Final

OpenDataHub
An Open Dataset Management System

MASTER DISSERTATION

Orêncio Rodolfo Abreu Gonçalves
MASTER IN INFORMATICS ENGINEERING

February | 2016

OpenDataHub

An Open Dataset Management System

Faculdade de Ciências Exatas e da Engenharia

Madeira Interactive Technologies Institute

Universidade da Madeira

This dissertation is submitted for the degree of

Master of Computer Science

Advisor: Prof. Duarte Nuno Jardim Nunes

Co-Advisor: Filipe Quintal

Co-Advisor: Lucas Pereira

By

Rodolfo Gonçalves

2015

First, I would like to thank my family and friends for all their support and encouragement for tenacity throughout the process.

A special feeling of gratitude to my loving parents, Alfredo Ramos and Maria Ferreira whose have always have been understanding on moments that I could not be present.

Thank to my advisors, who always accompanied me and helped me defining our guideline and M-ITI (Madeira Interactive Technologies Institute), for the providence of the space and resources to make the development of this project possible, and for the opportunity to participate on a project that really excited me for its challenges.

I would also like to thank to all of those who directly or indirectly contributed to my knowledge during my graduation and master's degree on Universidade da Madeira and M-ITI.

Thank you all!

Abstract

This thesis presents a cloud-based software platform for sharing publicly available scientific datasets. The proposed platform leverages the potential of NoSQL databases and asynchronous IO technologies, such as Node.JS, in order to achieve high performances and flexible solutions.

This solution will serve two main groups of users. The dataset providers, which are the researchers responsible for sharing and maintaining datasets, and the dataset users, that are those who desire to access the public data. To the former are given tools to easily publish and maintain large volumes of data, whereas the later are given tools to enable the preview and creation of subsets of the original data through the introduction of filter and aggregation operations.

The choice of NoSQL over more traditional RDDMS emerged from and extended benchmark between relational databases (MySQL) and NoSQL (MongoDB) that is also presented in this thesis. The obtained results come to confirm the theoretical guarantees that NoSQL databases are more suitable for the kind of data that our system users will be handling, i. e., non-homogeneous data structures that can grow really fast.

It is envisioned that a platform like this can lead the way to a new era of scientific data sharing where researchers are able to easily share and access all kinds of datasets, and even in more advanced scenarios be presented with recommended datasets and already existing research results on top of those recommendations.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

In this chapter, we introduce the motivation and biggest challenges to this work. Here we also identify the contributions of this thesis.

In the “Motivation” section we explore several issues related to research data sharing, identifying these as motivations to develop our vision. We start by introducing our problem context and proceed with the identification of the main issues reported by the community through public surveys.

In the “Vision” section we present the overall vision of our work. We start by identifying the need and the problems with current research data sharing tools. We then finalize by presenting an overview of our proposed solution for the identified problems. This solution will be described in detail in the remaining chapters of this thesis.

The “Challenges in our Vision” section aims to explore the most general and likely challenges to occur during the development of the proposed solution.

Lastly, in the “Thesis Contributions”’ we present the main contributions of our work to the scientific research and informatics engineering communities.

1.1 Motivation

As technology evolves and considering the complexity of many scientific problems, it becomes increasingly difficult for scientists to conduct ground-breaking research on their own. This not only promotes communication and cooperation among different researchers from distinct disciplinary backgrounds as well as it makes it a necessity.

This is why scientific research in the 21st century is more data intensive and collaborative than in the past. The amount of data collected, analysed, re-analysed and stored has increased in the past years due to the developments in computational simulation and modelling, automated data acquisition, and communication technologies and its sharing shall be promoted in order to contribute to scientific development [1].

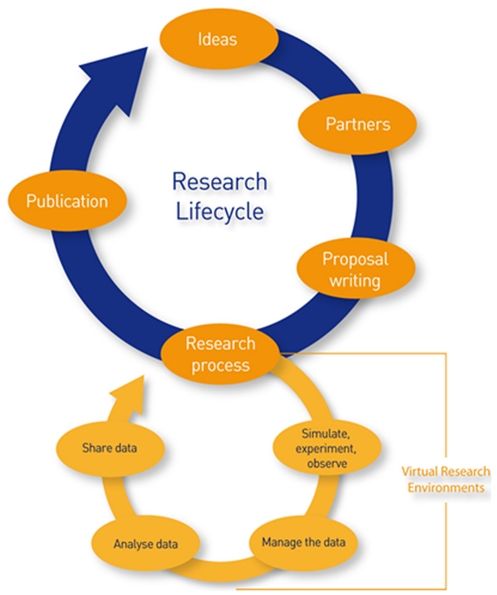


Figure 1.1 –Research Lifecycle [1]

The “Data Sharing by Scientists: Practices and Perceptions” [1], article reports the results of a survey with 1329 scientists related to their current data sharing practices and their perceptions of the barriers and enablers of data sharing. It presents a lifecycle that originates from the merge of research and data lifecycles.

Figure 1.1 shows the Research Lifecycle that identifies research and data lifecycles as being dependent of each other, as data is an indispensable element of scientific research.

Researchers started not only to share their results but also the raw data used in their scientific experiments. This phenomenon is also seen as an important measure to advance scientific results since data analysis is critical as it forms the basis for good scientific decisions and a wiser management and use of resources. Sensorial data for data mining for instance, its sharing enable users to reuse sensor measurements without having to invest time and money in the implementation of a new sensorial network.

Such cooperation have benefited different science fields like medicine, academia, government and business [2], enabling different perspectives on how data is modelled or represented, thus allowing further verification of the results and extending research from prior results.

"Data Sharing by Scientists: Practices and Perceptions" [1] authors' identify several advantages in data sharing. Between them, the identification of *data re-analysis* was identified as an important measure to help on results verification, since different interpretations or approaches to existing data contribute to scientific progress - especially when involving different research fields.

Another advantage worth mentioning is that well-managed, *long-term data* preservation helps retain its integrity and, when available, minimizes (re-) collection of data and improve resources management.

Furthermore, the authors also identify *replication studies* as resources for training tools for new generations of researchers and data as being not only the outputs of research but also as inputs to new hypotheses, enabling new scientific insights and driving innovation.

In "PARSE Insight" [3], a two-year project, the authors performed a survey report concerning the preservation of digital information in science. It consisted of a number of surveys to gather information about the practices, ideas, and needs of research communities regarding the preservation of digital research data.

The survey stakeholders were asked to give their opinion about why research data should be preserved, basing their answers on seven reasons defined by the researchers.

Table 1.1 shows the top 3 reasons for each stakeholder showing us that all stakeholders have stated that if research is publicly funded the data should be preserved as it belongs to the public as well.

Table 1.1 - Cross analysis of top 3 reasons for preservation

TOP 3 Reasons for preservation	
Research	
1	It will stimulate the advancement of science.

2	If research is publicly funded, the results should become public property and therefore properly preserved.
3	It allows for re-analysis of existing data.
Data management	
1	It is unique.
2	It potentially has economic value
3	If research is publicly funded, the results should become public property and therefore properly preserved.
Publishing	
1	It will stimulate the advancement of science
2	If research is publicly funded, the results should become public property and therefore properly preserved.
3	It allows for re-analysis of existing data

Yet, despite the importance of data preservation and sharing, there are several issues regarding the promotion of these activities and in how it should be done. In most the cases there are no standards or models to store and share data and there is no awareness for which data formats or structures should used. In general, available data also lacks the necessary metadata, which would allow the community to easily interpret the available data.

Storing and managing large volumes of data is another barrier to data sharing. Research data emerges from collection, observation or even creation (by simulation) for purposes of analysis which produces large volumes of data thus making the sharing and preservation of this highly heterogeneous amount of information a very hard and time consuming task.

On the "Data Sharing by Scientists: Practices and Perceptions" [1] survey most of the participants (59.8%) reported being satisfied with their processes for the initial, and short-term parts of the data or research but are not satisfied with long-term data preservation (73%). Many organizations do not provide support to their researchers for data management both in the short and long-term. However, the study shows that, if certain conditions are met (such as formal citation and sharing reprints) respondents agree that they are willing to share their data and re-use others' data. In addition to insufficient time, there is the lack of funding to make researcher's data available electronically.

Respondents of the PARSE survey also identified the top threats for scientific data preservation. Table 1.2 presents the top three threats identified; stakeholders' defined technical failure and inability to understand the meaning of the data as very important.

Table 1.2 - Cross analysis of top 3 threats to preservation

TOP 3 Threats to preservation	
Research	
1	Lack of sustainable hardware, software or support of computer environment may make the information inaccessible.
2	The current custodian of the data, whether an organization or project, may cease to exist at some point in the future.
3	Users may be unable to understand or use the data e.g. the semantics, format or algorithms involved.
Data management	
1	Lack of sustainable hardware, software or support of computer environment may make the information inaccessible.
2	Users may be unable to understand or use the data e.g. the semantics, format or algorithms involved.
3	The current custodian of the data, whether an organization or project, may cease to exist at some point in the future.
Publishing	
1	The current custodian of the data, whether an organization or project, may cease to exist at some point in the future.
2	Lack of sustainable hardware, software or support of computer environment may make the information inaccessible.
3	Evidence may be lost because the origin and authenticity of the data may be uncertain.

In “Data Sharing by Scientists: Practices and Perceptions” [1] respondents' identify the lack of awareness about the importance of metadata among the scientific community in order to make data and datasets retrievable in the future as a major problem. Metadata involvement is crucial in dealing with problems regarding data management; input and training modules must be a part of systems to assist scientists with preparing their data and datasets to be retrievable into the future. Adherence to formal metadata standards is crucial to retrieval effectiveness.

1.2 Vision

Cooperation between researchers from different backgrounds is an essential part of the scientific process. This need has been identified since the beginning of modern science and fomented the birth of the Internet as we know it.

Many datasets are available on huge compressed files and most of the time without offering metadata or the chance for a preview of the data thus forcing researchers to download large volumes of data without having a clear picture of its structure, format or even its origin or context.

Being the Internet the biggest communication network in the world, we believe that a possible approach for the problems presented above would be a cloud-based system that could allow researchers to publish and maintain their data on the Internet, while enabling others to quickly explore and create subsets of the original data. A web platform where the community could query public datasets and have access to its metadata, have a preview of the data itself or even query the original data retrieving only the information that is considered relevant for the researcher and its work.

A solution like this could help overcome several barriers on this topic. Such a web platform would serve two main groups of users, namely, the dataset providers and the dataset users.

Dataset providers would be the researchers or entities that whish to publish scientific data originating from a research study or project. Our envisioned system should not present any barriers for these individuals to publish their data. The web platform should supply an interface where dataset providers could create metadata and a description of their datasets, upload and maintain their data.

Dataset users would be the community that has the interest to access and explore the public data. A web solution containing the proper metadata could provide functions for data query and preview. It should allow filtering the data according to researchers' needs or even selecting only the fields that are considered important for their purposes. Allowing researchers to preview the resulting data before downloading and using it are important and interesting features that could promote the data re-use and facilitate the analysis of public scientific datasets.

This vision led to the creation of “OpenDataHub”, a web-platform for scientific data sharing and management.

1.3 Challenges in our Vision

In today’s digital age, a massive amount of data is steadily being produced from various sources, such as sensors, social media and GPS signals. This large amount of data is known as *Big Data*, one of the most discussed topics in digital information. It can be described as massive volumes of both structured and unstructured data that is so large that it is difficult to process with traditional database and software techniques.

The characteristics that broadly distinguish Big Data are the “3 V’s”: more volume, more variety and higher rates of velocity, i.e., faster insertions and reads [4].

Due to the high volume and the *non-homogeneous* data structures, *scalability and performance* on data query are the two main topics to be aware of. Traditional single-node data storage strategies are no longer a viable solution to big data problems. Regardless of the database technology, the solution shall pass by data *sharding*, i.e., breaking a large monolithic database into multiple, smaller, faster, more easily managed database instances across multiple servers, known as *shards*. [1]

It is not easy to overtake these problems affecting the task of data maintenance and sharing. Scientists used to have barriers when trying to share their data, since there is not much offer of online platforms that allow the data publication, while still giving users the ability to understand the data structure and meaning. This can therefore be identified as a challenge to our vision, the ability to create a *representation of public data in order to give users the data perception of its content, context and structure*.

[1] Data sharding, http://searchcloudcomputing.techtarget.com/definition/sharding [last accessed 22/02/2015]

Due the large amount of data and its non-homogeneous structure or data format, it is not easy to define an approach to represent the published data. Most of the time, data is available in big compressed files containing unknown data structures. Building an easy to interpret and clear representation of these large amounts of data has been a problem difficult to address.

Most of the time this kind of data is based on time series and a possible representation for data of this nature is through graphics or simply providing a preview of the original raw data.

The format defined by the researcher when publishing the data may not be the most appropriate for the user needs. This presents another challenge to our vision, the capability to *transform public data into different data formats according to the user requirements*.

During the course of this document we will have a closer look on how we intend to address these problems, referring in more detail the problem current state and developing an approach strategy.

1.4 Thesis Contributions

1.4.1 Benchmark

The first contribution of this work is the result from a benchmark between two different database technologies, SQL and NoSQL. More specifically, we developed a benchmark between MySQL and MongoDB, two of the most well known technologies on the data storage world. Ultimately, this evaluation enabled us to decide which technology would be a better fit for our application purposes, the OpenDataHub.

The benchmark is based on SustData dataset [5], a public scientific dataset related to electricity energy data collected from four energy monitoring and eco-feedback deployments that were done for the SINAIS[2] research project [6].

[2] Sustainable Interaction with social Networks, context Awareness and Innovative Services a 3-year research project which main goal was to raise the understanding and the awareness towards motivating people to consume more sustainably.

This evaluation benchmarks the performance of data reading and writing operations using both technologies. It focuses on the electric energy consumption of three different homes for an overall of 10 million records distributed among fifteen CSV files. The progressive insertion and read times for the different sets of data resulted in a performance curve until all the 10 million records were introduced on both databases.

Ultimately, in this contribution we provide an in-depth benchmark of these two technologies according to the following four dimensions: 1) time to read and write; 2) database size; 3) scalability; and 4) pre-presentation (i.e., fetching the results before presenting them to the users).

1.4.2 OpenDataHub

OpenDataHub is the result of the implementation of our vision in the real world. It is a web platform that offers researchers' data management engines, supporting CRUD operations over research data. It enables dataset users to consult and download data in both CSV and JSON formats.

As a use case, it holds the discussed SustData dataset with five different collections referring to energy consumption, power and users events, electric production and environmental conditions.

The system enables dataset users to access each of these collections and create different queries over the same data. It contains a graphic interface where it is possible to apply different filters, manipulate which fields should be presented and group the resulting records. The data is presented to the user without the need of a previous download, through two possible representations: Raw data in a simple tabular format or in graphical representation. After querying, users can also download the data resulting from the data manipulation operations in either CSV or JSON formats.

For dataset providers, OpenDataHub offers the possibility to create new collections and define its metadata. Defining metadata is a very important task in order to describe the data structure and serving as configurations for the definition of which query operations or data representations are possible. Moreover, OpenDataHub enables the possibility to define the data accessibility, a collection may not be ready for publication so researchers can define a collection as public or not.

In order to identify users as dataset providers or dataset users OpenDataHub offers a user account management engine, which enables the definitions of the permissions for each type of user.

1.5 Document organization

The remaining of this document is organized as follows:

In Chapter 2 we present an overview of the already existing solutions when it comes to data sharing and briefly discuss how they compare to OpenDataHub. We then provide some technical background regarding some of the most well-known technologies for web development and data management, and justify our choices for developing OpenDataHub.

In order to confirm our decisions over the technologies and having SustData public dataset as a test case, we developed a benchmark between SQL and NoSQL that is described in Chapter 3.

In Chapter 4 we provide an in-depth explanation of the OpenDataHub system. To this end we follow the waterfall software analysis process, defining system requirements, use cases, prototypes as well as the application and database architectures and the selected technologies for the solution development.

Lastly, in Chapter 5 we present the conclusions that emerged from this thesis and outline future work.

Chapter 2 Background research

2.1 State of the art

Along the years, the need to share and analyse other researchers' data has grown and with this, several solutions to overcome the barriers implicit to this kind of job have emerged. In this chapter we provide an overview of some of the solutions that already exist to address some of the issues related to the sharing of scientific data.

We start with the well-known "*UC Irvine Machine Learning Repository*"[3], a scientific dataset repository available online which at the time of this writng maintain about 335 datasets. It is described as a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms [7]. It offers a search engine where users can search for the desired dataset, access its description and download the raw data.

Another solution for data sharing is the *Open Science Data Cloud* or simply *OSDC*, which provides virtual machines to store, share and analyse scientific datasets that can reach sizes in the order of tera or even petabytes. With this approach, researchers have the opportunity to customize their machines with whatever tools necessary to analyse their data, and perform the analysis to answer their research questions [8].

Users can apply for an *OSDC* account and depending of the researcher's organization and the evaluation made over the application by the allocation committee, users can get access to the "main OSDC console" and from this, manage their access keys and virtual machines. It is not a very easy task, as it requires some advanced informatics skills. Still it is a valid

[3] Machine Learning Repository, http://archive.ics.uci.edu/ml/ [last accessed 04/03/2015]

approach. Once the machines are ready and the data is published, users can access and download large volumes of data through high-speed Internet connections, *StarLight*[4]/*Internet2*[5].

