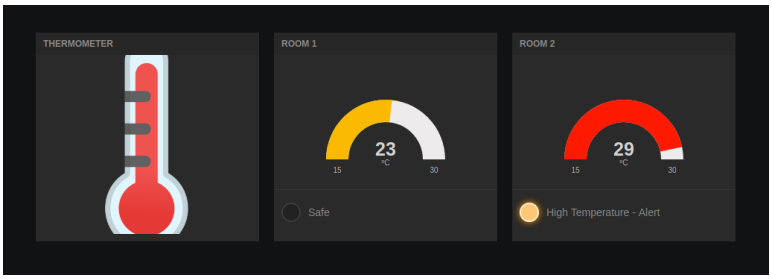


Figure 15: Thermometer dashboard showing the temperature measured in real time for two different rooms. It also shows the presence of an alarm for high temperature values.

Figure 14 presents a vital signs monitoring dashboard for a single patient, showing the blood pressure, heart rate and oxygen saturation values, in real time. It is also possible to observe the history chart of heart rate and oxygen saturation. Figure 15 presents an environment temperature monitoring in real time for two different rooms, with a simple alarm trigger. When the temperature value reaches a previously defined threshold value (which was set to 25 for this example), it triggers an alarm, as it is shown in room 2.

The generic analytical application intends to simulate the load of analytical applications over the platform, based on data already stored in databases. For that purpose, it is only required for this application to retrieve data from storage solutions, that would be necessary for machine learning or other analytic algorithms. Therefore, this application was integrated with both HBase and Cassandra drivers in order to establish connection to any database and fetch its data, along with the normal runtime of the platform.

5.3 SUMMARY

This chapter presented the followed approach in order to achieve the system architecture proposed in the previous chapter. It addresses the issues over the platform development, exposing the major decisions and explaining the implementation process.

In the decision process, all questions concerning the proposed approach were answered, highlighting the implementation of two data stores in the system (HBase and Cassandra) and the need to develop a component to integrate HBase with Kaa platform. Regarding the Data Processing block, it is important to underline the decision of developing two different components (Spark and a generic data analytic application) outside the central platform, to avoid unnecessary load to the system.

The implementation process is grouped into four stages explained in detail, including the cluster deployment of Kaa, HBase and Cassandra; the development of HBase log appender; the development of simulated medical sensors; and the implementation of Spark and the generic analytic application. The HBase log appender brings an innovative feature to Kaa platform, enabling to store data into HBase database and opening the possibility to perform a quantitative comparison between HBase and Cassandra data stores. Furthermore, Spark provides streaming tools extremely useful for medical data visualization and real time processing. Finally, the generic analytical application represents any component or software that might be integrated in the future into the system for analytic purposes.

Having the platform fully developed and operational, the next stage concerns the evaluation of the system.

6

CASE STUDIES / EXPERIMENTS

This chapter depicts the evaluation of the platform. Section 6.1 presents the setup of the experiments performed over the platform, while Section 6.2 presents their respective results and discussion. Finally, Section 6.3 exposes a brief summary of this experimental chapter.

6.1 EXPERIMENT SETUP

The implementation of the platform was performed in an experimental environment with virtual machines, in a local area network. Each virtual machine runs Ubuntu 16.04 and disposes of 2 CPU cores, 4 GB of RAM memory and 20 GB of available space in disk.

For this experiment setup, the cluster corresponding to the Central Platform, described in Section 5.2, was composed by three virtual machines with the specifications referred above. Each virtual machine represents one node of the cluster. In each node, it was deployed an instance of Kaa, Cassandra and HBase. Both Cassandra and HBase clusters were set with a replication factor of 2, which means that for every record there are written two copies. Furthermore, Cassandra consistency was set to "ONE", in order to improve the availability of the system.

Another virtual machine was dedicated for data source purposes. All simulated medical sensors were integrated in that machine, which works as an IoT gateway between those sensors and the platform.

As it was stated in Section 5.1, Spark and the generic analytical component were developed outside the platform cluster. In order to make them independent from each other, both components were assigned a dedicated virtual machine.

After setting up the experimental environment, it was possible to design the evaluation process of the platform performance. Two standard metrics widely used for that purpose include the throughput and the response time. The throughput represents the number of transactions per second handled by the platform, while the response time is the measure of time between a request operation by a client application and the respective answer by the server. For this master thesis, the considered response time corresponds to the time

interval between sending data to the platform and its storage in the respective database. To achieve that, sensors were adapted to send the timestamp at which data was generated, while databases already save the time at which data is stored. The difference between those timestamps is the response time considered in this project. It is important to guarantee that all machines have a synchronized clock in order to measure coherent timestamp values. Another metric evaluated was the queue of the platform, which represents the number of records that were sent to the platform but were not yet stored, in a specific instant.

Since the platform integrates two different NoSQL databases, it was possible to establish a comparison between the performance of the platform with HBase and with Cassandra. Based on that premise, the specified metrics and the experimental environment developed, there were designed four different scenarios.

6.1.1 *First scenario - Limits of the platform*

This scenario aims to discover how much write throughput the platform can handle with each database. This is especially useful for the management and design of future applications over the platform. For that purpose, the only load in the system is related to the storage of data sent from medical sensors.

In this experiment, data is sent at a starting rate of 2000 records per second and, for each minute that passes, it is increased by 500 records per second. This scenario simulates the situation where 2000 sensors are sending one sample per second to the platform and there is an increment of 500 sensors, also sending one sample per second, after every minute. It has a duration of 15 minutes, meaning that in the last minute there are 9000 sensors sending one sample per second.

The main objective of this test is to verify if the platform is able to store all data and analyze the response time associated with the increment of data. The experiment is performed three times for each database and the results are presented in 6.2.1.

6.1.2 *Second scenario - Constant write only applications*

This scenario was designed after the previous one, to evaluate the performance of the platform over different constant write throughput values. Based on the results from the first scenario, it is possible to divide this experiment in three groups with distinct write throughput values. In the first group, data is sent at a small constant rate, corresponding to 2000 records per second. Again, this illustrates the scenario where 2000 sensors are sending one sample per second into the platform. The second group presents 6000 sensors sending one sample per second, which represents the maximum throughput value without compromising the performance of the platform. It is important to highlight that this value

is a conclusion from the results obtained in the first scenario. In the third group, data is sent at an high throughput value, corresponding to 9000 records per second. All the experiment groups are performed during 12 minutes, where the first and the last minutes are disregarded, since they correspond to the initialization and closure processes, respectively.

The purpose of this experiment is to evaluate the performance of the platform with a previously specified number of samples, represented as the number of sensors. This evaluation concerns the data stored into the database, the response time and the state of the queue over the time. For each group, this experiment is performed three times, for each database. The results are presented in 6.2.2.

6.1.3 *Third scenario - Constant write and streaming applications*

This scenario focus on the impact of streaming applications on the normal process of sending and storing data into the platform. For this purpose, a Spark streaming application with simple data processing was developed. It consists of a Spark application to fetch the incoming data from thermometer sensors and, then, after every 5 seconds, calculate the average measured temperature. It is important to notice that this processing feature was implemented for concept proof. In this experiment, there were used 6000 sensors sending one sample per second into the platform.

Similar to the previous scenario, this experiment also intends to evaluate data stored in the database, response time and the state of queue over the time. However, this scenario increments the load of the system through the streaming application. It is also performed during 12 minutes, where the first and the last minutes are disregarded, since they correspond to the initialization and closure processes, respectively. It is performed three times for each database and the results are presented in 6.2.3.

6.1.4 *Fourth scenario - Constant write, streaming and analytic applications*

This scenario exposes a real world situation, where data is being continuously generated, stored, streamed and accessed. The experiment consists of sending data at a constant rate of 6000 records per second, implementing the Spark streaming application used in the previous scenario and implementing the generic analytical component responsible to fetch data from the databases. For experimental purposes, the analytical application performs read operations over the whole database every second during the execution of this experiment.