Data.gov[6], is another data sharing platform. It was created by the U.S. General Services Administration and focuses only on governmental data of United States of America with the goal of making government more open and accountable [9]. However, Data.gov does not host data directly. Instead, it just hosts metadata about public datasets in order to facilitate the process of searching for public governmental datasets, providing as well some references about where users can find the correspondent data, most of the time over CSV files or similar.

dash, DATA SHARNING MADE EASY[7] is another data sharing service where researchers can describe, upload, and share their research data in only four steps so data users can read it and download. For data users, these can search the available datasets on the platform and visualize certain details like authors, abstract or metadata and download the data. However, users cannot visualize the data before downloading it.

BigML[8] is a tool that, although it is not focused on data sharing, has features that are very close to our vision. In fact, it is the solution that best matches our vision, but its focus is on Machine Learning and prediction engines instead of data publishing and sharing.

The features that are very close to our vision are the manner how data is uploaded and manipulated on the platform. BigML let's their users upload all the desired data identifying it as "Data sources". Once the data is uploaded, users (i.e., data providers) can configure subsets of data, selecting which data sources and respective fields shall be available for publication and posterior analysis.

This kind of features shall be very similar to the main tasks for dataset providers on OpenDataHub and therefore, deserve special attention. We can use this solution as a

[4] StarLight, http://www.startap.net/starlight/ABOUT/ [last accessed 15/03/2015]
[5] Internet2, http://www.internet2.edu/about-us/ [last accessed 15/03/2015]
[6] DATA.GOV, http://www.data.gov/ [last accessed 21/03/2015]
[7] dash, https://dash.cdlib.org/ [last accessed 21/03/2015]
[8] BigML, https://bigml.com/how_it_works [last accessed 24/03/2015]

guideline for evaluating the best way to interact with data providers on their data management.

There are several other tools that we could have mentioned and provide a more in-depth description but ultimately all of them will share the same main issues. From these issues we highlight the *inability to query, preview and download data for posterior analysis*. Data users have always to download the data in some specific format and then evaluate if the raw data is what they expected, in order to be able to apply their analysis methods. We are dealing with large amounts of data that can take long periods of time to conclude the download and all this time can be put to waste in case the data does not serve the final user purpose.

OpenDataHub platform aims to provide researches the possibility of sharing their data and data users the ability to query, manipulate and have a preview of the raw data before downloading it. There are several challenges in this vision but we believe that the biggest is the capability of interacting with large volumes of data. In the next sub-chapter we aim to explore in more detail strategies to address our vision's challenges.

2.2 Technological background

In this section we explore and provide a brief description of the different technologies that we find as being potential solutions to our vision needs. We will focus on the main advantages and disadvantages of each one and provide a comprehensive comparison between them.

This section is divided in three main sub-sections. Firstly, we present the database models and management systems, which will help us in deciding which database technology shall we implement in our solution.

Secondly, in the server side technologies sub-section we given an overview of some of the most known and used technologies on server side development and evaluate which one can best serve our purpose.

Lastly, in the client side technologies sub-sections, we explore some of the existing technologies for web development. This time, focusing on the client side development and how these will enable us to build a graphical interface for our users.

2.2.1 SQL, NoSQL and different Database Models

The need for persistent data emerged since the beginning of computer science as most informatics systems need hardware and software to keep information stored to be used in the future. Along the technology evolution, the applications' need for storage resources and techniques has increased whether being desktop, mobile or web-based solutions. The need to process, record and retrieve data are examples of the most common features that applications share [10]. According to these needs, several ways to store information on a computer were developed and tested for different use cases and environments but the most common and known one is the use of databases.

Databases are collections of information that enable an easy access and management of data records. It is an organized collection of interrelated (persistent) data, organized to model aspects of the *real world* in a way that supports processes requiring information. Databases can be managed through software applications known as DBMS (database management system), computer software applications that interact with the user, other applications, and the database itself to capture and analyse data [11]. It offers a higher-level software to interact with lower-level application programming interfaces (APIs), that take care of read and write operations [10].

Over the time different database needs emerged along with different DBMS approaches and applications implementing them. In the remaining sections, we present a brief description of these and what distinguish them.

2.2.1.1 Database Management Systems

Database Management Systems can be viewed as a wide range of different tools for data access and manipulation implementing different approaches based on different data models and structures.

Different needs have emerged, in which data can assume different shapes and sizes, this lead to the development of different DBMS solutions and even more database applications. Yet, despite the emergence of different solutions over the years, only some of them became popular and commonly used. Among them, probably the most predominant one in the past decades are the Relational Database Management Systems (RDBMS) [10].

2.2.1.2 Database Models

Database models are data models that define the logical structure of a database, how shall data be stored and manipulated. It is a collection of concepts and rules that describe the database structure, such as data types, constraints and relationships among different pieces of information [12]. Every database application adopts a database model, defining the logical structure of the data.

The data model is the biggest determiner of how a database application will work and handle the information it deals with. There are several different database models offering different logical data structures, from which the relational model (RM) clearly stands out as the most used data model in the last few decades.

Relational model and relational databases, despite being powerful, flexible and the most known and used solutions, have several issues or features that never have been over crossed or provided. Consequently, recently a series of new and different systems called NoSQL (Not only SQL) have emerged immediately gained popularity, with the purpose of overcoming some of the barriers imposed by MySQL. In the remaining section we can which barriers are these and how NoSQL comes to solve them.

NoSQL aims to offer a much more freely shaped way of working with information, providing more flexibility and ease data management. NoSQL systems are known for its schema-less data approach that, unlike the relational model, can handle data with not very well defined structures, supporting structures that are or can become heterogeneous. It has its pros and cons, considering the important and indispensable nature of data.

2.2.1.2.1 The Relational Model

Introduced in 1970s, the relational model offers a very mathematically-adapted way of structuring, keeping and using data. In this model data is represented in terms of tuples, grouped into relations.

Databases based the relational model, known as relational databases, are mainly constituted by entities, attributes and relations with different cardinality. Altogether, this brings the capability to group different sets of data along different collections through its relation fields, thus enabling the structuration of information.

Decades of research and development in the field database systems that implement the relational model guided us to solutions increasingly efficient and reliable. Combined with the long experience of programmers and database administrators working with these tools, using relational database applications has become the de-facto choice for applications that can not afford loss of any information, in any situation [10].

Furthermore, despite the strict nature of handling data, there are several techniques to achieve extremely flexible solutions. One of such solutions is data normalization that provides a sequence of steps to develop a successful schema design. Non-normalization can lead the database to a set of problems, like inaccurate data, poor performance, and inefficiency and may produce data that one does not expect [10].

Data normalization consists of dividing the data into logical groups that become part of a whole, minimizing redundancy into the database and distributing data so it can be modified in a single point across the database. This enables data access and manipulation in a fast and efficient manner without compromising the integrity of the rest of the system [10].

2.2.1.2.2 The Model-less (NoSQL) Approach

NoSQL databases, unlike the relational model, provide an unstructured approach that aims at eliminating the limitations of having strict relations [10].

Due the scale and agility challenges that modern applications face, NoSQL databases have become the first alternative to relational databases, being scalability, availability, and fault tolerance the key deciding factors in satisfying the user needs. Likewise, another important factor in favour of NoSQL databases is the cheap storage and processing power available today [13] [14]. As such, several NoSQL database technologies were developed in recent years in response to the rising need for large data storage, access frequency and greater performance and processing needs.

Flexibility, schema-less model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology [13]. There are several NoSQL database types, which are briefly described next.

2.2.1.2.2.1 NoSQL database types

There are several different NoSQL databases, each one serving different purposes. For instance, if your application needs a Graph database, a simple *{ key: value }* will not address the application needs. Here we provide a brief description of the different NoSQL database types and its purposes.

Document databases, databases that store data in a *{ key: value }* structure known as document, where "value" can be a complex data structure. Documents can be structures as key-value, or key-array, or even nested documents [14].

Graph stores, databases that are used to store "network information", such as social connections for instance [14].

Key-value, similar to documents, is the simplest NoSQL database where data is stored in key-value pairs and value only sustains primitive data types [14].

Wide column stores, also called extensible record stores, store data in records with the ability to hold a very large amount of dynamic columns. Since the column names as well as the record keys are not fixed, and a single record can have billions of columns, wide column stores can be seen as two-dimensional key-value stores [15].

2.2.1.3 Popular Database Management Systems

In the remaining topics we will explore a little of the most popular database management systems. Although there are many DBMSes and data models, we will rely over the Relational and NoSQL databases.

As of today, it is safe to say that the Relational Model is the most popular data model in DBMS thanks to its simple data modelling abstraction. The relational model can easily map real-world problems in almost 90% of the cases [10]. On the other hand, NoSQL databases come to address some challenges created by modern applications for which relational databases were not designed for, like for example data scalability and agility.

2.2.1.3.1 Relational Database Management Systems

RDBMSes, are relational database management systems that implement the relational model. These systems are, and should be for the next years, the most popular choice of keeping data reliably and safe [10].

RDBMSes require a well defined and clearly set data structures in order to manage data. These structures define how the data will be stored and used. Data structures or schemas in the relational model are defined as tables, with columns representing the number and the type of information that belongs to each record [10].

There are several databases that are considered to be part of the most popular group. On the relational model we have: [10] [16]

- **PostgreSQL**[9]: The most advanced, SQL-compliant and open-source objective-RDBMS.
- **MySQL**[10]: The most popular and commonly used RDBMS.
- **SQLite**[11]: A very powerful, embedded relational database management system.

[9] PostgreSQL, www.postgresql.org/ [last accessed 07/04/2015]

[10] MySQL, www.mysql.com/ [last accessed 07/04/2015]

[11] SQLite, www.sqlite.org [last accessed 08/04/2015]

- **MariaDB**[12]: Has an enhanced query optimizer and other performance-related improvements, which give the database system a noticeable edge in overall performance compared to MySQL.

2.2.1.3.2 NoSQL Database Systems

NoSQL database systems do not apply models like relational solutions do. The schema-less approach opens a wide diversity of manners to store and manipulate data. Consequently, there are several different implementations of these database systems, each one serving different purposes, working differently and defining their own query system.

It is possible to group collections of data together with certain NoSQL databases, such as the MongoDB. These document keep each data item, together, as a single collection (i.e. document) in the database, and can be represented as singular data objects, similar to JSON [10].

The most popular NoSQL implementations are:

- **CouchDB**[13]: Uses JSON for documents, JavaScript for MapReduce indexes, and regular HTTP for its API.
- **Cassandra**[14]: Store and process vast quantities of data in a storage layer that scales linearly.
- **MongoDB**[15]: A document database that provides high performance, high availability, and easy scalability.
- **Hadoop**[16]: A framework built to for running applications on large cluster built. It implements Map/Reduce paradigm, dividing applications into many small fragments of work, enabling the execution or re-execution of each fragment on any node in the cluster [17].

[12] MariaDB, https://mariadb.org/ [last accessed 13/06/2015]
[13] CouchDB, http://couchdb.apache.org/ [last accessed 14/04/2015]
[14] Cassandra, http://cassandra.apache.org/ [last accessed 12/04/2015]
[15] MongoDB, https://www.mongodb.org/ [last accessed 19/04/2015]
[16] MySQL Administration Guide, https://www.novell.com/documentation/nw65/web_mysql_nw/data/bookinfo.html [last accessed 23/04/2015]

2.2.1.4 Comparing SQL and No-SQL DBMSes

There are several points to consider when comparing SQL and NoSQL technologies. In this section, we will give a general overview of different aspects of SQL and NoSQL. Our goal is to build a conclusion about which technology should be selected as the most suited, given the particularities of the proposed “OpenDataHub” web platform.

2.2.1.4.1 Benefits of SQL

SQL has been one of the most common and used solutions for data storage due to some important advantages [18] [19].

- It is easy to use, despite the several database management systems that implement SQL, every single one of these uses the same structured query language (SQL). This builds a common knowledge between all the SQL solutions, easing developers’ work.
- It is secure. Over the years SQL have been improved and tested in the most distinct environments building solid data security layers.
- SQL databases use long-established standards.
- SQL language support the latest object based programming and is highly flexible.

2.2.1.4.2 Benefits of NoSQL

NoSQL emerged in response to some restrictions of relational databases. It aims to offer more scalable solutions, supply superior performance and address some issues that relational data models were not designed for. Some benefits that we can refer about NoSQL are: [14]

- It handles large volumes of structured, semi-structured and non-structured data.
- Enables agile sprints, quick interaction and frequent code pushes. A change to the data structure does not need to change the whole database structure.
- It handles object-oriented programming that is easy to use and flexible.
- It is efficient. Offers a scale-out architecture instead of expensive and monolithic architectures.

2.2.1.4.3 Schema

NoSQL databases are known for their schema-less approach. Its solutions enable the application of agile development processes, i.e., if the application data structure varies very often it is quite easy to sustain the new database schema.

On the other hand, relational databases always need a very well defined schema. Every time a data structure needs to be modified, a new database schema has to be projected and the whole database needs to me migrated to the new solution. This process, depending of the database size, can be very time consuming and can conduct to large periods of down time and it can occur frequently if the development process follows agile methods.

Let's form a practical example of a digital store. There is a need to store users' information such as username, name and address. After some iterations, the application's stakeholders decided that it was an advantage to store the users' most seen items.

In a SQL solution all the requirements should to be addressed at the beginning. The change originated by these iterations would cause a new database schema projection and a migration plan, plus the down time for this to happen.

In a NoSQL solution all the projection and migration process is unnecessary. Thanks to the schema-less approach the development team only needs to store the new information as they wish and all the new records will assume the new solution [10]. Different sets of information with different structures or even with embedded data can be stored and queried very easily.

2.2.1.4.4 Scalability

Relational databases use to scale vertically due to its internal structure. A single server hosts the whole database, which limits the scalability and can come expensive very quickly, since every time the database needs more resources the solution passes by buying more hardware and more powerful equipment.

An horizontal distribution of resources is an approach that can address these problems, adding more servers sustaining the database instead of having a single super server. This database distribution across multiple servers is called “*sharding*” and can be achieved both by SQL and NoSQL technologies.

For relational databases, this process usually involves complex arrangements to make multiple hardware act as a single machine. Since the database does not provide this ability natively every database in each server needs to be handled as a standalone database. All the deployment process has to be replicated across the different databases and the development team has to implement solutions to store and handle data on each database autonomously.

It requires applications to offer mechanisms to distribute data, distribute queries and aggregate data from all the different database instances. Furthermore, resource failures need to be handled, *join* statements between different databases are very likely and data *rebalancing*, *replication* and other requirements are very hard to address. Plus, the application on this manual *sharding* can cost in the relational database main benefits like the transactional integrity.

On the other hand, NoSQL databases usually support *auto-sharding*. That is, NoSQL databases natively offer mechanisms that automatically spread data along the different database instances present on the different servers in a process that is transparent to the application. This approach eliminates problems related to the *replication* of deployment tasks and data, which is automatically balanced and queried across the servers and, on down time occasions, it can be quickly and transparently replaced without application disruption.

Data *sharding* in NoSQL databases can be easily addressed by some database and hosting configuration but it can be avoided as well. Thanks to the *auto-sharding* support, all the concerns related to *sharding* tasks are removed from the development team, avoiding the development of complex and expensive solutions to support their applications [10]. The database and some additional resources (like routers) will be able to handle all the *sharding* tasks by itself, turning all these transparent to the application.

2.2.1.4.5 Data integrity

2.2.1.4.5.1 ACID compliance (Atomicity, Consistency, Isolation, Durability)

Most of SQL databases are ACID[17] compliant, transactions on SQL are one or more actions over the database that can affect several different entities and are defined as a single operation. This come to address atomicity, consistency, isolation and durability.

An atomic transaction means that all the operations that constitute the transaction are executed or none of them are, hence leading to database consistency where the database always will be in a consistent state.

Transactions are isolated operations and each one is self-contained, i.e., its internal state can only be visible to itself and invisible to the rest of the system.

And finally durability, each time a transaction is committed all its internal operations shall persist even on system failure. This is ensured by log mechanisms that enable state recovery [20].

On the other hand, many NoSQL databases are not ACID compliant, sacrificing this in order to achieve best performance and scalability. After all, today's focus lays on highly available distributed computing and the possibility of replicating changes over different servers distributed along different geographic locations [21].

CAP[18] is a mathematical theorem, which states that it is impossible for a distributed system to provide a guarantee of Consistency, Availability and Partition tolerance all at the same time. Following this order of ideas we can note that ACID rules are barriers to NoSQL goals.

With this, a new concept named BASE emerged [21].

[17] ACID, https://pt.wikipedia.org/wiki/ACID [last accessed 04/05/2015]
[18] CAP, https://en.wikipedia.org/wiki/CAP_theorem [last accessed 04/05/2015]

2.2.1.4.5.2 BASE Compliance (Eventual Consistency)

BASE means Basically Available, Soft State, Eventual consistency and is a concept shaped to distributed systems to handle consistency. Eventual consistency means that at some point in time, all the data sources will be synchronized. That can take more time than a SQL database would but it will happen sooner or later over all database *replications* [21].

2.2.1.4.6 Summary

Despite all the points discussed above are important aspects to evaluate which technology can be elected as the best one, it makes very little sense to compare SQL and NoSQL without taking in consideration the type of problem we are solving.

Overall, depending on the different application needs, some aspects may become more important than other. Thus the best database technology for a given application will be the one that ultimately provides the best balance between pros and cons taking into account the application's necessities.

2.2.2 Server side technologies

In web development, when a new application emerges both server and client sides' technologies are a major concern as this decision will end up affect the whole application life-cycle In the sections below we will give an overview of some of the most known programming languages that are used on the server side when building applications based on the cloud.

2.2.2.1 Java

Java is a server-side programming language that is very well known between web and mobile developers, since it is also the core language for building Android mobile applications.

Java offers a wide variety of tools known as "The Java Platform", an open source development environment that includes libraries, frameworks, APIs, the JRE (Java Runtime Environment), Java plug-ins and the JVM (Java Virtual Machine) [22]. This environment offers all the resources needed to develop portable Java solutions, since JVM converts Java from source code into machine code.

There are several good reasons to develop with Java. Its frameworks can outperform other languages and frameworks like Ruby on Rails in terms of speed. It is simple to program, offering a simple interface for users and programmers. It Is platform independent thanks to the JVM and its “write once, run everywhere” philosophy [22].

On the other hand, there are several restrictions that are important to mention. It consumes higher rates of physical resources since it is a high level programming language that offers many resources that can be unnecessary to the implemented solution.

Java as a server-side programming language tries to solve all the problems in the world, which is not a requirement for every developer, which brings unnecessary overhead [23].

2.2.2.2 Python

Python, similar to java, is a high level, free, object-oriented programming language. It is very well known for its clear syntax and readability, being easy to learn and portable (i.e., its statements can be interpreted in a number of operating systems [24][25]).

It is useful for a range of application types, including Web development, scientific computing and education but it has its issues, being performance one of them.

There are several advantages of using python. Pierre Carbonnelle, a Python programmer says that “The main characteristics of a Python program is that it is easy to read”, defending that this enables programmers to think more clearly when writing their applications and it helps in its maintenance and improvement by other programmers in the future, enhancing development production levels [26].

Python has become popular in the Internet of things (IoT). Raspberry Pi's documentation, for instance, cites python as "a wonderful and powerful programming language that is easy to use (easy to read and write) and with Raspberry Pi lets you connect your project to the real world." [26].

Multi-paradigm approach is another feature that distinguishes python from others languages as it supports object-oriented, procedural and functional programming styles [26].

Python is asynchronous, using a single event loop to manage small units of a major task. Stephen Deibel, co-founder of Wingware a python's IDE, says that "Python's generators are a great way to interleave running many processing loops in this approach." [26].

However as with every other programming language, it has its pros and cons. Python is an interpreted language meaning that python is slower than compiled technologies in most of the cases. Mobile development is not an option for python as well. Python can be used for server and desktop platforms but is mostly absent when it comes to client side solutions, like browser and mobile applications [26].

Design restriction is another disadvantage of python. Python's community cite several issues with the design of the language. Python is single thread, meaning that its internals can only be acceded for one thread at the time, that is why is important to produce asynchronous solutions or use the multiprocessing module [26].

2.2.2.3 PHP

PHP is a widely-used open source programming language especially suited for web development, created for server side scripting [27].

It has its pros and cons like every other programming language. It is free and open-source, relatively easy to learn and flexible. It also supports communication with a range of database types, which is an important feature when dealing with applications that need to communicate with databases.

Thanks to its open-source approach, it has a large community contributing to its continuously improvement. It has a very strong and complete documentation created by PHP developers describing all its functions and demonstrating how to use them, which makes PHP a very easy to learn language.

Object oriented programming is currently a well known and much used programming convention and is fully supported by PHP. Beside this, PHP development can also be accomplished without the OO approach, which makes it a very flexible technology.

On the other hand, PHP is ineffective at producing desktop applications and is usually slower than other languages since it is an interpreted language. Another disadvantage is it has poor support for error handling, which contributes for its slowness [28].

2.2.2.4 JavaScript

JavaScript (JS) has been for a long time mostly known for its capability to enhance and manipulate web pages and client browsers. Yet lately it has become more and more prominent as a server side programming language (SSJS).

It is an object-oriented dynamic language with types and operators, standard built-in objects, and methods. Although it is an OOP language it has no classes, instead it has object prototypes. Another main difference between JavaScript and common OOP languages is that functions in JS are like any other object that can be passed around like a regular variable. It is fast and convenient. Besides, JavaScript is one of the fastest or even the fastest dynamic language. Most web developers know JS and the opportunity to use the same programming language on both client and server sides turn JS one of the best choices for web programming.

When talking about Server Side Java Script (SSJS) it is inevitable to talk about Node.JS, a JavaScript runtime built on Google Chrome's V8 JavaScript engine. It uses event-driven, non-blocking I/O model turning it into a lightweight and efficient solution.

V8 is a high performance engine written in C++, built in order to overcome some performance issues when developing bigger JS applications. It compiles and executes JavaScript source code, handles memory allocation for objects and garbage collects objects that are no longer needed [29].

JavaScript is single thread, which means that time-consuming operations would freeze the entire application. The non-blocking I/O model solves the problems that could occur from this processing model and it can be achieved in several ways. Yet the easiest one is probably the event loop and the use of callbacks. While the event loop is always listening for new events, callbacks enable to put time-consuming tasks off to the side, usually by specifying what should be done when these tasks are complete, thus allowing the processor to handle other requests in the meantime [30].

2.2.2.5 ECMAScript 6 (ES2015)

ES2015 is the new way to write JavaScript. It is the newest version of ECMAScript standard. It is a significant update to the language, and the first major update to the language since ES5 was standardized in 2009. It introduces new concepts to the JavaScript world like classes and subclasses, block-scoped binding constructs (i.e., block hoisting), arrow functions, template strings, and many other features.

Although all these features and the acceptance of this new way to write JS by the community, it is not yet supported natively by the most of the browsers. This way, some alternatives have to be explored in order to overcome this limitation. One of them is Babel[19] JavaScript compiler, which enable the compilation of ES2015 into ES5 in such that any browser it able to interpret the code.

2.2.2.6 Non blocking operation vs multi threaded request response

Traditional web application used to follow the “Multi-Threaded Request-Response” or simply “Multi-Threaded Request-Response”. This model is constituted by the following main steps [31]:

1. Clients Send request to Web Server.
2. Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
3. Web Server is in Infinite Loop and waiting for Client Incoming Requests.
4. Web Server receives those requests.

[19] Babel, https://babeljs.io/ [last accessed 12/01/2016]

a. Web Server picks one Client Request.
b. Web Server picks one Thread from Thread pool.
c. Web Server assigns this Thread to Client Request.
d. This Thread will take care of reading Client request, processing Client request, performing any Blocking IO Operations (if required) and preparing Response.
e. This Thread sends prepared response back to the Web Server.
f. Web Server in-turn sends this response to the respective Client.

In this approach the server is always listening for new requests in infinite loop and performs all the previous steps every time a new request is received. This means that this model creates one thread per client request which can lead to high rates of consumption of server's physical resources [31].

If many client requests happen in quick succession requiring more blocking IO operations, this will consume almost all the threads. Each thread will then be busy preparing the response for its request and the remaining client requests will have to wait longer time before they are handled by the server.

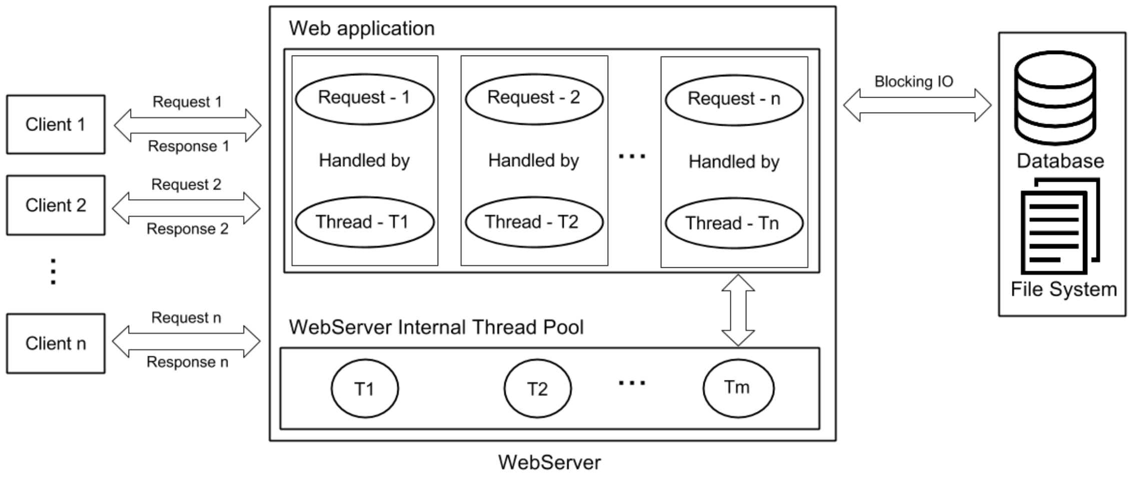


Figure 2.1 –Request/Response Model [31]

Figure 2.1 is a graphical representation of the Request/Response model on a server that takes requests from **n** clients and has **m** threads available on the thread pool, picking up a **T** thread for each request, where any request can require blocking IO operations such as iteration with database or the file system [31].

If **n** is greater than **m** (which is very often), many requests should wait in the Queue until some of the busy Threads finish their Request-Processing Job and become free to pick the next Request [31].

Instead, Node.JS uses Single-Threaded Event Loop Model that is constituted by the following main processing steps [31]:

- Clients Send request to Web Server.
- Node.JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
- Node.JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.
- Node.JS Web Server internally has a Component, known as “Event Loop”. It uses indefinite loop to receive requests and process them. (See some Java Pseudo code to understand this below).
- Event Loop uses Single Thread only. It is main heart of Node.JS Platform Processing Model.
- Even Loop checks if any Client Request is placed in Event Queue. If no, then wait for incoming requests for indefinitely.
- If yes, then pick one Client Request from Event Queue
    - Starts process that Client Request
    - If that Client Request Does Not requires any Blocking IO Operations, then process everything, prepare response and send it back to client.
    - If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services then it will follow different approach
        - Checks Threads availability from Internal Thread Pool
        - Picks up one Thread and assign this Client Request to that thread.
        - That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop

- Event Loop in turn, sends that Response to the respective Client.

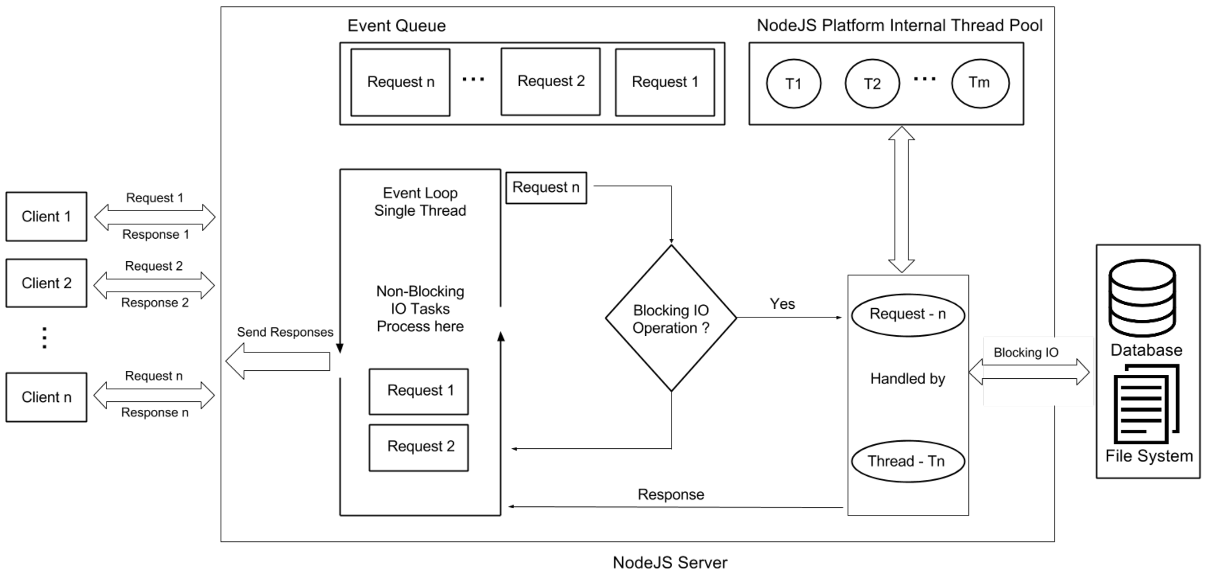


Figure 2.2 – Single-Threaded Event Loop Model

Figure 2.2 is a graphical representation of the Single-Threaded Event Loop Model on a server taking requests from **n** clients and has **m** threads available on the thread pool. This time there is no need to afford a thread per request. Instead, the event loop evaluates if the request requires blocking or time-consuming tasks.

If not, the request is processed by the main thread and the response given to the client right way. If yes, the event loop picks a thread from the thread pool responsible for read and processes the request. The thread then performs the blocking operations and prepares the response before sending it to the event loop that is responsible for delivering the response to the original request [31].

If **n** is greater than **m** (which happens very often), it is not necessarily a problem. Since much of the requests should not require blocking operations, many of of them would not need a thread form the thread pool to get processed. Thanks to this feature, Node.JS can handle many more requests with fewer resources than a server with the Request/Response model.

Another motivation to Node.JS adoption is its complicity with NoSQL databases, MongoDB specially. Node.JS offers a driver to interact with MongoDB databases, and since it is a document based database, all the data managed between the database and the Node application is handled through JSONs (JavaScript Object Notation), enabling the development of homogeneous solutions avoiding the need to transform data in the form of objects, maps, arrays and others into JSON.

The Node.js driver can be used on its own, but it also serves as the basis for several object-mapping libraries, such as Mongoose.

2.2.3 Client side technologies

Every web application has to interact with its final users somehow, be it a mobile interface, a browser or some other mean. In this subsection we will look at client technologies supported by web browsers that will be the default mean of interaction in OpenDataHub.

When talking about web development and browser interfaces, it is indispensable to refer to technologies such as HTML, CSS and Javascript. These technologies are known since the beginning of Internet and therefore do not require big introductions.

HTML is a standard markup language that enables the web page creation, consisting of structured document that implement building blocks, media components and interactive forms. All this is interpreted by the browser and originates visual and audio interfaces with the user [32]. But a well structured purely HTML page might not be sufficient to provide the best experience for the user. CSS and JavaScript offer different mechanisms to improve the user interaction.

CSS is a style sheet language used to describe the presentation of a HTML page. It enables developers to create visually engaging webpages, user interfaces for web applications as well as user interfaces for many mobile applications [33].

JavaScript, as we have already seem above, is a scripting language widely used on web pages to enable the creation of dynamic contents thus promoting more pleasant interactions and the development of more efficient client side solutions [34].

JavaScript is known as an interpreted language but for Google Chrome users this is not true. Google Chrome’s V8 engine compiles JavaScript into native machine code before executing it, introducing some additional optimization in the process promoting a faster execution and a smoother interaction with the user.

Nevertheless, there are many other client side technologies that complement these three. In the remaining section we will give an overview of some of the most interesting and promising technologies nowadays.

2.2.3.1 Less and Sass

Less is a CSS pre-processor, i.e., it extends the CSS language and it aims at turning CSS more maintainable, theamable and extendable. The way it addresses this is through the introduction of *mixins*, variables, nesting, inheritance, *namespacing*, scoping, functions and many other techniques to the world of CSS.

Similar to Less there is Sass, another CSS pre-processor that implements all the above techniques. Sass is well known and has been target of distinction thanks to its faster compiling times and for offering Compass, a Sass framework designed to make the work of styling interfaces smoother and more efficient.

2.2.3.2 Bootstrap

Bootstrap[20] is a HTML, CSS and JS framework for web and mobile developments. Until the last stable release, its source code uses Less pre-processor, making it very customizable.

Bootstrap is as of today one of the most well-known and used web technologies. It offers many different CSS and JS resources that promote a faster and cleaner development turning responsive design really easy to accomplish thanks to its grid system.

[20] Bootstrap, http://getbootstrap.com/ [last accessed 17/08/2015]

2.2.3.3 jQuery and jQuery UI

jQuery[21] is a JavaScript library that makes JavaScript easier to implement and read. Through its API, DOM manipulation, event handling, animation and Ajax[22] request are much simpler [35].

Complementing jQuery, there is jQuery UI. A curated set of user interface interactions, effects, widgets, and themes create on top of the jQuery. jQueryUI was built to support development of highly interactive web application.

2.2.3.4 AngularJS and Angular Material

AngularJS[23] is a structural framework for dynamic web apps that has been increasingly used for frontend development. AngularJS has several features that can change the way we structure and develop frontend applications.

It implements the MVC architecture, which leads developers to write components that very clearly separates the logical sections from the graphical interface thus enhancing code readability and reuse. Thanks to this architecture, AngularJS enables developers to easily implement unit tests and provide several other advantages like dependency injection, data binding, form validation, request routing and others.

Angular Material[24], is an implementation of the Google's Material Design Specification. [36] A specification created by the Google team, describing the classic principles of good design with the final goal of developing a single underlying system that allows for a unified experience across platforms and device sizes.

Angular Material comes to implement a set of components and services wrote in AngularJS that implement these principles and enable developers to create graphical

[21] jQuery, https://jquery.com/ [last accessed 24/08/2015]
[22] Ajax, http://www.w3schools.com/ajax/ [last accessed 24/08/2015]
[23] AngularJS, https://angularjs.org/ [last accessed 04/02/2016]
[24] Angular Material Design, https://material.angularjs.org/latest/ [last accessed 04/02/2016]

interfaces that can adapt to both desktop and mobile environments. In very simple terms, Angular Material can be thought of as a jQuery UI for AngularJS applications.