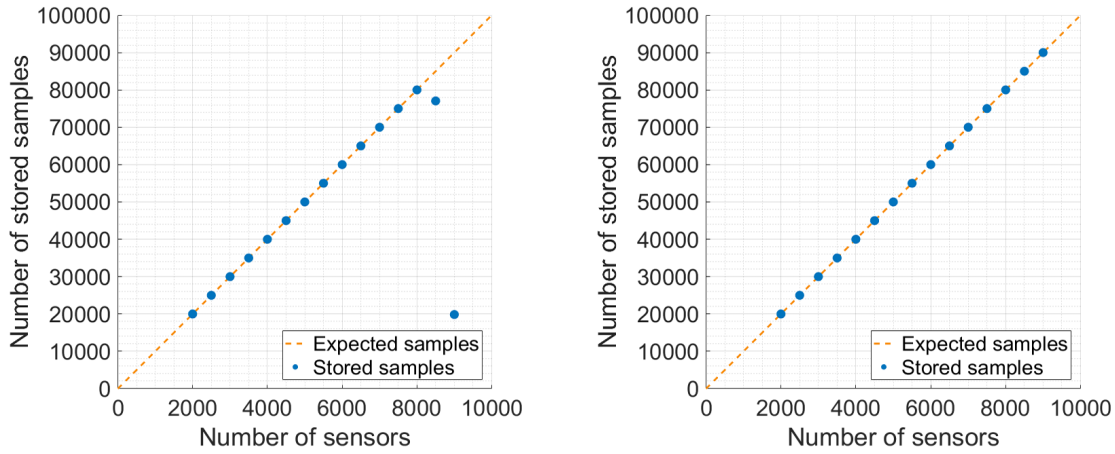
Similar to the previous scenario, it is also performed during 12 minutes, where the first and the last minutes are disregarded, since they correspond to the initialization and closure processes, respectively. Again, it is performed three times for each database and the results are presented in 6.2.4.

6.2 RESULTS

This section presents the results of the several tests described in the experimental setup. The discussion in detail for every result is addressed after each scenario, in order to provide a better comprehension of the obtained results.

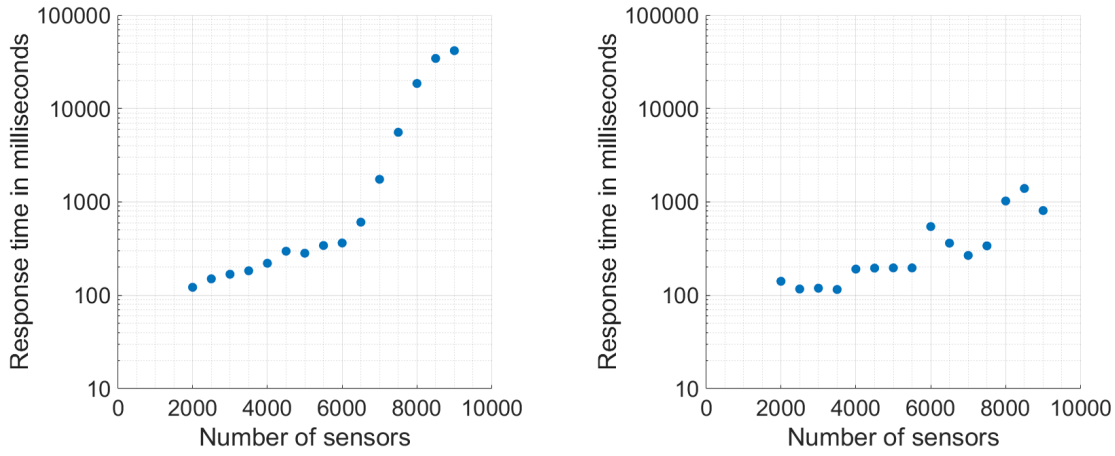
6.2.1 *First scenario results*

The results from the experiments of the first scenario are presented as charts in Figures 16 and 17. To avoid presenting a large quantity of data on the charts, the number of samples collected where grouped in intervals of ten seconds, meaning that for each number of sensors, sending one record per second, there were expected ten times the number of stored samples. Figure 16 shows the average number of stored samples and the number of samples expected per number of sensors, for each database, based on the three performed tests. Figure 17 presents the average response time per number of sensors in a logarithmic scale, which reduces the wide range of values to a more manageable and visible representation and, thus, allowing to cover a large range of values.



(a) Cassandra (b) HBase

Figure 16: Average number of stored samples per number of sensors for (a) Cassandra and (b) HBase. It is possible to observe a decline in the number of samples stored in Cassandra database for a number of sensors above 8000, while HBase stores all data.



(a) Cassandra (b) HBase

Figure 17: Average response time in milliseconds per number of sensors for (a) Cassandra and (b) HBase. In general, Cassandra presents higher values for the average response time than HBase, reaching values superior to 1 second for a number of sensors above 6500. HBase only reaches average response time values above 1 second for 8000 and 8500 sensors.

The first experimental scenario enables an overview of the platform limits in terms of write operations from medical sensors. The experiment starts with a few number of sensors sending one sample per second, 2000 to be precise, which are incremented by 500 after every minute, during 15 minutes. In the end of the experiment, it is reached a throughput of 9000 samples per second. As it was previously stated, since data was processed in intervals of 10 seconds, the expected data stored in the database after each 10 seconds is ten times the number of sensors operating in that interval. In Figure 16, it is presented the average number of samples that were actually stored, comparing to the expected stored samples, per number of sensors, for each database. It is possible to notice that the platform with HBase is able to store all the expected data, while with Cassandra it is compromised for a number of sensors above 8000, which is reflected by being unable to store all the expected data during the runtime of the test. In Figure 17, it is presented the average response time, during the experiment, per number of sensors. At the first glance, it is observed that both databases present response time values below 1 second (1000 milliseconds on the chart) for a throughput value below 6500 samples per second, represented by 6500 sensors sending one sample per second. With Cassandra, above that number of sensors, the response time reaches extremely high values, highlighting near 20 seconds for 8000 sensors, which causes the system to be compromised. With HBase, the platform only achieves an average response time of 1 second for 8000 sensors and above. It is interesting to notice that for 9000 sensors the average response time decreases to values below 1 second again, which might suggest that HBase is able to adapt to the high throughput of data over time. However, further experiments for that load need to be addressed in order to obtain a viable conclusion.

6.2.2 *Second scenario results*

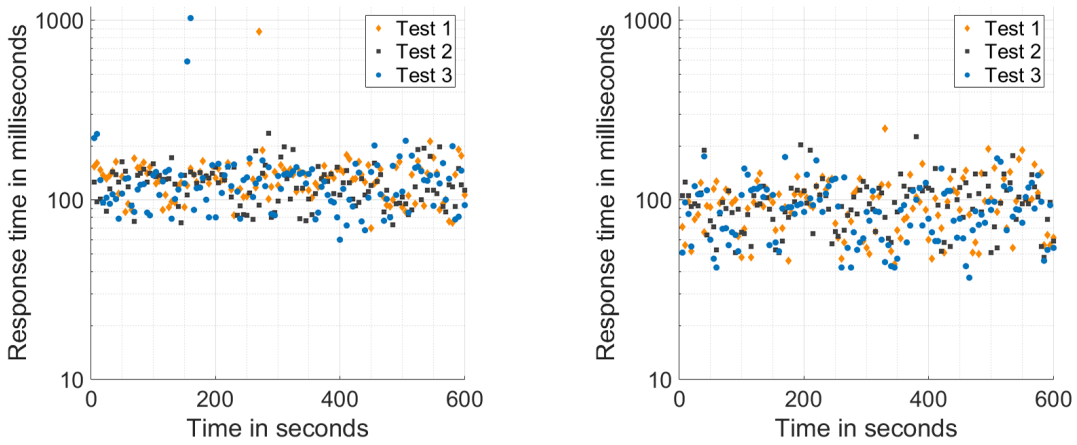
Based on the results from the first experimental scenario, which attempted to explore the limits of the platform, it is possible to differentiate three sub-scenarios. The first one regards the impact of small load into the platform, which was set to a throughput equal to the minimum tested in the first scenario, namely 2000 sensors, sending one sample per second. The second one concerns the maximum load without compromising the system with any database. Since data is sent in 1 second intervals, the system was considered compromised for response time values far above 1 second. Thus, based on the results from the first experimental scenario, a number of 6000 sensors per second was defined as the maximum load at which both databases perform efficiently. The last one is related to the behaviour of the platform in extreme conditions of write operations, which was set to a throughput equal to the maximum tested in the first scenario, namely 9000 sensors per second. The purpose of the second experimental scenario is to evaluate the platform performance with a constant throughput, contemplating those three sub-scenarios, during 10 minutes of execution time.