2.3 Summary

Having this background research completed, we have presented a solid base to make our decisions about what our system shall be and how we shall proceed in order to provide and effective implementation of our envisioned OpenDataHub. In the remaining sections we will address some of the topics discussed abovein order to ground our decisions over the project progress.

Chapter 3 Benchmark MySQL vs MongoDB

In this chapter we describe, present and discuss the results from a benchmark produced from SustData dataset between one SQL and a NoSQL database, MySQL and MongoDB respectively.

We start describing the environment for the tests, then we proceed with a description of

3.1 Introduction

SQL and NoSQL have been the subject of several discussion in the past few years regarding which is the best technology for data storage [37]. Yet, the overall conclusion is that the selection of the appropriate technology should be made taking into account the application specifications, and that there is no "one-size-fits-all" approach. Ultimately, choosing the right technology depends of the use case. If data is continuously changing or growing fast and you need to be able to scale it quickly and efficiently, maybe NoSQL is the right choice. On the other hand, if a data structure is well defined, and it will not change much frequently and data does not grow that much then SQL is the best answer.

In this work we aim to go beyond the theoretical guarantees about data storage technologies. To this end, we have decided to perform an extensive benchmark between SQL and NoSQL databases.

This benchmark was done against one public scientific dataset and the several tests were created taking into consideration the dataset needs and the purpose of the actual data. More specifically, this benchmark was done between MongoDB and MySQL, two of the most known technologies on data storage world. Ultimately, at the end of this analysis we will be able to decide which storage technology is a better fit for our proposed dataset management system, the OpenDataHub.

In order to perform fair benchmarks between both technologies, different environments and different sets of data were created so both technologies could be tested in the same exact conditions. Having the environments all set, PHP (for MySQL) and Node.JS (for MongoDB), algorithms were created to perform read and write operations in order to assess the performance of each technology.

Once we gathered all the results, we have developed an extended analysis of the results and identified which technology offers the best conditions for the creation of the OpenDataHub.

3.2 Benchmark Environment

In order to test both technologies under the same conditions two virtual machines were created on an apple iMac machine, with the following specifications:

- OS: OS X Yosemite
- Storage: 1TB Sata disk;
- Memory: 10GB 1067MHz DDR3;
- Processor: 3,06GHz Intel Core 2Duo.

For the VM creation we used Oracle VM VirtualBox[25], a virtualization product that enabled the installation of machines with the following specifications:

- OS: Ubuntu (64 bits);
- Storage: 500GB (virtual);
- Memory: 4096 MB;
- Processor: 2 processors.

Once the VMs were created some additional software was installed in each machine, in order to build and run the PHP and Node.JS algorithms.

[25] VirtualBox, https://www.virtualbox.org/ [last accessed 14/08/2015]

For the “MySQL Machine” XAMPP (Apache, MySQL, PHP, Perl) was installed. XAMPP is a very popular, free and open source cross-platform web server solution stack package, consisting mainly of the Apache HTTP Server, MySQL database,and interpreters for scripts written in the PHP and Perl programming languages [38].

For the “MongoDB Machine”, MongoDB and Node.JS were installed.

Having all the environment set, two REST services were written to record the performance of the write and read operations for both technologies, one using PHP for MySQL and the other one using Node.JS for MongoDB.

3.3 Data Under Test

Large amounts of data were used to evaluate the performance and scalability in big data real problems of both database technologies.

For the benchmark we used SustData dataset, a public scientific dataset related to electricity energy data. It contains five years of electric energy related data collected from four energy monitoring and eco-feedback deployments that were done during the SINAIS project [6].

Overall, SustData contains over 35 million individual records from 50 monitored homes covering electricity consumption logs and demographic information as well as the energy production in Madeira Island. The dataset also contains 3 years worth of environmental data for the island (temperature, cloud coverage, etc.) [5].

For the performance evaluation we used the energy consumption logs, using fifteen CSV files containing different volumes of data for both write and read operations.

Overall, we built a performance curve that simulated the progressive insertions and readings of the different sets of data until a total of ten million records were introduced in both databases. Figure 3.1 presents the different volumes of data in each set.

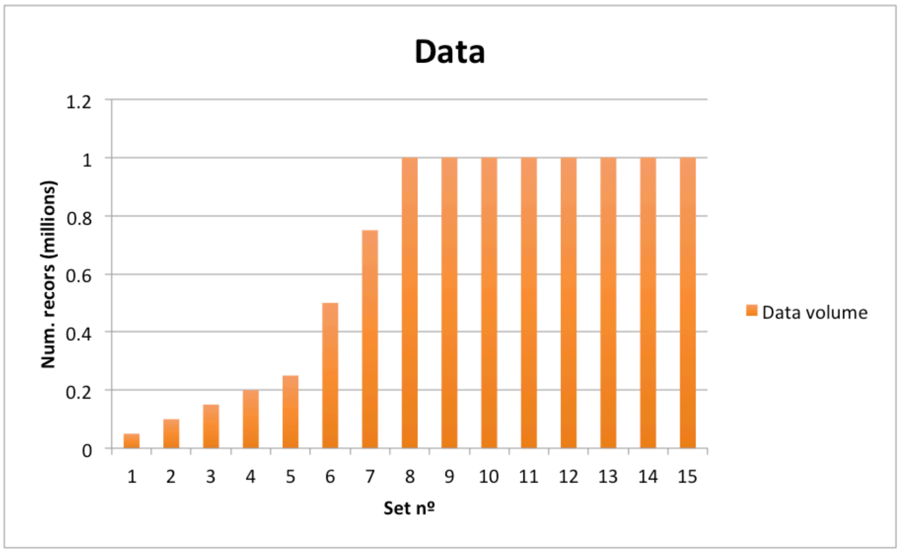


Figure 3.1 – Data

3.3.1 Data Representation and Storage in MySQL

MySQL uses a normalized data structure approach for data storage, but since our main goal is to build and test an environment where read operations will occur more frequently, our major concern is to optimize reads (in public datasets read queries are much more common than the data manipulation counterparts).

In order to achieve this effect, we decided not to normalize the data. Instead we created a schema that supports all the different data structures in a single table.

We are aware that this approach is not the most common and correct for relational databases since this will result in worst performance for data storage sizing and scalability. Yet, this approach has the upper hand on read / write performance since its not necessary introduce time consuming *JOIN* statements in the SQL read queries, and all the data is inserted in a single table.

As mentioned before, SustData emerged from four different deployments each one with different data structures. Consequently, for the MySQL database, a table that contains all the different structures had to be created. The structure of that table is shown in Table 3.1. The different sets of data were then loaded using the SQL query present in Code Block 3.1

Table 3.1 – MySQL Data

Id	**iid**	**tmstp**	**deploy**	**Imin**	**Imax**	**Iavg**	**Vmin**	**Vmax**	**Vavg**
1	1	14/09/11 23:39	2	2.14	2.27	2.23067	237	240	238.424
Pmin	**Pmax**	**Pavg**	**PFmin**	**PFmax**	**PFavg**	**Qmin**	**Qmax**	**Qavg**	**miss_flag**
507.18	542.427	531.607	0.993068	1	0.999552	NULL	NULL	NULL	0

```
LOAD DATA INFILE '".$file_path."'
INTO TABLE power_sample
FIELDS TERMINATED BY ','
(   home_id, timestamp, deploy, Imin, Imax, Iavg,
    Vmin, Vmax, Vavg, Pmin, Pmax, Pavg, Qmin, Qmax,
    Qavg, PFmin, PFmax, PFavg, miss_flag)
```

Code Block 3.1 - Import data MySQL Query

On a normalized fashion, to store this kind of data we would need two additional tables. One storing all the different variables (i.e. I, V, P, PF and Q) and another one holding a many-to-many relation between the Homes table and the variables table, storing MIN, MAX and AVG values as well as the measurement timestamp. Using this approach read operations would have to contain *JOIN* and *Group* statements, which would results in considerably higher reading times.

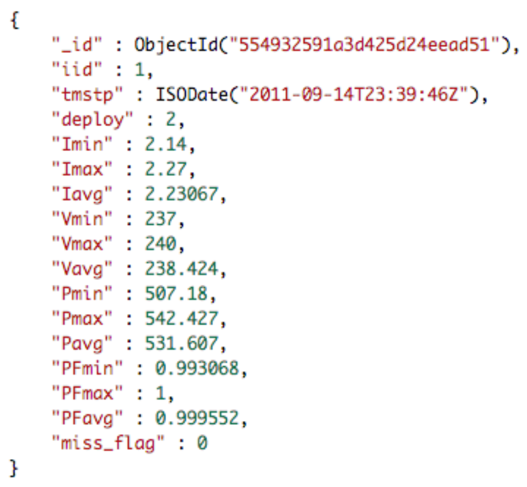
3.3.2 Data Representation and Storage in MongoDB

MongoDB is a schema-less *JSON-style* data storage technology that supports multiple documents with different structures in a single collection. Hereupon, having a single collection with only non-null values for all the dataset is not a problem.

Using *mongoimport* tool and *--ignoreblanks* options set to true (Code Block 3.2) we managed to import all the data producing documents like the one shown on Figure 3.2.

```
mongoimport
    --host localhost
    --port 27017
    --collection power_sample
    --db '+config.db.database+'
    --type csv
    --headerline
    --file '+config.fs.storage_path+file)+'
    --ignoreBlanks
```

Code Block 3.2 - Import data MongoDB



```
{
    "_id" : ObjectId("554932591a3d425d24eead51"),
    "iid" : 1,
    "tmstp" : ISODate("2011-09-14T23:39:46Z"),
    "deploy" : 2,
    "Imin" : 2.14,
    "Imax" : 2.27,
    "Iavg" : 2.23067,
    "Vmin" : 237,
    "Vmax" : 240,
    "Vavg" : 238.424,
    "Pmin" : 507.18,
    "Pmax" : 542.427,
    "Pavg" : 531.607,
    "PFmin" : 0.993068,
    "PFmax" : 1,
    "PFavg" : 0.999552,
    "miss_flag" : 0
}
```

Figure 3.2 –MongoDB Document

3.4 Method

For each REST services we have implemented a benchmarking algorithm with the same logic. PHP with MySQL and Node.JS with MongoDB. The logic of the implemented algorithm is shown in Figure 3.3.

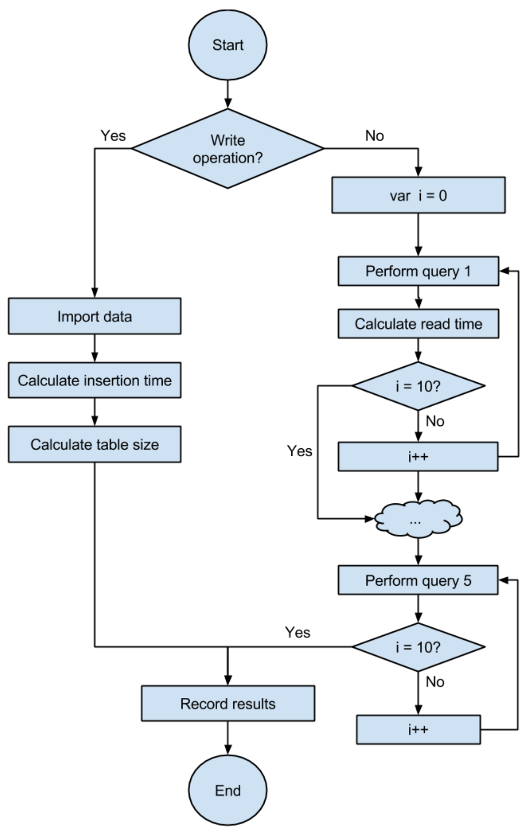


Figure 3.3 –Benchmark flowchart

These two scripts are responsible for two major tasks in our benchmark, namely: 1) insertion of the identified sets of data; and 2) the execution of different queries to measure read and write performance.

Five different aggregation queries were implemented and executed against the data from three different houses. Each query was executed 10 times in order to minimize effects produced by external issues like processor or memory overhead. Ultimately, this resulted in a total of 150 log records for each dataset.

This was then reproduced over the fifteen datasets already presented above (see Figure 3.1), producing a total of 2250 read log records for each technology.

3.5 Data Indexing

Having the data storage strategy set, we have created indexes for the most common fields used among the aggregation queries in order to enhance the read operations performance in both technologies.

Database indexes are an important aspect to consider when looking for fast data access. Yet, there is no standard or pattern to define indexes, as this heavily depends of each use case. Consequently, before defining the indexes we first had to understand what were the most common ways that our data would be accessed. Only then we could create strategies to enhance the performance of the reading operations by means of data indexing.

To this end, we have looked at the most common queries that were performed to the data during SINAIS project and selected the top five. In the next section we present in detail each selected query.

3.6 Queries

All the selected queries performed aggregation operations in order to produce electric energy statistics. Table 3.2 summarizes the data each query is expected to retrieve.

Table 3.2 – Queries

Query 1	Calculates the power average by hour for a specific installation ID.
Query 2	Calculates the power average and the average of the results of the multiplication of current and voltage average values. The obtained results are then grouped by hour for a specific installation ID.
Query 3	Calculates the power average per hour for a specific date and installation ID.
Query 4	Selects all the power averages calculated on SustData during a specific week of the year.
Query 5	Sums power averages per hour during a specific month and for a specific installation ID.

In the sections bellow we present how these queries were implemented in MongoDB and Mysql.

3.6.1 Implementation

In MongoDB we used the aggregation pipeline, a framework for performing aggregation tasks, modelled on the concept of data processing pipelines [39]. The pipeline allows to process data from a collection with a sequence of stage-based manipulations transforming the documents into aggregated results. In MySQL the default *GROUP BY* clause was used for data aggregation. This clause offers several aggregation functions such as average, sum, count and others.

Query 1

For the MongoDB a pipeline with four stages was defined (see Code Block 3.3):

- The first one performs a match operation, selecting all the records for Home 1 and with miss_flag set to zero (miss_flag identifies the records that were added to SustData by means of post-processing (1 for post-process data and 0 otherwise). Thus setting this to zero means that only the original data will be loaded).
- Second stage manipulates documents selecting Pavg, tmstp and decoupling timestamp in year, month, day and hour.
- Stage three, groups the result by hour, calculating the average of pAVG field and the maximum tmstp.
- Finally, fourth stage sorts the results by tmstp in ascending order.

For MySQL a *GROUP BY* clause and *AVG* aggregation function were used in the sequence presented bellow:

- Restrains results to Home 1 and with miss_flag set to zero.
- Group the results by date and hour, calculating the average of pAVG.
- Sort the results by tmstp in an ascendant order.

```
power_sample.aggregate(
    {$match: {iid: 1, miss_flag: 0}},
    {
        $project: {
            y: {$year: '$tmstp'},
            m: {$month: '$tmstp'},
            d: {$dayOfMonth: '$tmstp'},
            h: {$hour: '$tmstp'},
            Pavg: 1,
            tmstp: 1,
        }
    },
    {
        $group: {
            _id: {
                year: '$y',
                month: '$m',
                day: '$d',
                hour: '$h'
            },
            pAVG: { $avg: "$Pavg" },
            tmstp: { $max: "$tmstp" }
        }
    },
    {$sort: {tmstp: 1}}
);
```

```
SELECT
    AVG(pAVG)
FROM power_sample
WHERE home_id = '1' AND miss_flag = 0
GROUP BY DATE(TIMESTAMP), HOUR(TIMESTAMP)
ORDER BY TIMESTAMP ASC
```

Code Block 3.3 - Query 1. MongoDB (left), MySQL (right)

Query 2

For MongoDB a pipeline with four stages was defined (see Code Block 3.4):

- The first one performs a match operation, selecting all the records for Home 1 and with miss_flag set to zero.
- Second stage manipulates documents selecting Pavg, Vavg, pAvgS, tmstp and decoupling timestamp in year, month, day and hour. Important to notice that pAvgS is composed by the multiplication of Iavg and Vavg fields.
- Stage three, groups the result by hour, calculating the average of Pavg and pAvgS fields and the maximum tmstp.
- Finally, fourth stage sorts the results by tmstp in an ascendant order.

On MySQL is used *GROUP BY* clause and *AVG* aggregation function.

- Restrains results to Home 1 and with miss_flag set to zero.
- Multiplies Iavg and Vavg fields originating pAvgS.

- Groups the results by date and hour, calculating the average of pAVG and pAvgS.
- Sorts the results by tmstp in an ascendant order.

```
power_sample.aggregate(
    {$match: {iid: 1, miss_flag: 0}},
    {
        $project: {
            y: {$year: '$tmstp'},
            m: {$month: '$tmstp'},
            d: {$dayOfMonth: '$tmstp'},
            h: {$hour: '$tmstp'},
            Pavg: 1,
            pAvgS: { $multiply: [ "$Iavg", "$Vavg" ] },
            tmstp: 1,
        }
    },
    {
        $group: {
            _id: {
                year: '$y',
                month: '$m',
                day: '$d',
                hour: '$h'
            },
            pAvg: { $avg: "$Pavg" },
            pAvgS: { $avg: "$pAvgS" },
            tmstp: { $max: "$tmstp" }
        }
    },
    {$sort: {tmstp: 1}}
);
```

```
SELECT
     AVG(pAVG)           AS 'AveragePower',
     AVG(iAvg * vAvg)    AS 'AverageS'
FROM power_sample
WHERE home_id = '1' AND miss_flag = 0
GROUP BY HOUR(timestamp), Date(timestamp)
ORDER BY Date(timestamp), Hour(timestamp)
```

Code Block 3.4 - Query 2. MongoDB (left), MySQL (right)

Query 3

For MongoDB a pipeline with five stages was defined:

- The first one performs a match operation, selecting all the records for Home 1 and with miss_flag set to zero.
- Second stage manipulates documents selecting Pavg, Vavg, tmstp and decoupling timestamp in year, month, day and hour.
- Stage three, performs another match operation, selecting all the records which record date is 2010-11-25.
- Stage four, groups the result by hour, calculating the average of Pavg field and the maximum tmstp.
- Finally, fifth stage sorts the results by tmstp in an ascendant order.

For MySQL a *GROUP BY* clause and *AVG* aggregation function were used. It:

- Restrains results to Home 1, miss_flag set to zero and all the records which date is 2010-11-25.
- Groups the results by date and hour, calculating the average of pAVG.
- Sorts the results by tmstp in an ascendant order.

```
power_sample.aggregate(
    {$match: {iid: 1, miss_flag: 0}},
    {
        $project: {
            y: {$year: '$tmstp'},
            m: {$month: '$tmstp'},
            d: {$dayOfMonth: '$tmstp'},
            h: {$hour: '$tmstp'},
            Pavg: 1,
            tmstp: 1,
        }
    },
    {$match: {y: 2010, m: 11, d: 25}},
    {
        $group: {
            _id: {
                year: '$y',
                month: '$m',
                day: '$d',
                hour: '$h'
            },
            pAvg: { $avg: "$Pavg" },
            tmstp: { $max: "$tmstp" }
        }
    },
    {$sort: {tmstp: 1}}
);
```

```
SELECT
    AVG(pAvg)   AS 'Average Power',
    tmstp
FROM power_sample
WHERE home_id = '1'
    AND miss_flag = 0
    AND DATE(TIMESTAMP) = '2010-11-25'
GROUP BY HOUR(timestamp), Date(timestamp)
ORDER BY Date(timestamp), Hour(timestamp)
```

Code Block 3.5 - Query 3. MongoDB (left), MySQL (right)

Query 4

On MongoDB a pipeline with four stages was defined:

- The first one performs a match operation, selecting all the records for Home 1 and with miss_flag set to zero.
- Second stage manipulates documents selecting Pavg, Vavg, tmstp and decoupling timestamp in year, month, and week.
- Stage three, performs another match operation, selecting all the records which record date is 2010-11-25.
- Finally, fourth stage sorts the results by tmstp in an ascendant order.

On MySQL a *GROUP BY* clause and *AVG* aggregation function were used. It:

- Restrains results to Home 1 and with miss_flag set to zero.

- Multiplies Iavg and Vavg fields originating pAvgS.
- Groups the results by date and hour, calculating the average of pAVG and pAvgS.
- Sorts the results by tmstp in an ascendant order.

```
db.power_sample.aggregate(
    {$match: {iid: 1, miss_flag: 0}},
    {
        $project: {
            y: {$year: '$tmstp'},
            m: {$month: '$tmstp'},
            w: {$week: '$tmstp'}
            Pavg: 1,
            tmstp: 1,
        }
    },
    {$match: {w: 37, y: 2010}},
    {$sort: {tmstp: 1}}
);
```

```
SELECT
    pAvg AS 'AveragePower',
    tmstp
FROM summary_hour
WHERE iid = '1'
    AND miss_flag = 0
    AND WEEK(TIMESTAMP) = '37'
    AND YEAR(TIMESTAMP) = '2010'
ORDER BY timestamp asc
```

Code Block 3.6 - Query 4. MongoDB (left), MySQL (right)

Query 5

On MongoDB a pipeline with five stages was defined:

- The first one performs a match operation, selecting all the records for Home 1 and with miss_flag set to zero.
- Second stage manipulates documents selecting Pavg, Vavg, tmstp and decoupling timestamp in year, month, and week.
- Stage three, performs another match operation, selecting all the records which record date is 2010-11-25.
- Finally, fourth stage sorts the results by tmstp in an ascendant order.

On MySQL is used *GROUP BY* clause and *AVG* aggregation function. It:

- Restrains results to Home 1 and with miss_flag set to zero.
- Multiplies Iavg and Vavg fields originating pAvgS.
- Groups the results by date and hour, calculating the average of pAVG and pAvgS.
- Sorts the results by tmstp in an ascendant order.

```
db.power_sample.aggregate(
    {$match: {iid: 1, miss_flag: 0}},
    {
        $project: {
            y: {$year: '$tmstp'},
            m: {$month: '$tmstp'},
            w: {$week: '$tmstp'},
            d: {$dayOfMonth: '$tmstp'},
            h: {$hour: '$tmstp'},
            Pavg: 1,
            tmstp: 1,
        }
    },
    {$match: {y: 2012, m: 3}},
    {
        $group: {
            _id: {
                year: '$y',
                month: '$m',
                day: '$d',
                hour: '$h'
            },
            AveragePower: { $sum: "$Pavg" }
        }
    },
    {$sort: {tmstp: 1}}
);
```

```
SELECT
    SUM(pAVG) AS 'AveragePower',
    timestamp
FROM power_sample
WHERE home_id = '1'
     AND miss_flag = 0
    AND MONTH(TIMESTAMP) = '3'
    AND YEAR(TIMESTAMP) = '2012'
GROUP BY DAY(timestamp), HOUR(timestamp)
ORDER BY timestamp
```

Code Block 3.7 - Query 5. MongoDB (left), MySQL (right)

3.7 Results

Once the recording process was completed, the collected data was catalogued into four different groups in order to create different analyses.

In group I we have an analysis of the database sizes and insertion times. With these two metrics we developed averages and charts representing the time and storage size, relating these with the growing number of records.

Group II represents the performance of every aggregation query performed for each different home ID. This gives us the performance curve related to the data growth and allows a direct comparison of each case between both technologies.

Then, for a briefest review we have group III representing the overall query performance for each home ID. Once again, enabling a direct comparison between MongoDB and MySQL.

Finally, there is group IV revealing the final overall results for both technologies where we can see the global performance for each technology.

In the remaining of this section we explain in detail the results for each of the abovementioned groups.

Group I

From Figure 3.4 we can analyse the insertion time of the different sets of data on MongoDB and MySQL databases. As the chart shows, MySQL has faster *insertion times* than MongoDB.

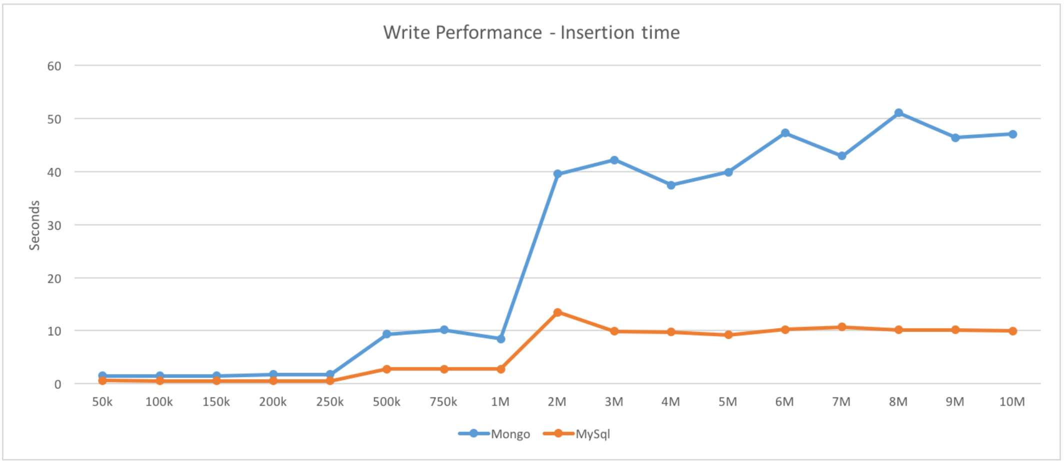


Figure 3.4 –Insertion time

In big data environments storage can lead to big problems. Data is always increasing but storages technologies are not necessarily keeping up, which drives us to measure this important metric on this benchmark.

Once again the analysis is produced over the different sets of data where we can see that MySQL stores the same data in a smaller *storage size*.

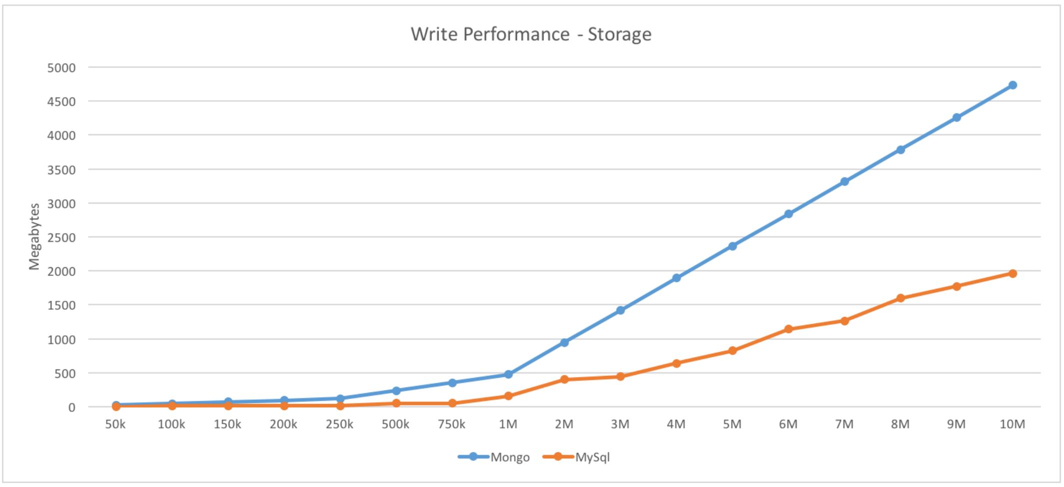


Figure 3.5 – Data storage

Group II

For read operations a more detailed analysis was produced. From Figure 3.6 to Figure 3.20 we have charts representing the average performance and standard deviation for all the 10 iterations, along with the data growth for each home ID. This was done for each aggregation query on both technologies.

Here it is possible to see that in every case MongoDB *queries the data* faster than MySQL.