For each of the three groups, there are two figures showing the charts, for each database, of all the three performed tests. The first figure depicts the average response time of storing data during the 10 minutes of execution, while the second figure exposes the state of the queue at each represented instant.

Since data was sent with one second interval to the platform and the experimental time was equal to 10 minutes, presenting all data would difficult the comprehension of the charts. In order to reduce the number of samples in charts, data was processed in intervals of 5 seconds each, presenting the average response time for that interval and the state of the queue at each 5 seconds.

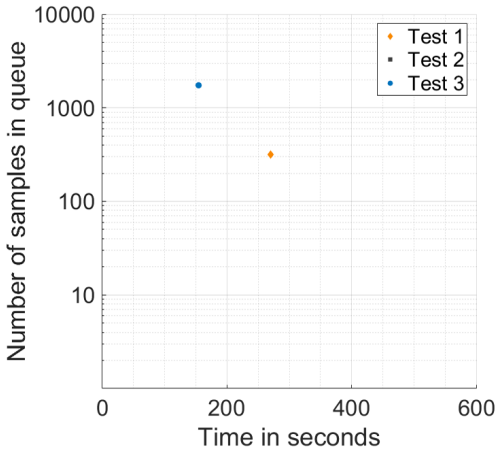
In addition, all figures are presented with a logarithmic scale to provide a more manageable and visible representation, as it was stated above, in the first scenario. Since the logarithmic scale does not allow zeros in its representation, the figures addressing the state of the queue at each instant only present the samples when the queue is not empty.

Constant write experiment with 2000 sensors



(a) Cassandra (b) HBase

Figure 18: Average response time of the platform for an input of 2000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase. In general, the platform presents response time values below 200 milliseconds for both cases.



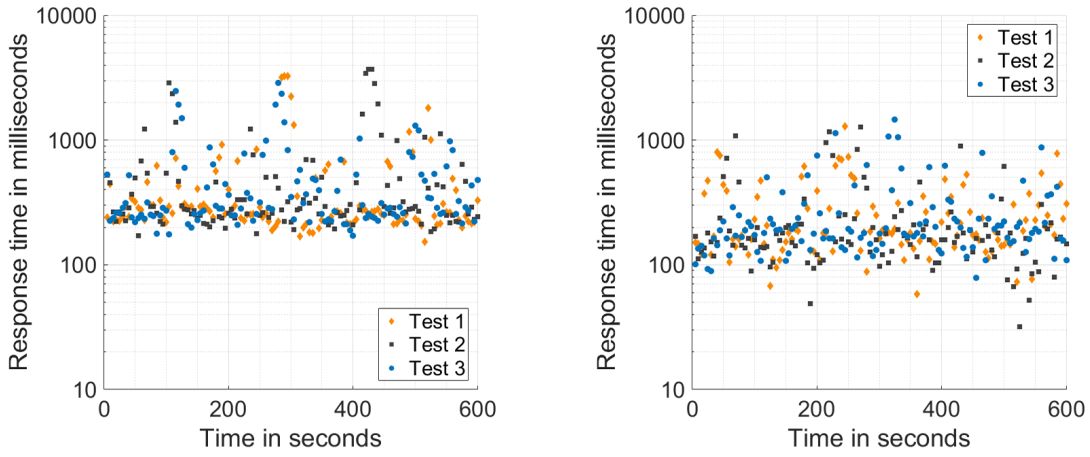
(a) Cassandra

Figure 19: Number of samples in queue for an input of 2000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra. In HBase the queue is always empty and, thus, it is not presented.

Concerning the experiment with 2000 sensors sending one sample per second, resulting in a throughput of 2000 records per second, both HBase and Cassandra are able to store all records efficiently. In Figure 18, it is observed that most of samples present an average response time below 200 milliseconds for both databases. It is also possible to notice that HBase presents a greater number of samples with an average response time below 100

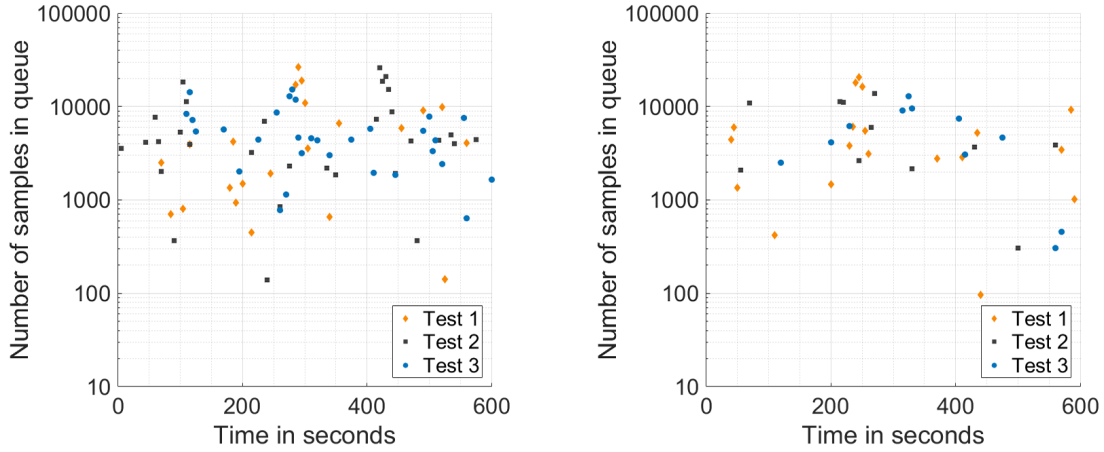
milliseconds than Cassandra, which allows to conclude that HBase presents slightly better results than Cassandra. With Cassandra, the Test 3 shows two outliers, one above 1 second and the other around 600 milliseconds, while Test 1 shows one outlier near 900 milliseconds. Those values might have been caused by other operations in the platform mechanism and can be despised from the discussion. Figure 19 attempts to present the number of samples in queue, for each 5 second interval. However, since the load of this experiment is small, in the overall the queue is always empty for both databases. In fact, for HBase the queue is always empty and, for that reason, the chart is not presented. As for Cassandra, due to the outliers referred with a response time above and near 1 second, the queue presents near 2000 samples and near 300 samples in the queue to be stored for Test 3 and Test 1, respectively. This highlights the fact that high response time values influence negatively the number of samples presented in queue, which is reflected on the increase of the queue.

Constant write experiment with 6000 sensors



(a) Cassandra (b) HBase

Figure 20: Average response time of the platform for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase. The platform presents response time values below 1 second for most samples, in both cases. However it is possible to observe several higher distinct values, especially with Cassandra.

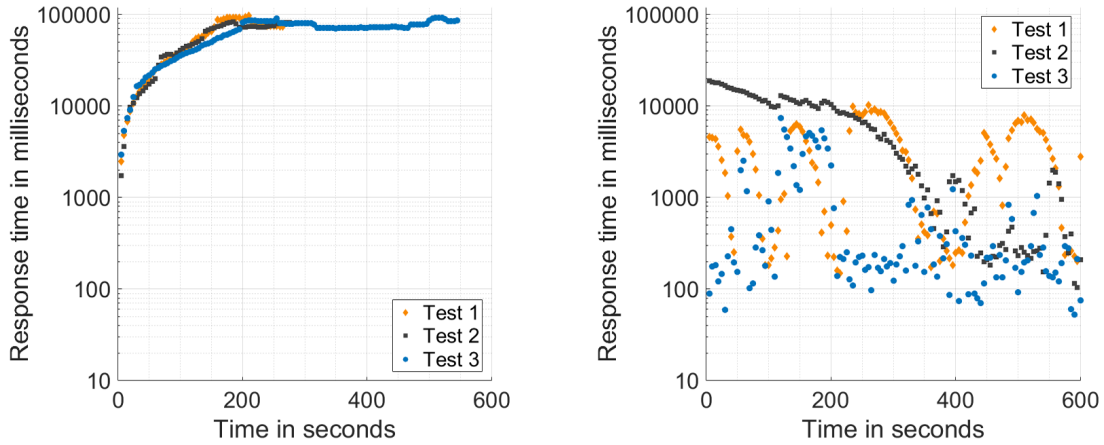


(a) Cassandra (b) HBase

Figure 21: Number of samples in queue for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase. During most of the time, the queue presents values below 10000 samples, showing empty values frequently in both cases. It is also important to notice that the queue is never saturated.