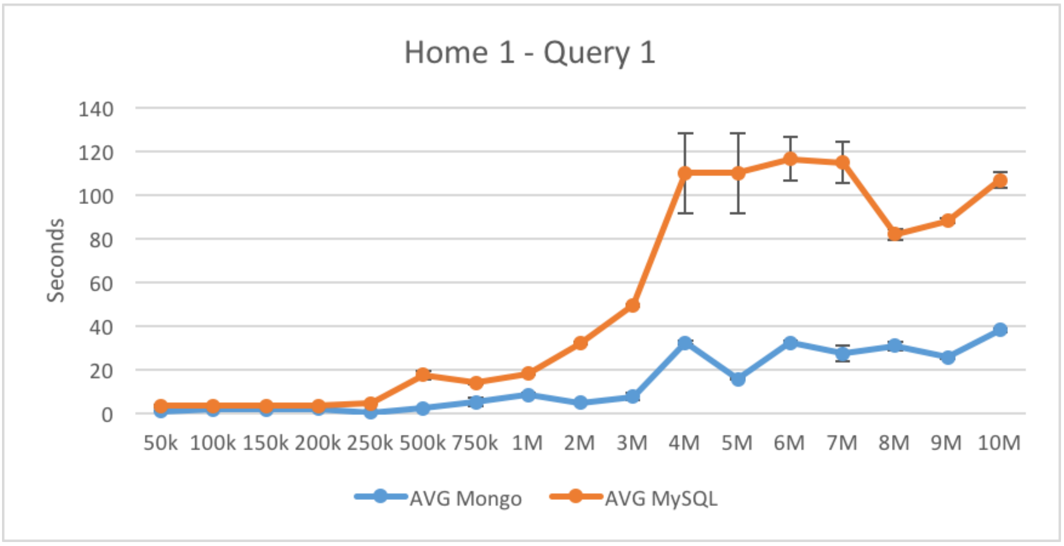


Figure 3.6 – Query 1, Home 1

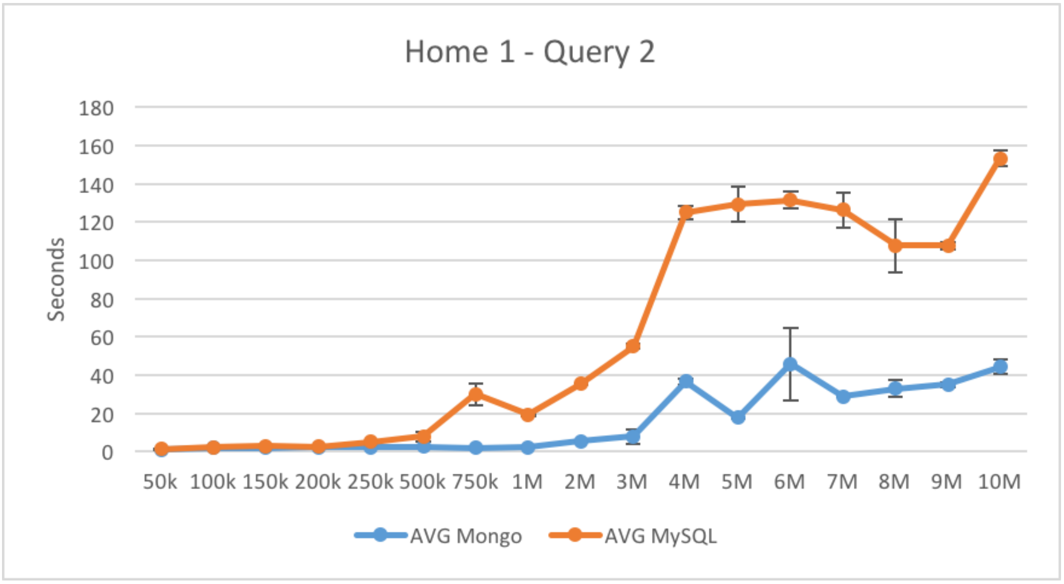


Figure 3.7 – Query 2, Home 1

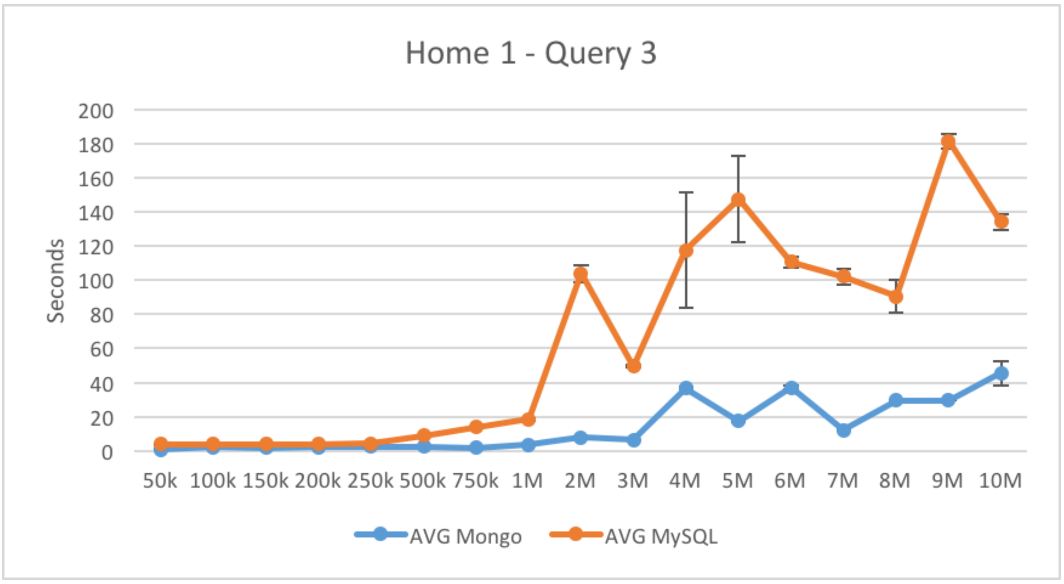


Figure 3.8 – Query 3, Home 1

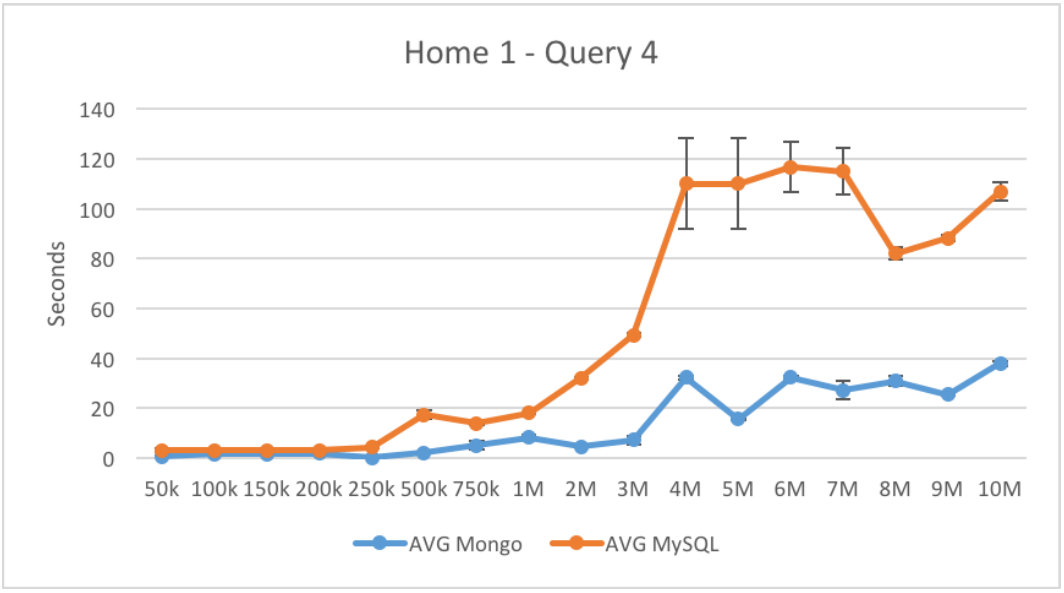


Figure 3.9 – Query 4, Home 1

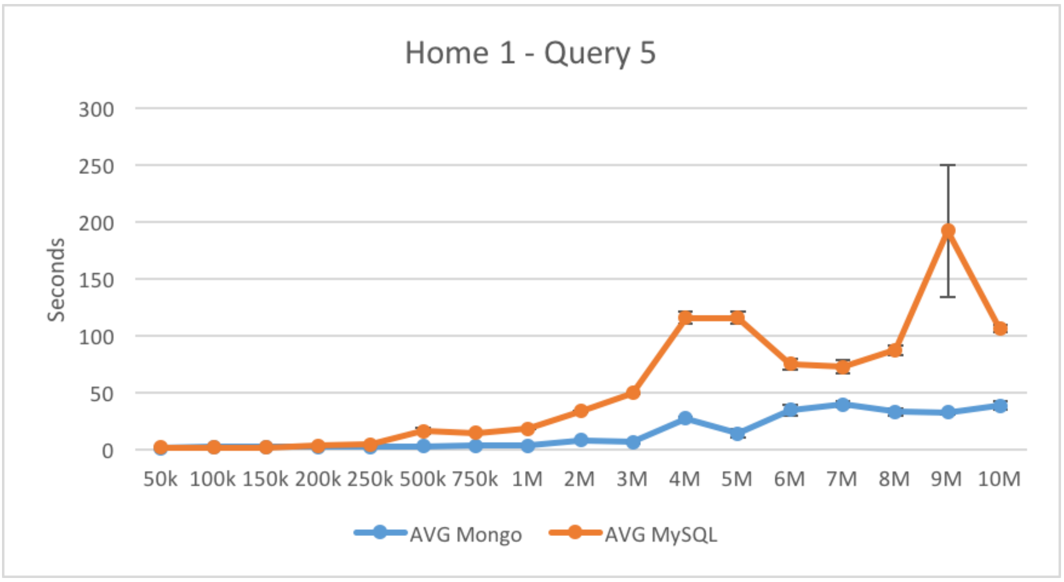


Figure 3.10 – Query 5, Home 1

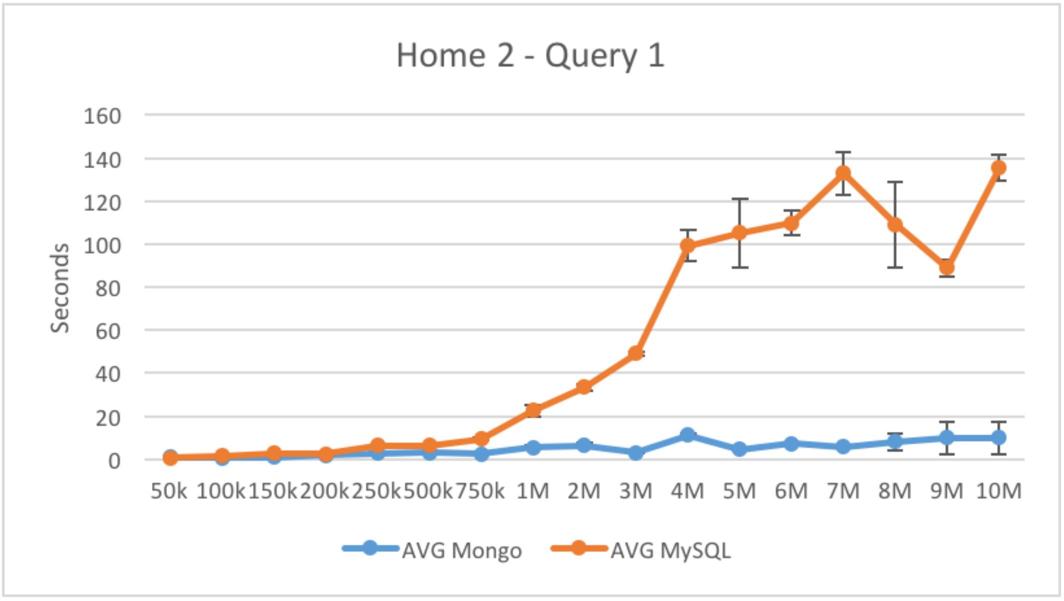


Figure 3.11 – Query 1, Home 2

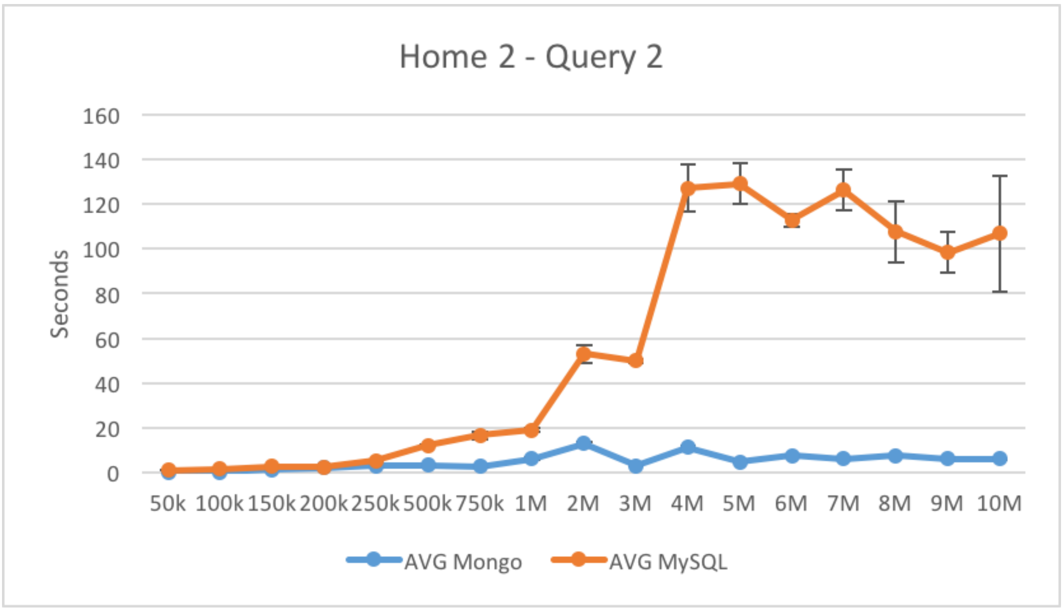


Figure 3.12 – Query 2, Home 2

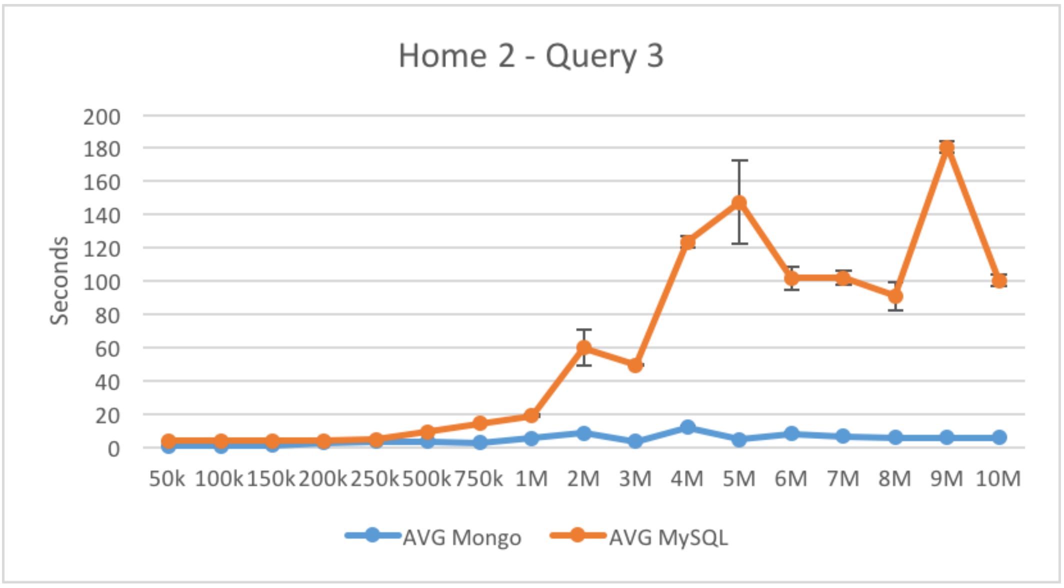


Figure 3.13 – Query 3, Home 2

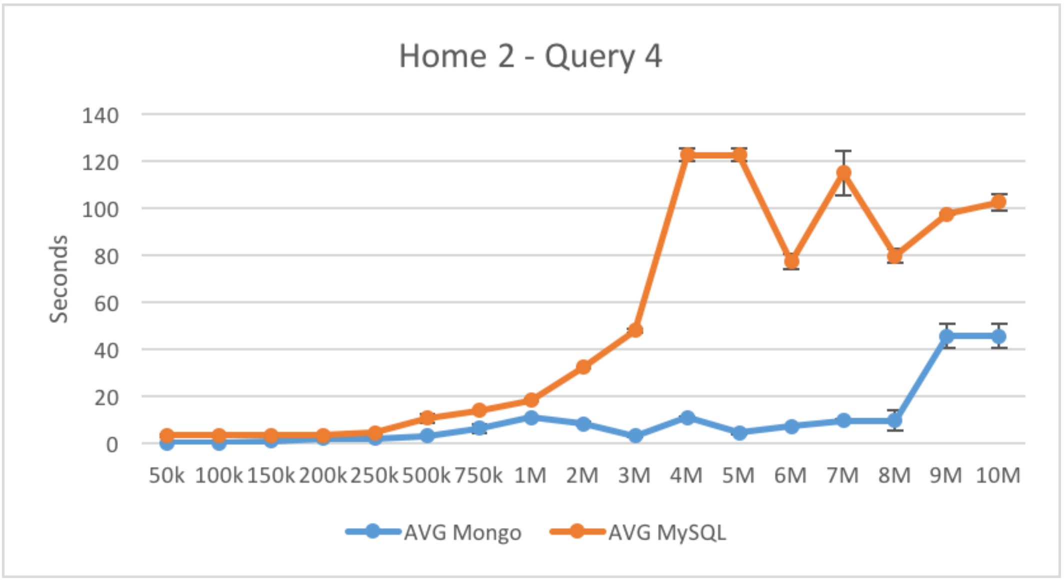


Figure 3.14 – Query 4, Home 2

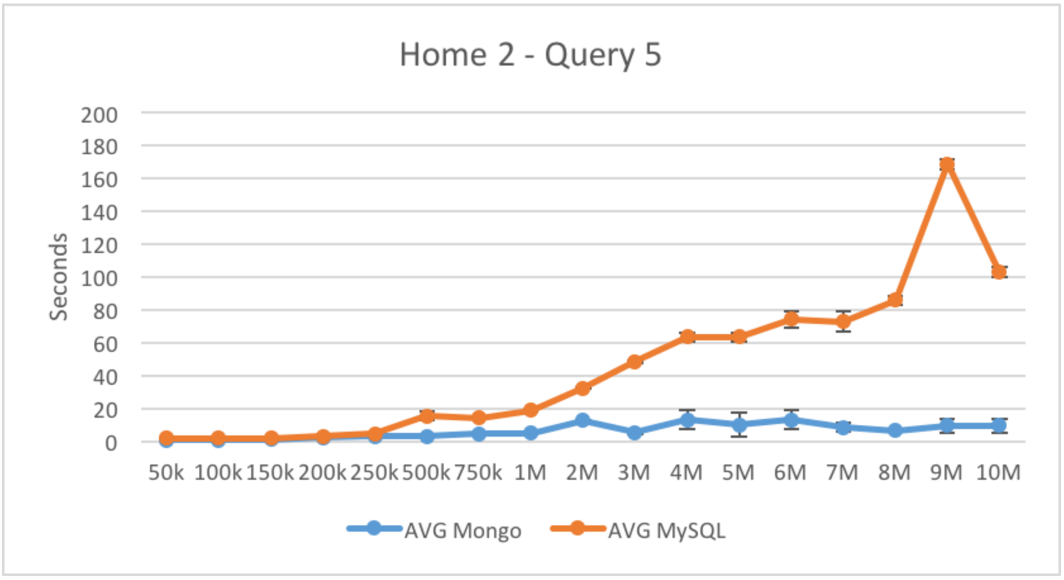


Figure 3.15 – Query 5, Home 2

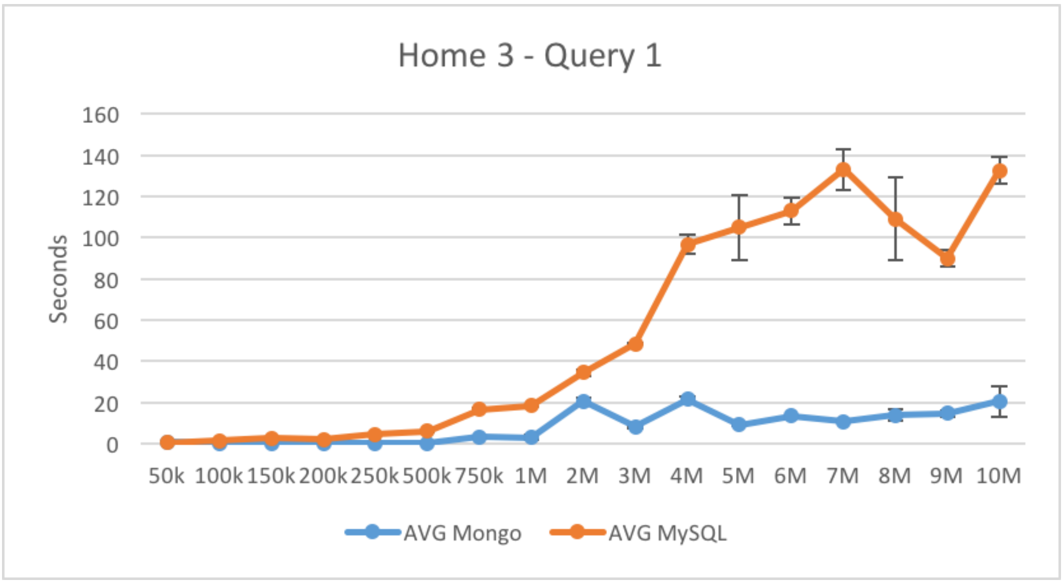


Figure 3.16 – Query 1, Home 3

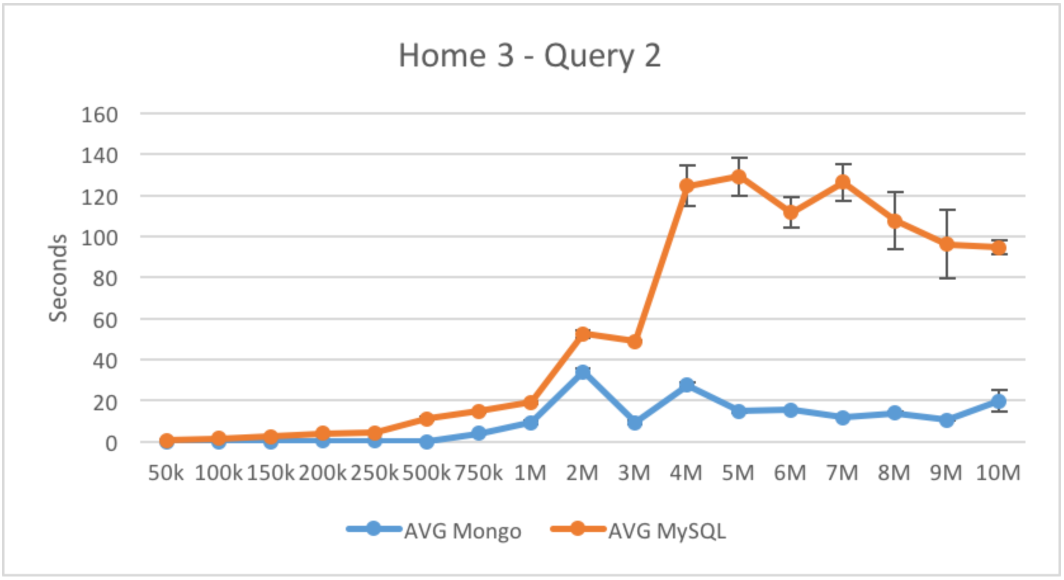


Figure 3.17 – Query 2, Home 3

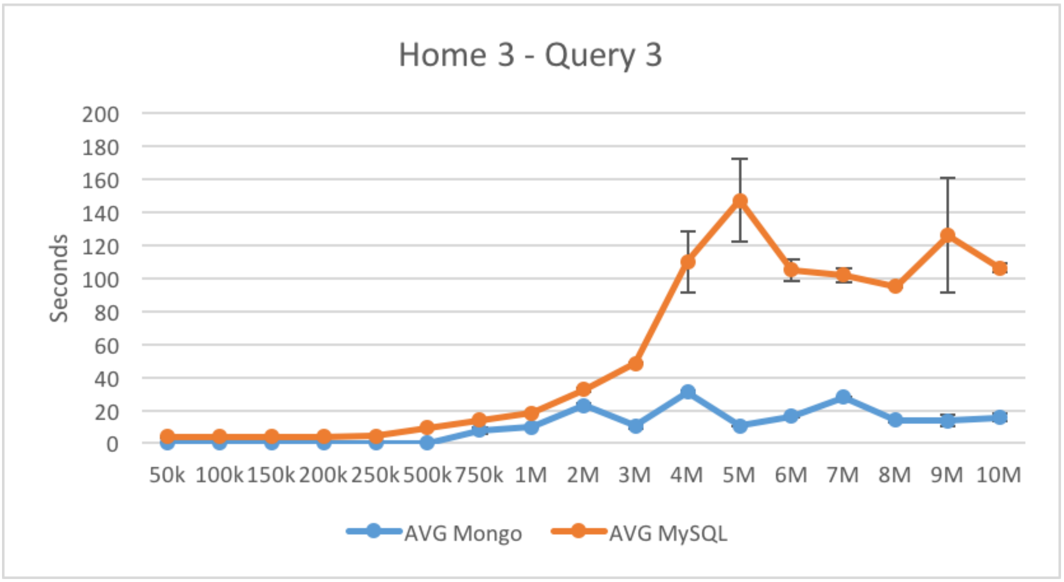


Figure 3.18 – Query 3, Home 3

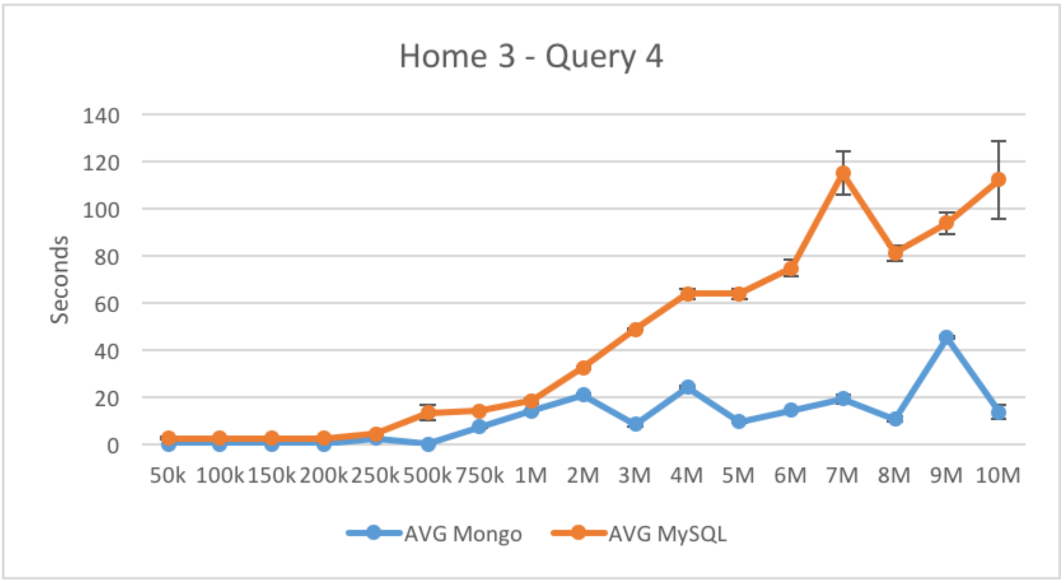


Figure 3.19 – Query 4, Home 3

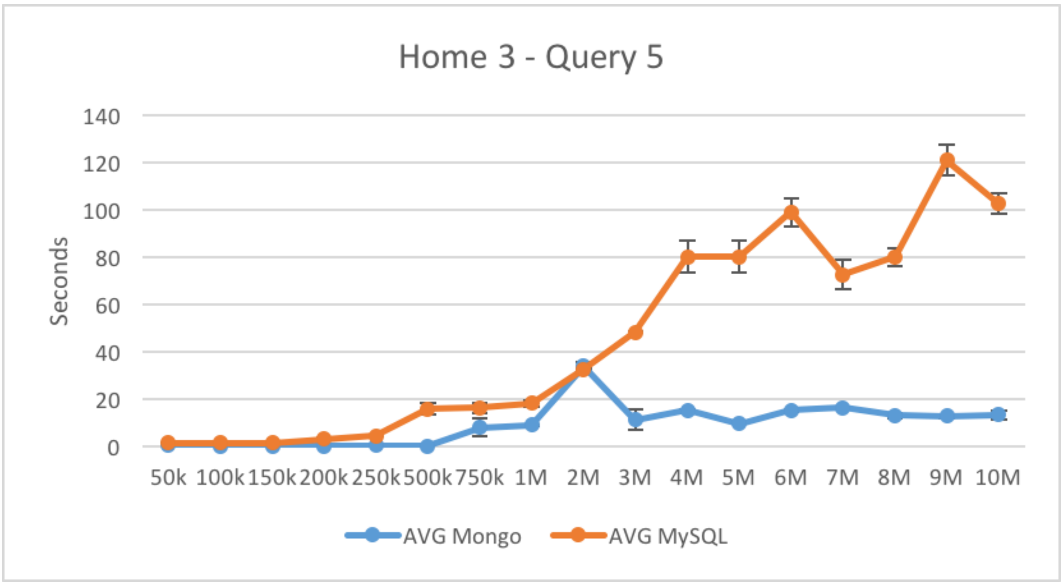


Figure 3.20 – Query 5, Home 3

Group III

As a summary we have the query performance by home ID. Figure 3.21 to Figure 3.23 presents the query performance for all the different aggregation queries performed for the different homes.

As expected from the previous analysis, MongoDB obtains better results than MySQL in every case.

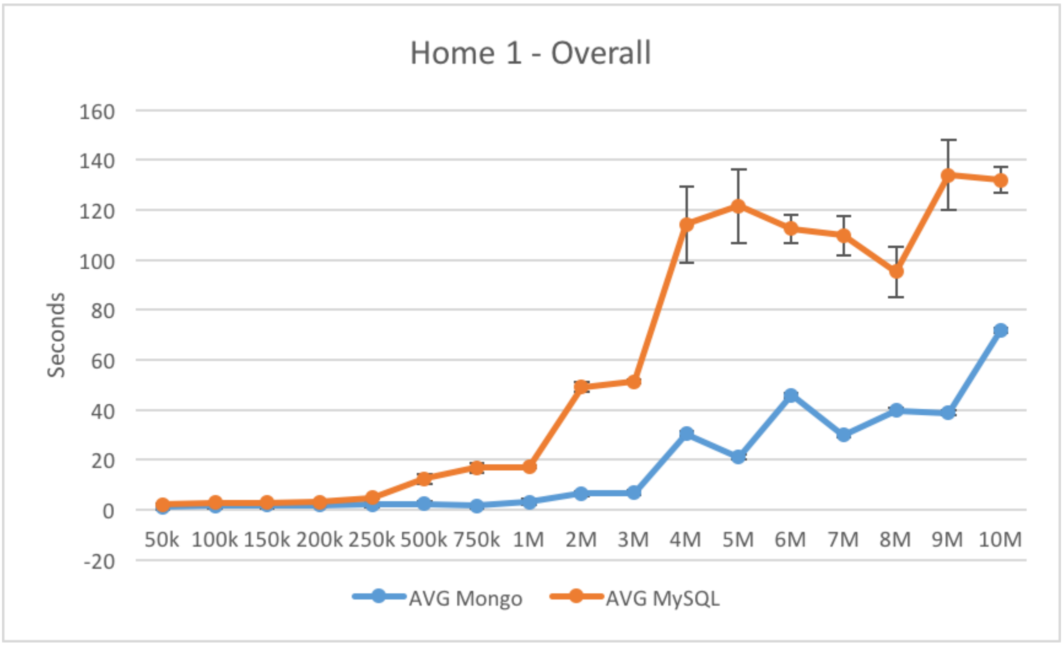


Figure 3.21 – Overall performance, Home 1

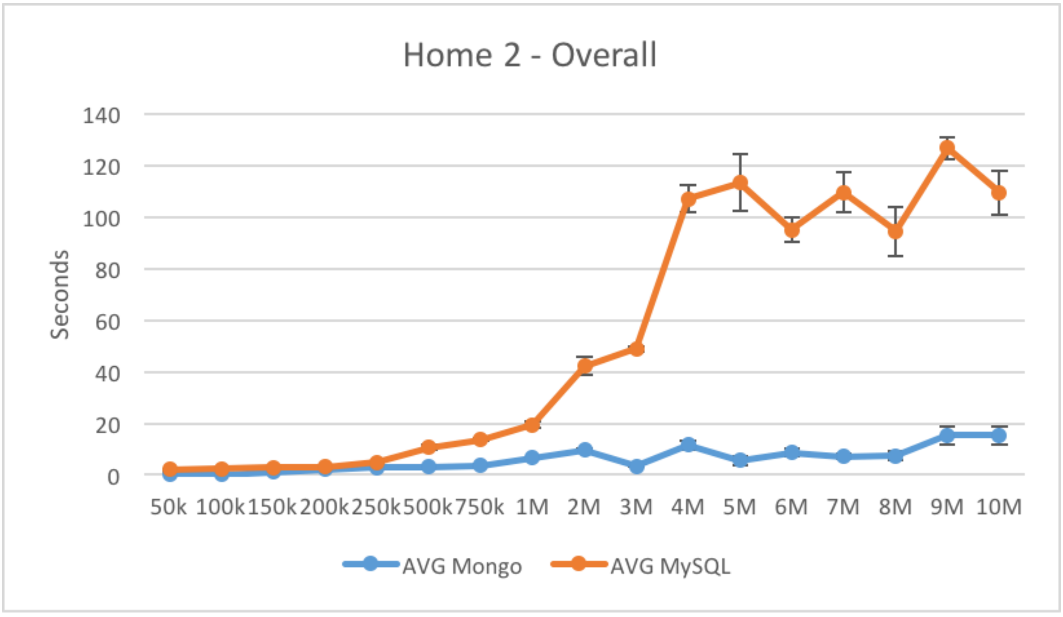


Figure 3.22 – Overall performance, Home 2

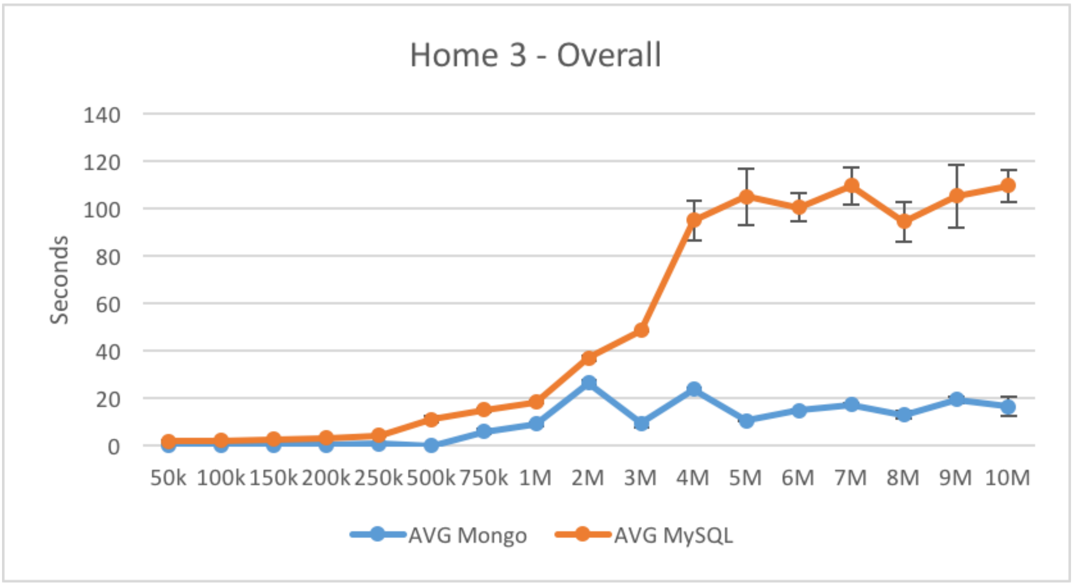


Figure 3.23 – Overall performance, Home 3

Group IV

Instead of having different charts for each query, Figure 3.24 represents the query performance for each aggregation query on both technologies on the same chart, considering all the different home IDs. Here MongoDB is presented by the series with the light blue colour palate.

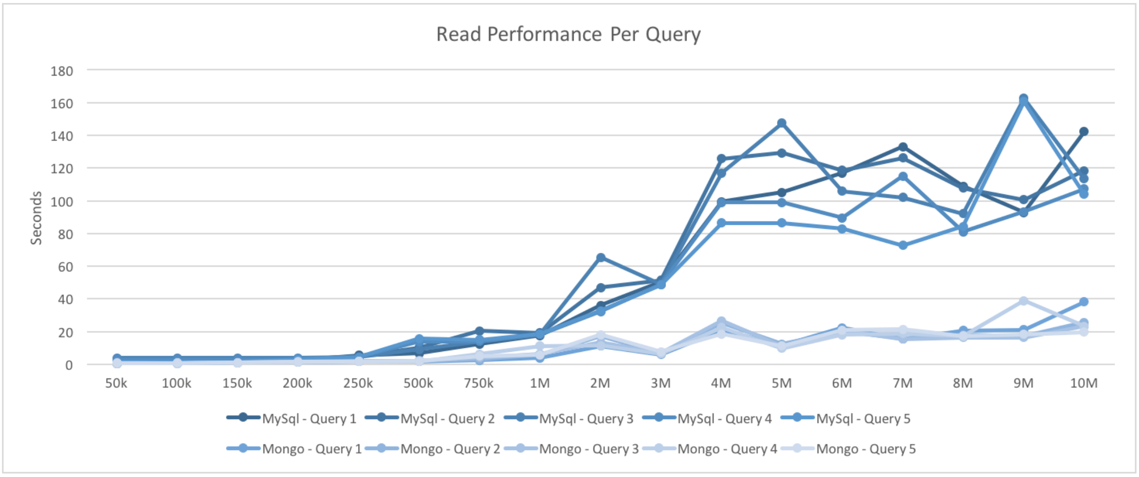


Figure 3.24 – Read Performance (per query)

Finally, as the final overall analysis, Figure 3.25 represents the performance average for all the different queries among the different home IDs.

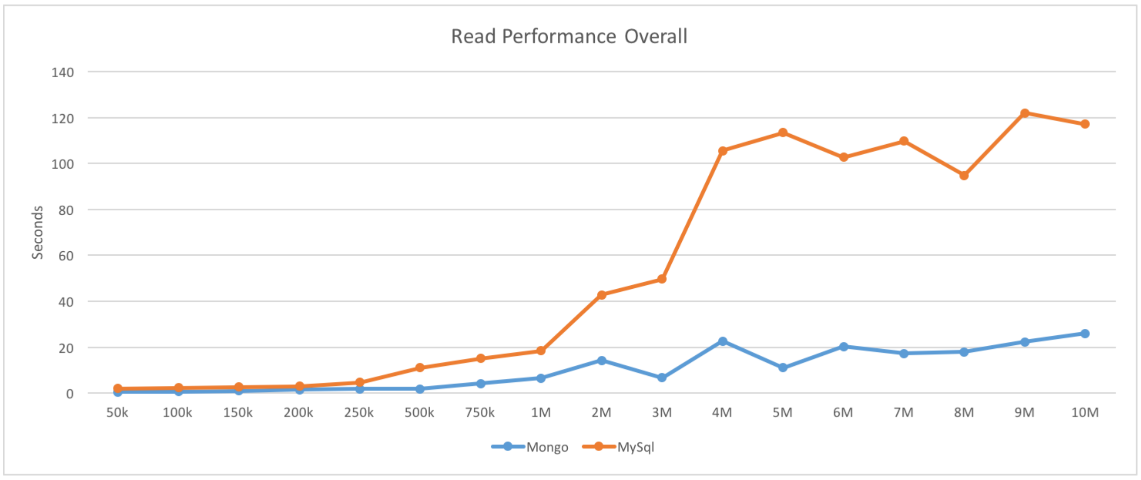


Figure 3.25 – Read Performance (overall)

3.8 Discussion and Conclusions

Having these results, we are able to make our evaluation over insertion and query performance on MySQL and MongoDB, and select one of these as the one that better responds to our purposes and necessities.

3.8.1 Write Performance

Referring to data insertion, MySQL consumed 1 964 MB to store the total of 10 000 000 records and had an average insertion time of 6,15 seconds for all the different sets of data.

On the other hand, MongoDB consumed 4 730 MB to store all the 10 000 000 records and have 25,26 seconds as average for insertion of all the different sets of data. This means that MongoDB takes 240% more disk space and is 410% slower when compared to MySQL.

We should note that the obtained results for storage size and time could be enhanced if we deleted all the data indexes. In fact, indexing data improves reads but compromise writes because every time a record is inserted all the associated indexes must be updated.

Nevertheless, our goal here was to analyse performances on a production environment where there are much more read than write operations, so we decided not to take this approach.