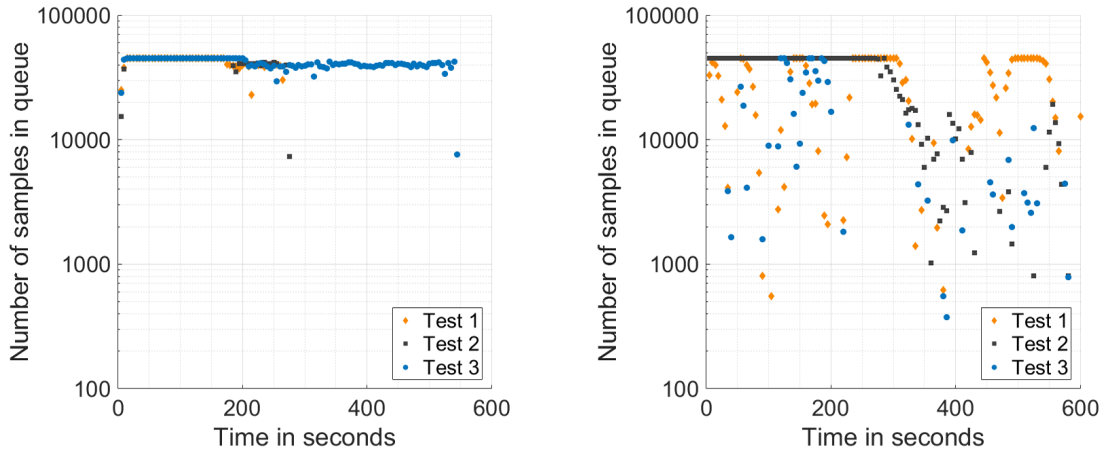
Regarding the 6000 sensors experiment, all records sent were successfully stored in both Cassandra and HBase. At the first glance, in Figure 20, it is possible to conclude, for both databases, that despite having a large number of samples below the 1 second average response time, there are several samples above it. This proves that for a number of sensors superior to 6000 sensors sending one sample per second, the efficiency of the platform might be compromised due to the high response time values. With Cassandra, both three tests present peaks of response time values above 2 seconds. With HBase, it is possible to observe that there are no samples above 2 seconds and most of the average response time values are located below 1 second. Furthermore, it is possible to notice that with HBase there is a large number of samples presenting a response time value between 100 and 200 milliseconds. On the contrary, with Cassandra, almost every sample presents a response time value above 200 milliseconds. In the overall, both databases present acceptable response time values, below 1 second, with the presence of some peak values that tend to decrease over time, highlighting the capability of the platform to handle those occurrences. Nevertheless, the platform performance with HBase presents slightly lower response time values, in general, than with Cassandra. This is also proved in Figure 21, where it is possible to observed a larger number of samples in queue for Cassandra than HBase. Once again, it is possible to relate the presence of larger number of samples in queue with the higher peaks of response time values, for the respective instant. Based on this figure, it is also important to notice that the queue is never saturated in any database.

Constant write experiment with 9000 sensors



(a) Cassandra (b) HBase

Figure 22: Average response time of the platform for an input of 9000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase. In general, the platform presents high and inconsistent response time values for both cases. With Cassandra, it presents extremely high values, having response times above 10 seconds for almost the whole time of the experiment duration. With HBase, the response time values are always below 20 seconds and reveal a tendency to decrease over time when a peak occurs.



(a) Cassandra (b) HBase

Figure 23: Number of samples in queue for an input of 9000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase. With Cassandra, the queue is always saturated, while with HBase the queue is saturated during the first 300 seconds and then it decreases, even reaching some empty values.

The 9000 sensors experiment represents an extreme condition to the platform, as it is shown in Figure 22. In this case, contrary to the other experiments with fewer number

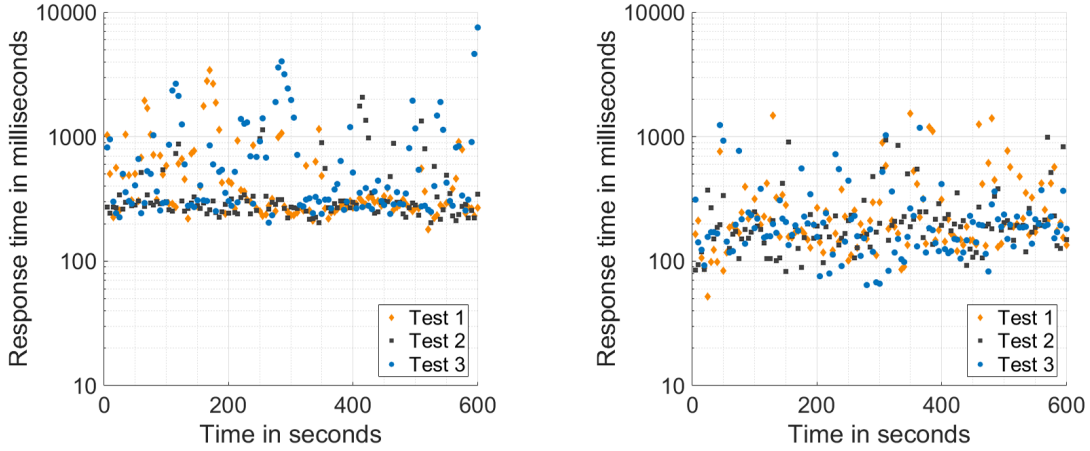
of samples per second, the platform with Cassandra not only presents extremely high response time values, but it was also unable to reach the end of the experiment time. In fact, it is possible to observe, from the lack of samples in the chart, that the platform stopped working around the 300 seconds of execution time for Tests 1 and 2, while Test 3 stops noticeably around 550 seconds. On the contrary, HBase is able to store all data during the whole experiment time. Figure 22 proves the great impact of this experiment on the platform performance, by presenting extremely high values of response time, especially with Cassandra. For this database, the platform presents an increase of the response time over time in every test, reaching values above 1 minute, which compromises the whole system. As for HBase, the response time values are frequently below 10 seconds. It is interesting to notice that Test 2, despite starting with an average response time of 20 seconds, tends to decrease to low response time values. The same phenomenon occurs every time the platform reaches a peak of high values, which is well represented in the Test 1. Test 3 presents low values for almost the whole experiment. While it is true to say that response time values far above 1 second might compromise a platform designed for sensors that send data per second, it is possible to state that the platform performance with HBase for large amounts of data provides better results than the platform with Cassandra. This fact is corroborated by Figure 23, which presents the state of the queue for this experiment. Since the response time values are frequently higher than 1 second, it is normal to encounter a large number of samples in the queue. It is important to notice that the chart regarding Cassandra database shows that the queue is saturated all the time, which is expected due to the extremely high values of the average response time. As for HBase, the queue was also saturated, but only during the first 300 seconds of the experiment time, corresponding to the higher values of response time, having decreased and reached a lower number of samples in queue after that. A note for Test 1, which despite presenting another moment of queue saturation after the 500 seconds, the platform is able to decrease the number of samples to lower values, once again, before the end of the experiment. The exception here is the HBase's Test 3, which presents mostly a low number of samples in queue during the test.

6.2.3 *Third scenario results*

The first and second scenarios provided information about the limits and behaviour of the platform for different values of throughput, represented as a number of sensors. Therefore, it was possible to conclude that the platform implemented on this experimental environment is capable of handling a maximum of 6000 sensors without compromising its performance with any database. For further evaluation, it was implemented a Spark streaming

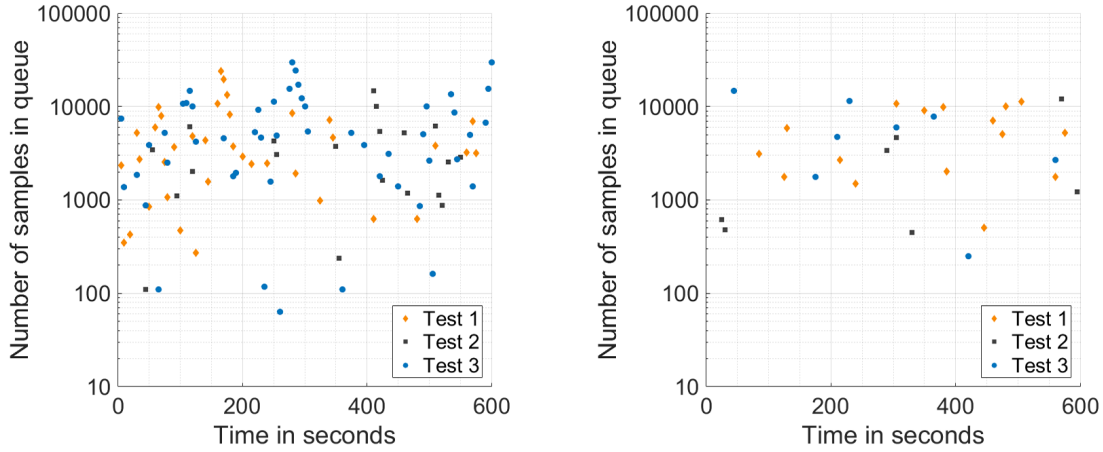
application, outside the platform cluster to avoid unnecessary load, for that quantity of sensors.

The third scenario explores the Spark streaming application impact on the overall platform performance. To avoid overwhelming the charts with data, the same five second interval strategy explained in Section 6.2.2 was adopted. Figure 24 addresses the average response time during 10 minutes with 6000 sensors sending one sample per second, while Figure 25 exposes the state of the queue during the experiment. The charts are also represented with the logarithmic scale mentioned on the previous scenarios.



(a) Cassandra (b) HBase

Figure 24: Average response time of the platform for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase, along with Spark streaming application. The platform presents response time values below 1 second for most samples. In general, HBase provides lower response time values.



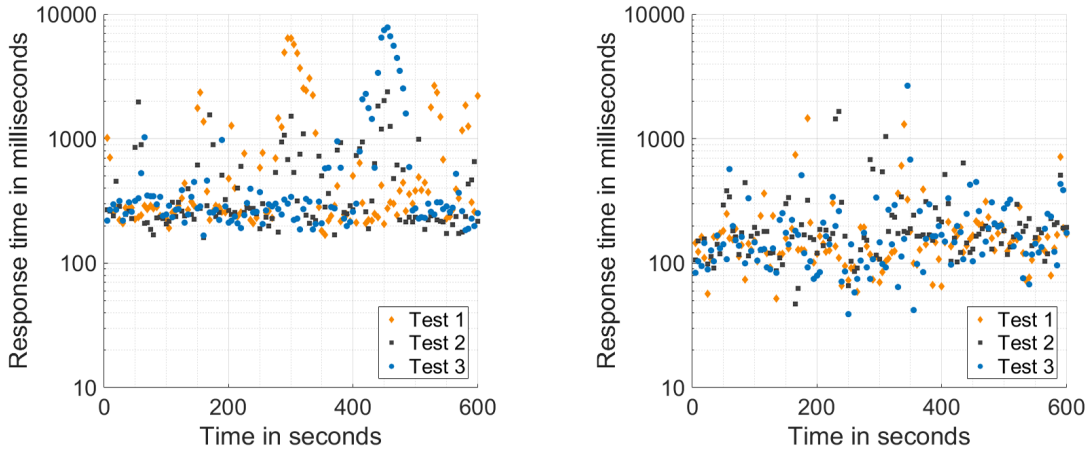
(a) Cassandra (b) HBase

Figure 25: Number of samples in queue for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase, along with Spark streaming application. In general, HBase provides lower number of samples in queue.

All data sent to each database was successfully stored in this experiment. Figure 24 presents the response time values for each database in the presence of the streaming application and, at the first glance, it is clear that the platform integrated with HBase presents lower response time values than with Cassandra. Even excluding the three outliers with values above 4 seconds from Test 3 in the Cassandra chart, it is possible to encounter several response time samples above 1.5 seconds. As for HBase, there is no samples above that value. Similarly to the experiment with only write operations from 6000 sensors, the platform with HBase presents a large number of samples with a response time value between 100 and 200 milliseconds while, with Cassandra, almost every sample presents a response time value above 200 milliseconds. Figure 25 evidences the difference between the number of samples in queue for the platform with HBase and with Cassandra. Despite the queue being frequently empty for both databases, which is inferred by the absence of the majority of the samples, there is a larger number of samples in the queue, during the experiment, for Cassandra than HBase. It also emphasizes the influence of high response time values on the number of samples in queue at each instant, where the response time peaks correspond to the larger number of samples in queue. Furthermore, it is possible to conclude that the impact of the Spark streaming application in the platform performance is not significant, since the resulting charts are similar to the ones in the figures of the experiment with 6000 sensors performing only write operations.

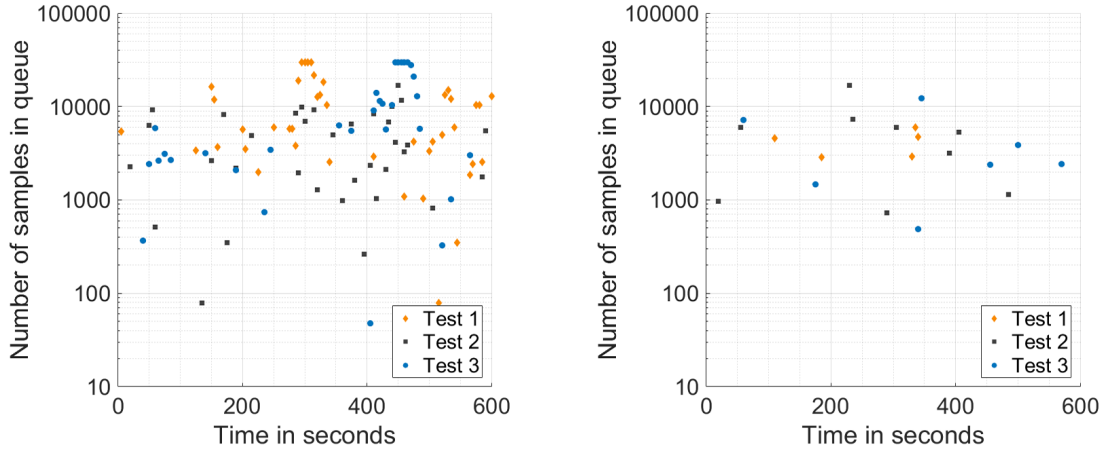
6.2.4 *Fourth scenario results*

The fourth scenario addresses a real world situation where data is sent from 6000 sensors (sending one sample per second), streamed with Spark streaming application and read with a generic application for analytics and machine learning purposes. In order to measure the impact of constant read and stream operations on the overall performance of the platform, it was performed the same evaluation method over the writes into both databases. The results are exposed in the charts of Figures 26 and 27. Once again, to avoid overwhelming the charts with data, the same five second interval strategy explained in Section 6.2.2 was adopted. Figure 26 presents the average response time during 10 minutes of execution, while Figure 27 presents the state of the queue during the experiment. The charts are also represented with the logarithmic scale mentioned on the previous scenarios.



(a) Cassandra (b) HBase

Figure 26: Average response time of the platform for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase, in a real world scenario. The platform presents response time values below 1000 milliseconds for most samples. In general, HBase provides lower response time values, while Cassandra presents more peaks in its response time values.



(a) Cassandra (b) HBase

Figure 27: Number of samples in queue for an input of 6000 sensors during an execution time of 10 minutes, per test, for (a) Cassandra and (b) HBase, in a real world scenario. In general, HBase provides lower number of samples in queue, and it is never saturated. In Cassandra it is possible to observe some saturation moments in the queue.

Similarly to the previous scenario, it was observed that all data was successfully stored for both Cassandra and HBase in this experiment. The average response time results, presented in Figure 26, show the occurrence of lower values for HBase, in general, than Cassandra. Comparing the results presented in this figure with the ones from the previous scenario, in Figure 24, it is not possible to infer the impact of this experiment in the platform with HBase. However, there is an increase of the response time peak values in the platform with Cassandra. In Cassandra response time chart from Figure 26, it is possible to observe several values above 2 seconds and two peaks with values around 6.5 and 8 seconds, respectively. While it is true that in Figure 24 the platform with Cassandra already presented several values above 2 seconds, there is a larger number of samples with values above 4 seconds, which might infer a slight impact of this scenario on the response time. This impact is highlighted on Figure 27, where it can be observed two distinct moments of queue saturation on the platform in the Cassandra chart. Nevertheless, it is important to mention that the queue remains empty for most of the experiment time, due to the absence of the majority of samples in the chart. As for HBase's chart, it is similar with its corresponding chart in Figure 25, presenting even fewer occurrences of samples in queue, which consolidates the conclusion about this scenario not having a significant impact on the platform with HBase.

6.3 SUMMARY

This chapter addresses the evaluation of the developed platform, starting with a detailed explanation of the experiment setup, including the experimental environment and tested scenarios, up to the presentation and discussion of the results.

The platform was implemented in an experimental environment with virtual machines running Ubuntu 16.04. The Kaa cluster, along with Cassandra and HBase, was implemented in three virtual machines, resulting in a three node cluster. The other components of the system, namely the client, Spark streaming and generic analytical applications, were deployed on individual virtual machines. Having the experimental environment configured, the evaluation process was divided into four different scenarios.

The first scenario was designed to provide an overview of the platform limits in terms of the write throughput. For that purpose, an increasing number of sensors, sending one sample per second, were integrated to increase the data throughput along the runtime. The results showed that the platform was able to efficiently handle the write throughput with both databases until 6000 records per second, starting to be compromised for values above that number.

The second scenario concerns the platform performance for a constant throughput over time. There were performed three experiments with different throughput values based on the number of sensors: 2000, 6000 and 9000. For 2000 sensors, the platform with both databases was able to efficiently store all the data. As for 6000 sensors, there were some high response time occurrences, but not sufficient to compromise the platform performance in both databases. The platform with HBase presented slightly better results than with Cassandra. The 9000 sensors experiment intended to simulate a scenario with more load than the platform was able to handle, resulting in the compromise of the system performance.

The third scenario integrated a Spark streaming application along with a constant throughput corresponding to 6000 sensors sending one sample per second, in order to evaluate the impact of data streaming on the platform. Once again, the results showed that the platform with HBase was the solution that provided lower response time values and less loaded queues. The impact of data streaming applications on the platform was not significant, since there was no visible variation on the results, comparing to the experiment with only write operations with the same throughput.

The fourth, and last, scenario was designed to simulate a real world situation. It consists of a constant throughput of 6000 sensors sending data per second, a Spark streaming application and a generic analytical application to perform read operations over the databases of the system. The impact of this scenario on the platform performance was slightly reflected on the increased response time of the platform with Cassandra. However, the impact on the platform with HBase was not noticeable.

In conclusion, the platform with HBase provided, generally, better results in comparison with Cassandra in terms of response time and loaded queues.

7

CONCLUSION

This chapter ends the dissertation by presenting the main conclusions that can be taken from the development of the described project, in Section 7.1. Some suggestions about the future work that can still be done over this platform are exposed in Section 7.2.

7.1 CONCLUSIONS

This master thesis main goal was the development of an IoT platform designed for data collection and analytic purposes for medical sensors, in the healthcare environment. For that purpose, it was necessary to collect and analyze several heterogeneous sources and bibliographic documents to assess the feasibility of that implementation, described as the first objective of this project in Section 1.2. A system capable of handling big data provides an excellent opportunity for big data analytics to get in action, in order to originate useful insights based on all that data. In the healthcare sector, this means that not only it will be possible to provide high quality and personalized care to patients, but it also enables the development of preventive care systems. Furthermore, the application of IoT technologies allows to establish connection between all devices and medical sensors in the respective healthcare environments, which is especially useful for remote patient monitoring. The development of a platform capable of integrating both the IoT and big data concepts was the main motivation for this dissertation.

During this project's development, many issues regarding these systems were addressed, using only open source software. The core of the designed solution was the implementation of Kaa IoT platform, responsible for handling all the communication and security processes, as well as managing all the Kaa endpoints, representing client applications, or in this case, medical sensors. In order to store and manage big data resulting from those sensors, the platform was provided with two NoSQL databases: HBase and Cassandra. Kaa already supported Cassandra, officially. However, the development of a component to establish the connection between Kaa and HBase was required, resulting in the major contribution of this master thesis. Thus, it was possible to perform the assessment of the system performance using each database. Furthermore, Spark was integrated in the system for processing

purposes and, especially, streaming applications. Another component introduced in the platform was developed for simulating the load that analytic and machine learning applications would cause by performing constant reads to the system databases. Achieving this system architecture concluded the second objective of this dissertation, successfully.

The third objective concerns the assessment of the developed platform performance and behaviour for an healthcare environment. The first experimental scenario presented an overview of the performance over an increasing number of medical sensors, which allowed to establish the limit throughput of the platform. The second experiment evaluated the performance over time, for a small, maximum and overwhelming number of sensors. In the experiments with small and maximum load, the platform was able to store all data with both databases. However, for an overwhelming number of sensors, only the platform with HBase was able to store all records sent. In general, the platform with HBase presents lower average response time values than with Cassandra. The third and fourth scenarios were designed to represent a real world situation, by adding data processing operations during the arrival of data from medical sensors. The impact of those implementations was not noticeable on the performance of the platform with HBase, while with Cassandra there was a slightly increase on the average response time values.

In sum, the platform proved to be suitable for the deployment in an healthcare environment, where there is a large amount of data generated in small time intervals. The implementation of HBase in Kaa platform represents a good contribution to the system, because not only presents an alternative with slightly better response time values than Cassandra, but also provides a great opportunity for analytical applications.

7.2 PROSPECTS FOR FUTURE WORK

The platform developed in this master thesis was designed as the base architecture for the implementation of several healthcare IoT solutions. Therefore, there are different ways to improve this platform or simply integrate more components to the whole system.

The deployment of the platform was performed on a three node cluster environment. Since all of its components are horizontally scalable, it is safe to say that the overall performance of the platform would increase with the addition of more nodes. Nevertheless, the assessment of the platform with more nodes would be an interesting subject to study.

Since the major goal of the platform developed was to provide a solution for big data collection and analytics for medical sensors, the next step would be to improve the analytic block of the system. The base architecture of the platform presented in this master thesis already provides the Spark implementation and an application to fetch data from both databases. Thus, it enables the use of Spark libraries (like Spark MLib) or the integration of other third party software to implement machine learning algorithms or other analytic

applications on the platform. A major contribution to this dissertation would be the implementation of analytic applications over real medical data to extract crucial information and even develop real time preventive systems.

Finally, despite being used mainly for data collection purposes in this project, Kaa platform also provides several other features, such as notification management and remote device provisioning and configuration. The exploration of these features could improve the capabilities of the current system.

BIBLIOGRAPHY

[1] Ahmad M Nagib and Haitham S Hamza. SIGHTED: A framework for semantic integration of heterogeneous sensor data on the internet of things. *Procedia Computer Science*, 83:529–536, 2016.

[2] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: opportunities and challenges. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 285–292. IEEE, 2015.

[3] Darrell M West. Improving health care through mobile medical devices and sensors. *Brookings Institution Policy Report*, 10:1–13, 2013.

[4] Jorge Gomez, Byron Oviedo, and Emilio Zhuma. Patient monitoring system based on internet of things. *Procedia Computer Science*, 83:90–97, 2016.

[5] Junaid Mohammed, Chung-Horng Lung, Adrian Ocneanu, Abhinav Thakral, Colin Jones, and Andy Adler. Internet of things: Remote patient monitoring using web services and cloud computing. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*, pages 256–263. IEEE, 2014.

[6] Iuliana Chiuchisan, Hariton-Nicolae Costin, and Oana Geman. Adopting the internet of things technologies in health care systems. In *Electrical and Power Engineering (EPE), 2014 International Conference and Exposition on*, pages 532–535. IEEE, 2014.

[7] YIN Yuehong, Yan Zeng, Xing Chen, and Yuanjie Fan. The internet of things in healthcare: An overview. *Journal of Industrial Information Integration*, 1:3–13, 2016.

[8] Peter Groves, Basel Kayyali, David Knott, and Steve Van Kuiken. The 'big data'revolution in healthcare. *McKinsey Quarterly*, 2:3, 2013.

[9] D Saidulu and R Sasikala. Understanding the challenges and opportunities with big data applications over a smart healthcare system. *International Journal of Computer Applications*, 160(8), 2017.

[10] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):3, 2014.

[11] Sebastian Salas-Vega, Adria Haimann, and Elias Mossialos. Big data and health care: challenges and opportunities for coordinated policy development in the eu. *Health Systems & Reform*, 1(4):285–300, 2015.

[12] Sanskruti Patel and Atul Patel. Abig data revolution in health care sector: Opportunities, challenges and technological advancements. *International Journal of Information*, 6(1/2), 2016.

[13] Kyoungyoung Jee and Gang-Hoon Kim. Potentiality of big data in the medical sector: focus on how to reshape the healthcare system. *Healthcare informatics research*, 19(2): 79–85, 2013.

[14] Apache Hadoop documentation. http://hadoop.apache.org/docs/stable/, Accessed: 2018-02-01.

[15] Apache HBase reference guide. http://hbase.apache.org/book.html, Accessed: 2018-02-01.

[16] Apache Cassandra documentation. http://cassandra.apache.org/doc/latest/, Accessed: 2018-02-01.

[17] Cheikh Kacfah Emani, Nadine Cullot, and Christophe Nicolle. Understandable big data: a survey. *Computer science review*, 17:70–81, 2015.

[18] Apache Spark documentation. https://spark.apache.org/docs/latest/, Accessed: 2018-02-01.

[19] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[20] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[21] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The internet of things for health care: a comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[22] Kaa IoT platform documentation. https://kaaproject.github.io/kaa/docs/v0.10.0/, Accessed: 2018-02-01.

[23] Rohit Suvarna, Sushant Kawatkar, and Dhanamma Jagli. Internet of medical things [iomt]. *International Journal*, 4(6), 2016.

[24] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 42–47. IEEE, 2013.

[25] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: from big data to big impact. *MIS quarterly*, pages 1165–1188, 2012.

[26] Marco Viceconti, Peter Hunter, and Rod Hose. Big data, big knowledge: big data for personalized healthcare. *IEEE journal of biomedical and health informatics*, 19(4): 1209–1215, 2015.

[27] Shen Yin and Okyay Kaynak. Big data for modern industry: challenges and trends [point of view]. *Proceedings of the IEEE*, 103(2):143–146, 2015.

[28] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[29] Jimeng Sun and Chandan K Reddy. Big data analytics for healthcare. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1525–1525. ACM, 2013.

[30] Nitesh V Chawla and Darcy A Davis. Bringing big data to personalized healthcare: a patient-centered framework. *Journal of general internal medicine*, 28(3):660–665, 2013.

[31] Ashwin Belle, Raghuram Thiagarajan, SM Soroushmehr, Fatemeh Navidi, Daniel A Beard, and Kayvan Najarian. Big data analytics in healthcare. *BioMed research international*, 2015, 2015.

[32] Sunghoon Ivan Lee, Hassan Ghasemzadeh, Bobak Mortazavi, Mars Lan, Nabil Alshurafa, Michael Ong, and Majid Sarrafzadeh. Remote patient monitoring: what impact can data analytics have on cost? In *Proceedings of the 4th Conference on Wireless Health*, page 4. ACM, 2013.

[33] Susan E White. A review of big data in health care: challenges and opportunities. *Open Access Bioinformatics*, 6:13–18, 2014.

[34] MH Padgavankar and SR Gupta. Big data storage and challenges. *International Journal of Computer Science and Information Technologies*, 5(2):2218–2223, 2014.

[35] K Hammad, M Fakharaldien, J Zain, and M Majid. Big data analysis and storage. In *International Conference on Operations Excellence and Service Engineering*, pages 10–11, 2015.

[36] Jaroslav Pokorny. Nosql databases: a step to database scalability in web environment. *International Journal of Web Information Systems*, 9(1):69–82, 2013.

[37] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[38] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam AM Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: advances, systems and applications*, 2(1):22, 2013.

[39] Yan Hu and Guohua Bai. A systematic literature review of cloud computing in ehealth. *arXiv preprint arXiv:1412.2494*, 2014.

[40] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[41] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[42] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE, 2011.

[43] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. The vertica analytic database: C-store 7 years later. *Proc. VLDB Endow.*, 5(12):1790–1801, August 2012. ISSN 2150-8097. doi: 10.14778/2367502.2367518.

[44] David Simmen, Karl Schnaitter, Jeff Davis, Yingjie He, Sangeet Lohariwala, Ajay Mysore, Vinayak Shenoi, Mingfeng Tan, and Yu Xiao. Large-scale graph analytics in aster 6: bringing context to big data discovery. *Proceedings of the VLDB Endowment*, 7(13):1405–1416, 2014.

[45] Florian M Waas. Beyond conventional data warehousing—massively parallel data processing with greenplum database. In *International Workshop on Business Intelligence for the Real-Time Enterprise*, pages 89–96. Springer, 2008.

[46] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[47] Kristina Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.", 2013.

[48] Voldemort project. http://www.project-voldemort.com/voldemort/, Accessed: 2018-02-01.

[49] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: The Definitive Guide: Time to Relax*. " O'Reilly Media, Inc.", 2010.

[50] Rabi Prasad Padhy, Manas Ranjan Patra, and Suresh Chandra Satapathy. Rdbms to nosql: reviewing some next-generation non-relational database's. *International Journal of Advanced Engineering Science and Technologies*, 11(1):15–30, 2011.

[51] Aisling O'Driscoll, Jurate Daugelaite, and Roy D Sleator. 'big data', hadoop and cloud computing in genomics. *Journal of biomedical informatics*, 46(5):774–781, 2013.

[52] Avita Katal, Mohammad Wazid, and RH Goudar. Big data: issues, challenges, tools and good practices. In *Contemporary Computing (IC3), 2013 Sixth International Conference on*, pages 404–409. IEEE, 2013.

[53] Abdul Ghaffar Shoro and Tariq Rahim Soomro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 2015.

[54] Ibm corporation. https://www.ibm.com/, Accessed: 2018-02-01.

[55] Cloudera, inc. https://www.cloudera.com/, Accessed: 2018-02-01.

[56] Dell emc. https://www.dellemc.com/, Accessed: 2018-02-01.

[57] Philip Russom et al. Big data analytics. *TDWI best practices report, fourth quarter*, 19(4): 1–34, 2011.

[58] Apache Zookeeper documentation. https://zookeeper.apache.org/doc/r3.4.8/, Accessed: 2018-02-01.

[59] Apache Flume documentation. https://flume.apache.org/FlumeUserGuide.html, Accessed: 2018-02-01.

[60] Ronald C Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. In *BMC bioinformatics*, volume 11, page S1. BioMed Central, 2010.

[61] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[62] Xuebin Chen, Shi Wang, Yanyan Dong, and Xu Wang. Big data storage architecture design in cloud computing. In *National Conference on Big Data Technology and Applications*, pages 7–14. Springer, 2015.

[63] Jeffrey Shafer, Scott Rixner, and Alan L Cox. The hadoop distributed filesystem: Balancing portability and performance. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 122–133. IEEE, 2010.

[64] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

[65] Manpreet Singh and Arshad Ali. *Big Data Analytics with Microsoft HDInsight in 24 Hours, Sams Teach Yourself*. Sams Publishing, 2015.

[66] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proceedings of the VLDB Endowment*, 5(12):2014–2015, 2012.

[67] Shiva Achari. *Hadoop Essentials*. Packt Publishing Ltd, 2015.

[68] Nikita Bhojwani and Asst Prof Vatsal Shah. A survey on hadoop hbase system. *Development*, 3(1), 2016.

[69] Amitanand S Aiyer, Mikhail Bautin, Guoqiang Jerry Chen, Pritam Damania, Prakash Khemani, Kannan Muthukkaruppan, Karthik Ranganathan, Nicolas Spiegelberg, Liyin Tang, and Madhuwanti Vaidya. Storage infrastructure behind facebook messages: Using hbase at scale. *IEEE Data Eng. Bull.*, 35(2):4–13, 2012.

[70] Giorgos Saloustros and Kostas Magoutis. Rethinking hbase: Design and implementation of an elastic key-value store over log-structured local volumes. In *Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on*, pages 225–234. IEEE, 2015.

[71] An-ping Xiong and Jiao Zou. Research of dynamic load balancing strategy on hbase. 2015.

[72] Tyler Harter, Dhruba Borthakur, Siying Dong, Amitanand S Aiyer, Liyin Tang, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Analysis of hdfs under hbase: a facebook messages case study. In *FAST*, volume 14, page 12th, 2014.

[73] Lars George. *HBase: the definitive guide: random access to your planet-size data.* " O'Reilly Media, Inc.", 2011.

[74] Ankur Khetrapal and Vinay Ganesh. Hbase and hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*, pages 22–28, 2006.

[75] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8. Boston, MA, USA, 2010.

[76] Huajin Wang, Jianhui Li, Haiming Zhang, and Yuanchun Zhou. Benchmarking replication and consistency strategies in cloud serving databases: Hbase and cassandra. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 71–82. Springer, 2014.

[77] Apache Cassandra. Apache cassandra. *Google Scholar*, 2015.

[78] E Hewitt. Cassandra-the definitive guide: Distributed data at web scale. definitive guide series, 2010.

[79] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. A big data modeling methodology for apache cassandra. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 238–245. IEEE, 2015.

[80] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 20, 2011.

[81] Mat Brown. *Learning Apache Cassandra*. Packt Publishing Ltd, 2015.

[82] Cassandra architecture. https://www.javatpoint.com/cassandra-architecture, Accessed: 2018-02-01.

[83] Michael Armbrust, Tathagata Das, Aaron Davidson, Ali Ghodsi, Andrew Or, Josh Rosen, Ion Stoica, Patrick Wendell, Reynold Xin, and Matei Zaharia. Scaling spark in the real world: performance and usability. *Proceedings of the VLDB Endowment*, 8(12): 1840–1843, 2015.

[84] R Shyam, Sachin Kumar, Prabaharan Poornachandran, and KP Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

[85] Mohammad Abu Alsheikh, Dusit Niyato, Shaowei Lin, Hwee-Pink Tan, and Zhu Han. Mobile big data analytics using deep learning and apache spark. *IEEE network*, 30(3): 22–29, 2016.

[86] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[87] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[88] Sarita Agrawal and Manik Lal Das. Internet of things—a paradigm shift of future internet applications. In *Engineering (NUiCONE), 2011 Nirma University International Conference on*, pages 1–7. IEEE, 2011.

[89] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7): 1497–1516, 2012.

[90] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[91] Alok Kulkarni and Sampada Sathe. Healthcare applications of the internet of things: A review. *International Journal of Computer Science and Information Technologies*, 5(5): 6229–6232, 2014.

[92] R Senthilkumar, RS Ponmagal, and K Sujatha. Efficient health care monitoring and emergency management system using iot. *International Journal of Control Theory and Applications*, 9(4):137–145, 2016.

[93] Cecile Davis, Miriam Bender, Tyler Smith, and Jason Broad. Feasibility and acute care utilization outcomes of a post-acute transitional telemonitoring program for underserved chronic disease patients. *Telemedicine and e-Health*, 21(9):705–713, 2015.

[94] Bruno MC Silva, Joel JPC Rodrigues, Isabel de la Torre Díez, Miguel López-Coronado, and Kashif Saleem. Mobile-health: A review of current state in 2015. *Journal of biomedical informatics*, 56:265–272, 2015.

[95] Shivayogi Hiremath, Geng Yang, and Kunal Mankodiya. Wearable internet of things: Concept, architectural components and promises for person-centered healthcare. In *Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on*, pages 304–307. IEEE, 2014.

[96] Reza Rawassizadeh, Blaine A Price, and Marian Petre. Wearables: Has the age of smartwatches finally arrived? *Communications of the ACM*, 58(1):45–47, 2015.

[97] Keith M Diaz, David J Krupka, Melinda J Chang, James Peacock, Yao Ma, Jeff Goldsmith, Joseph E Schwartz, and Karina W Davidson. Fitbit®: An accurate and reliable device for wireless physical activity tracking. *International journal of cardiology*, 185: 138–140, 2015.

[98] Muaddi Alharbi, Adrian Bauman, Lis Neubeck, and Robyn Gallagher. Validation of fitbit-flex as a measure of free-living physical activity in a community-based phase iii cardiac rehabilitation population. *European journal of preventive cardiology*, 23(14): 1476–1485, 2016.

[99] Nga H Nguyen, Nyssa T Hadgraft, Melissa M Moore, Dori E Rosenberg, Chris Lynch, Marina M Reeves, and Brigid M Lynch. A qualitative evaluation of breast cancer survivors' acceptance of and preferences for consumer wearable technology activity trackers. *Supportive Care in Cancer*, 25(11):3375–3384, 2017.

[100] Igor Bisio, Fabio Lavagetto, Mario Marchese, and Andrea Sciarrone. A smartphone-centric platform for remote health monitoring of heart failure. *International Journal of Communication Systems*, 28(11):1753–1771, 2015.

[101] Robert SH Istepanian, Sijung Hu, Nada Y Philip, and Ala Sungoor. The potential of internet of m-health things "m-iot" for non-invasive glucose level sensing. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 5264–5266. IEEE, 2011.

[102] Timothy Malche and Priti Maheshwary. Harnessing the internet of things (iot): A review. *Int. J*, 5(8), 2015.

[103] IBM Watson IoT Platform. https://www.ibm.com/internet-of-things, Accessed: 2018-02-01.

[104] Amazon Web Services IoT Core. https://aws.amazon.com/iot-core/, Accessed: 2018-02-01.

[105] Cisco IoT Solutions. https://www.cisco.com/c/en/us/solutions/internet-of-things/overview.html, Accessed: 2018-02-01.

[106] Subhasri Duttagupta, Mukund Kumar, Ritesh Ranjan, and Manoj Nambiar. Performance prediction of iot application: An experimental analysis. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 43–51. ACM, 2016.

[107] Microsoft Azure IoT Platform. https://azure.microsoft.com/en-us/suites/iot-suite/, Accessed: 2018-02-01.

[108] DeviceHive open source IoT Platform. https://devicehive.com/, Accessed: 2018-02-01.

[109] OpenIoT Platform. http://www.openiot.eu/, Accessed: 2018-02-01.

[110] Pankaj Ganguly. Selecting the right iot cloud platform. In *Internet of Things and Applications (IOTA), International Conference on*, pages 316–320. IEEE, 2016.

[111] ThingSpeak open source IoT Platform. https://thingspeak.com/, Accessed: 2018-02-01.

[112] T Jyothi. Open source middleware for internet of things. *IJIREEICE*, 2016.

[113] Cyber Vision website. http://www.cybervisiontech.com/, Accessed: 2018-02-01.

[114] Apache License 2.0. https://www.apache.org/licenses/LICENSE-2.0, Accessed: 2018-02-01.

[115] Cao Lu. Comparison between sensiblethings and kaa platform, 2016.

[116] Apache Avro documentation. https://avro.apache.org/docs/current/, Accessed: 2018-02-01.

[117] NJ Manson. Is operations research really research? *Orion*, 22(2):155–180, 2006.

[118] Dina Wahyuni. The research design maze: Understanding paradigms, cases, methods and methodologies. 2012.

[119] Martin S Olivier. A research methods course in it.

[120] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[121] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[122] V Vaishnavi, W Kuechler, and S Petter. Design science research in information systems. 2004.

[123] GitHub - HBase log appender project. https://github.com/JoaoRei/kaa/tree/master/server/appenders/hbase-appender, Accessed: 2018-04-05.

[124] Open-Source Integrated Clinical Environment. https://www.openice.info, Accessed: 2018-04-05.

[125] Freeboard - Free dashboard for Internet of Things visualization. http://freeboard.io/, Accessed: 2018-04-05.

NB: place here information about funding, FCT project, etc in which the work is framed. Leave empty otherwise.