Likewise, the decision of not normalizing the data in MySQL would cost us a non-scalable solution but it definitely helped us to achieve better results both for write and read operations.

MongoDB stores data in a *{key : value}* format in order to enable the store of multiple documents with different structures and avoiding the storage of empty fields. This produces more scalable data structures but consumes more disk space and increases the amount of data to write, since every key must be inserted along with its values.

Is also important to refer the usage of the *mongoimport* tool for insertion, which is known to have worse performance than the MySQL *LOAD DATA INFILE* operation. This occurs because MongoDB's data is stored on BSON format, meaning that most of the effort is spent on data serialization since neither JSON or CSV are native MongoDB formats.

This said, we conclude that MySQL has the best performance for data insertion, since MongoDB consumed about 2,5 times more storage and about 4,1 times more time for data insertion. Yet, once again, it is important to remember that the non-normalized data approach on MySQL it is not an optimal solution, since it leads to a lot of effort every time a change in the database is needed. A situation that is very likely to happen during application operation and maintenance.

3.8.2 Read Performance

Referring to data querying, MySQL produced a global average of about 53 seconds to consult all the different sets of data. On the other hand, MongoDB presented an overall average of about 10 seconds for the same operations, which is about five times faster than MySQL.

We believe that these results could not be much more optimized since we had the care of creating appropriated indexes and stored the data in a way that could offer the best results.

The main reason that justifies this performance difference is in how both technologies store their data. Mongo DB stores embedded data into the same document/collection. This way the data is written in sequential disk positions, which accelerates and reduces the number of round trips to one, since the information can be read all at once. Consequently, because the first disk access is the one that consumes more time (1ms essentially) this detail is important to consider.

3.8.3 Fetching

An important task that we did not consider in this benchmark is fetching the query results. At the beginning of the benchmark this task was performed, but the PHP's maximum allowed memory size was easily exceeded when fetching MySQL results. Consequently, it was not possible to produce a fair benchmark on this type of operation.

In Node.JS + MongoDB algorithm we did not have this problem. When a MongoDB query is performed through the Node.JS driver a cursor is returned. Afterwards we can loop the cursor, iterating all the query results. This operation has a singular particularity: the results are loaded in batches until all the results are fetched. This not only reduces the amount of data to be loaded, avoiding the memory exhaustion but also turns the data access faster since there are smaller chunks of data to return at each time.

An alternative to solve this kind of problems in MySQL is to use OFFSET and LIMIT operators or, for instance, mysql_unbuffered_query function. mysql_unbuffered_query enables the query execution without automatically fetching and buffering the result rows as mysql_query() does. But, besides being deprecated since PHP 5.5.0 it has some costs, we cannot use mysql_num_rows() and mysql_data_seek() on a result set until all rows are fetched, and we also have to fetch all the result rows from an unbuffered SQL query before we can send a new SQL query to MySQL within the same database connection.

3.8.4 Limitations

Although we have created symmetric environments for both technologies in order to test them in the exact same conditions using virtual machines, there was no guarantee that the test execution of one technology would not have impact on the other's performance. This happens because both virtual machines are sharing the same physical resources.

As we can see in the results, some of them present some disturbance and consequently, not very linear results. Such disturbance can be justified by the execution of the tests on both technologies in simultaneous, yet we cannot claim this with 100% confidence. An alternative would be to perform the tests on different physical machines with the exact same conditions but unfortunately at the time of this work we did not posses the necessary resources.

Nevertheless, despite the nonlinear results and observed disturbances, given the significant differences in the obtained results we are confident that the final conclusions would be very similar to the ones presented here.

3.8.5 Summary

Considering the fact that the main goal of OpenDataHub is to bring to the research community a tool for a simple and fast access to public datasets, read operations will necessarily be the most common. Consequently, our biggest concern is to adopt the technology that offers the best performance for this type of operations.

Another concern is the data scalability, since there is a high chance of having to handle non-homogeneous data structures in scientific datasets. The non-normalized approach on MySQL achieved best performance for read/write operations, however this does not offer scalability due to MySQL's relational model.

For this specific case, MongoDB offers the best options, due to its schema-less feature; any kind of documents can be stored along the same collection without the need of any change in the database structure or configuration.

Building these analyses and conclusions, we can summarize our results stating that MongoDB is the best solution for our purposes as it provides the best conditions for data consult and data scalability.

Chapter 4 OpenDataHub

In this chapter we provide a more detailed overview of the *OpenDataHub* platform, an open dataset management system that aims at overcoming the barriers of data sharing and maintenance.

All the stages of a software analysis and development lifecycle are presented in detail from the definition of the system requirements, to the use cases and user interface prototypes. Additionally, we also present the overall architecture of the NoSQL database and the *OpenDataHub* application itself.

As a use case of our approach we will be using the SustData dataset, which was already mentioned in previous chapters.

4.1 Requirements

Our analysis starts with the functional and non-functional requirements definition. To make it more simple and readable, we decided to break some of the requirements into different sections.

4.1.1 Functional requirements

Functional requirements are the definition of the functionalities that our system shall address. Are the functional needs from the original problem and serve as a guideline for the definition of the solution. The following requirements define how our solution shall address our problem needs.

1. Frontend and backend;
   a. It shall offer a frontend workspace for all the users that visit SustData;

   b. An “About” area shall be present, giving users a general overview and introduction to the dataset, the authors and metadata about the different collections that constitute the whole dataset;
   c. It shall present a “Contacts” area, presenting contacts related to the project;
   d. A registration/authentication area, enabling users to register and access the dataset data;

2. Users management;
   a. Shall support user registration and authentication;
   b. Different group of users with different permissions shall be supported;
   c. Account management;
      i. User should be able to change its account settings when desired;
   d. Activity history;
      i. Users should be able to consult their actions’ history over the datasets;
         1. A timeline showing the user’s main tasks performed, such as downloads and collection querying;
   e. A password recovery engine shall be available;
3. Groups of users;
   a. Dataset providers. Users who are responsible for the data sharing and management;
   b. Dataset users. Users that wish to access to datasets and consult the shared data;
4. Data query
   a. Users shall be able to consult all the public collections created by data providers;
   b. For each collection, it should be possible to filter data using filters for each one of the fields present on the collection;
   c. Filters shall vary according to the field type (defined through the metadata). I.e., date-time picker for timestamps, numeric input for decimals and text input for text fields.
   d. On data query, users should be able to transform the data, having opportunity to identify which fields shall be presented;
   e. Users shall be able to group their results by timestamp (minute, hour, day, week, month and year;
   f. Users shall be able to limit the number of obtained results;

g. A data preview shall be performed by the system, presenting all the data resulting from the produced query;
i. Two types of preview shall be supported. Graphical and tabular data.
h. Users should be able to download the data resulting from the performed queries. It also should be possible to download all the raw data i.e., without performing any kind of query.

5. Data management;
   a. Data management shall only be available for dataset providers;
   b. Users shall be able to create collections identifying the proper metadata;
   c. Users should be able to import new data for existing collection whenever they wish;
      i. CSV and JSON formats shall be supported;
   d. Users should be able to identify a collection as publicly available or private;
   e. Private collections shall be available only for the dataset providers;

4.1.2 Non-functional requirements

Non-functional requirements aim to describe how the system is supposed to be. They are used to judge the operation of a system, rather than specific behaviours. The list below presents OpenDataHub non-functional requirements.

1. Scalability;
   a. The solution needs to scale in order to manage large volumes of data and dynamic creation of collections;
2. Availability;
   a. Any "Down time" shall be avoided, engines that guarantee high availability shall be introduced;
3. Performance;
   a. The application needs to perform requests in a non-blocking operation approach in order to achieve the best performance rates possible;
4. Recoverability;
   a. In a "Down Time" situation the system should be able to recover non-losing any data recorded before the occurrence;

5. Security;
   a. The systems shall confine features according to the user permissions;
6. Usability;
   a. It shall have a user friendly interface, i.e., users should easily understand how to operate the system without needing instructions in addition to what is already in the user interface;

4.2 Use cases

Use cases are used for defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. Figure 4.1 shows a graphical representation of the system's requirements and its users' permissions.

As one can se by the diagram there are three types of users. Dataset providers can manage the different collections that compose the dataset. Dataset users on the other hand can only manage info about their own account and query the data provided. Lastly, non-authenticated users can only access a brief description of the dataset, related contacts and the registration section.

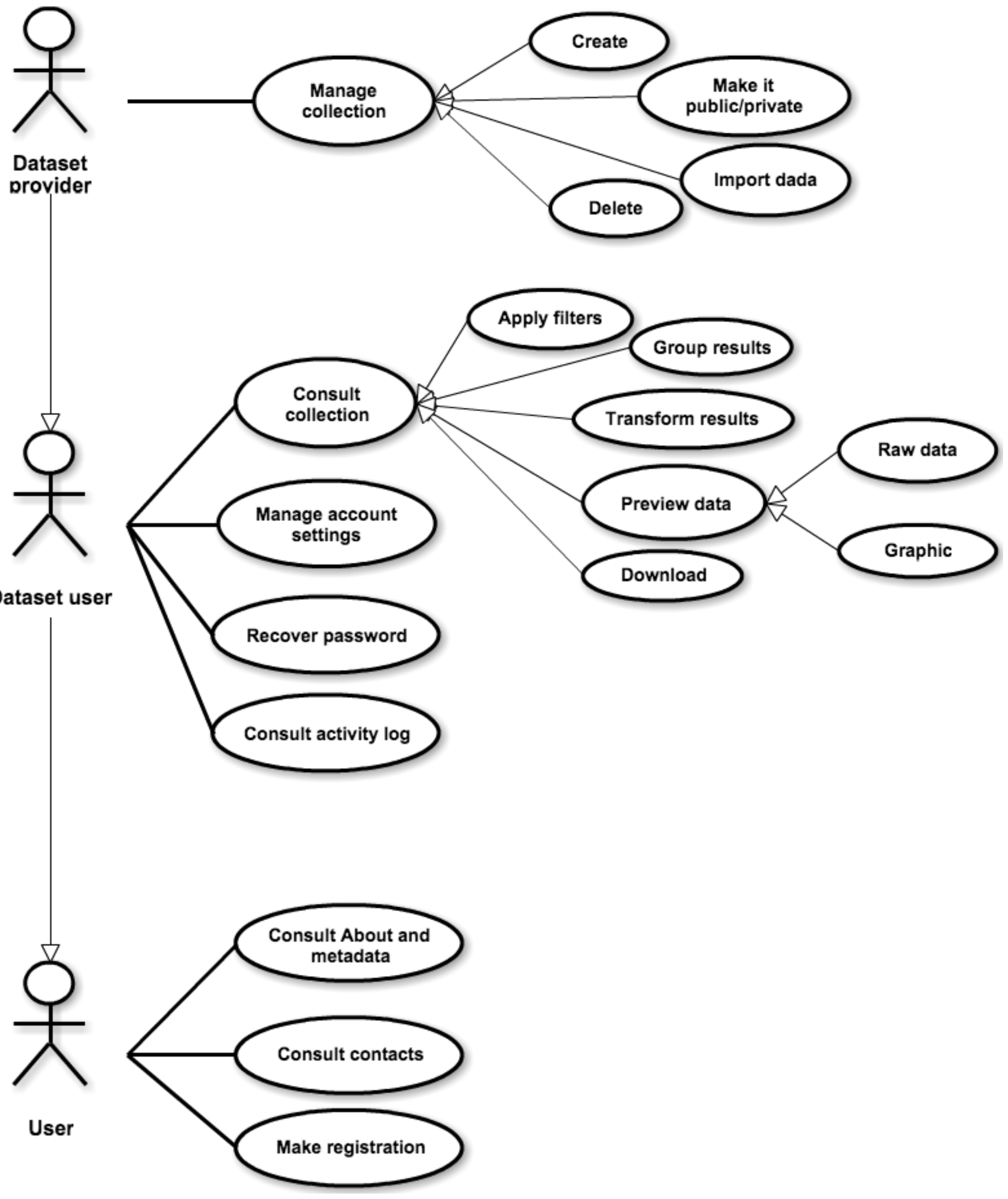


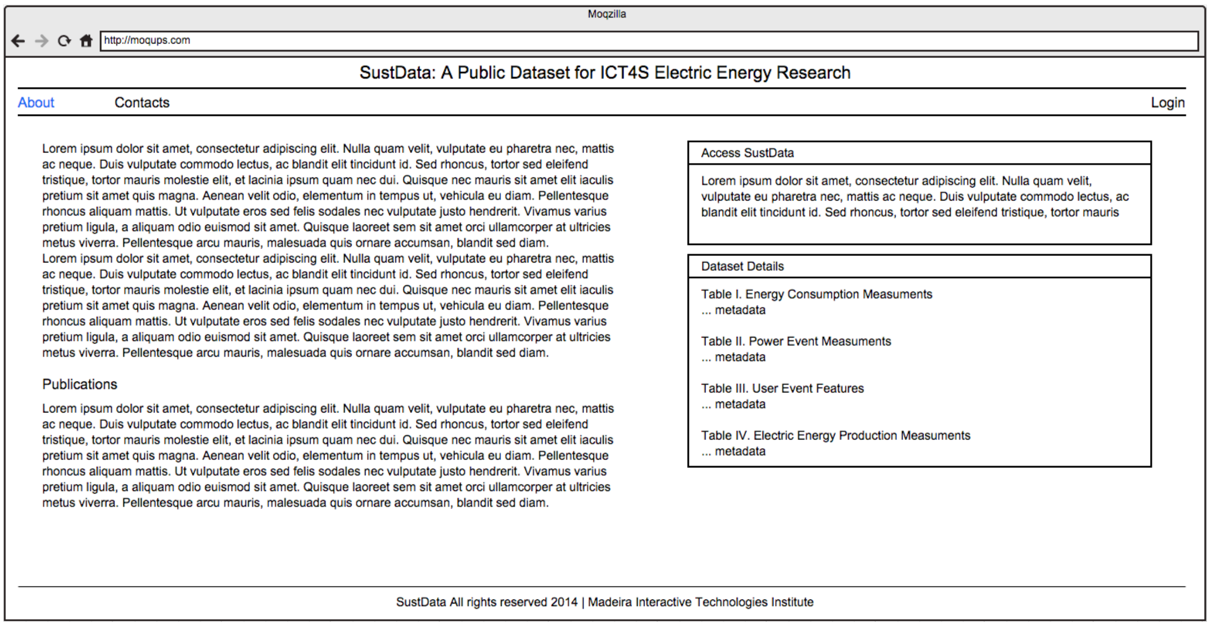
Figure 4.1 – Use cases

4.3 Prototypes

Having the system's requirements and use cases defined, we are able to build the user interface that will enable users to access and manipulate the dataset in different ways.

The following prototypes were built using the SustaData dataset as a test case.

Figure 4.2 represents the home page. There users can find a brief introduction of the dataset, related publications and general metadata about the different collections that constitute the public dataset.



SustData: A Public Dataset for ICT4S Electric Energy Research

About Contacts Login

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Publications

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Access SustData

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris

Dataset Details

Table I. Energy Consumption Measuments
... metadata

Table II. Power Event Measuments
... metadata

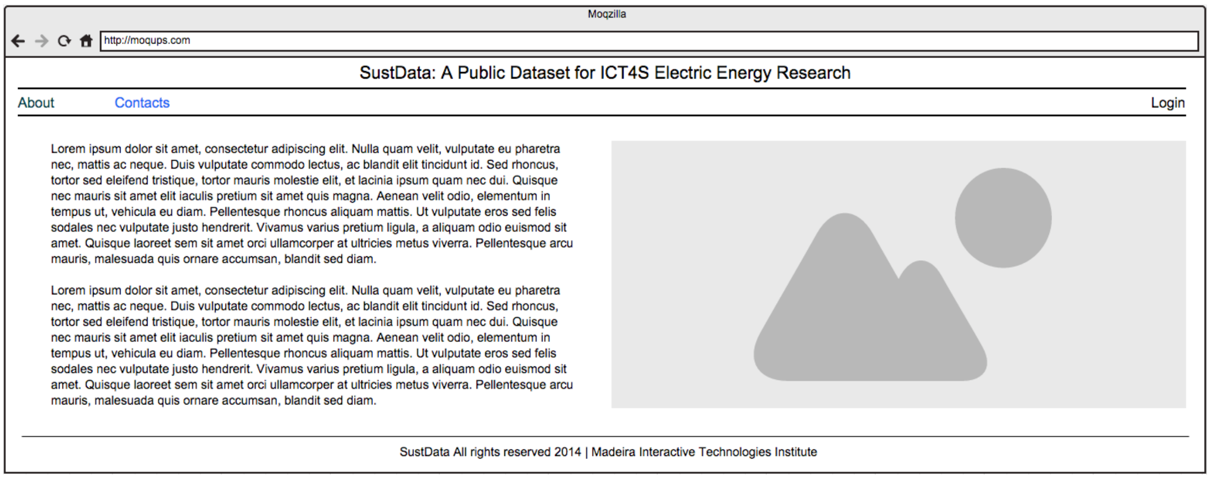
Table III. User Event Features
... metadata

Table IV. Electric Energy Production Measuments
... metadata

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.2 – About page

In the contacts page, represented on Figure 4.3, users can find some information about dataset creators. This includes the people involved and how and who should be contacted in order to get further information.



SustData: A Public Dataset for ICT4S Electric Energy Research

About Contacts Login

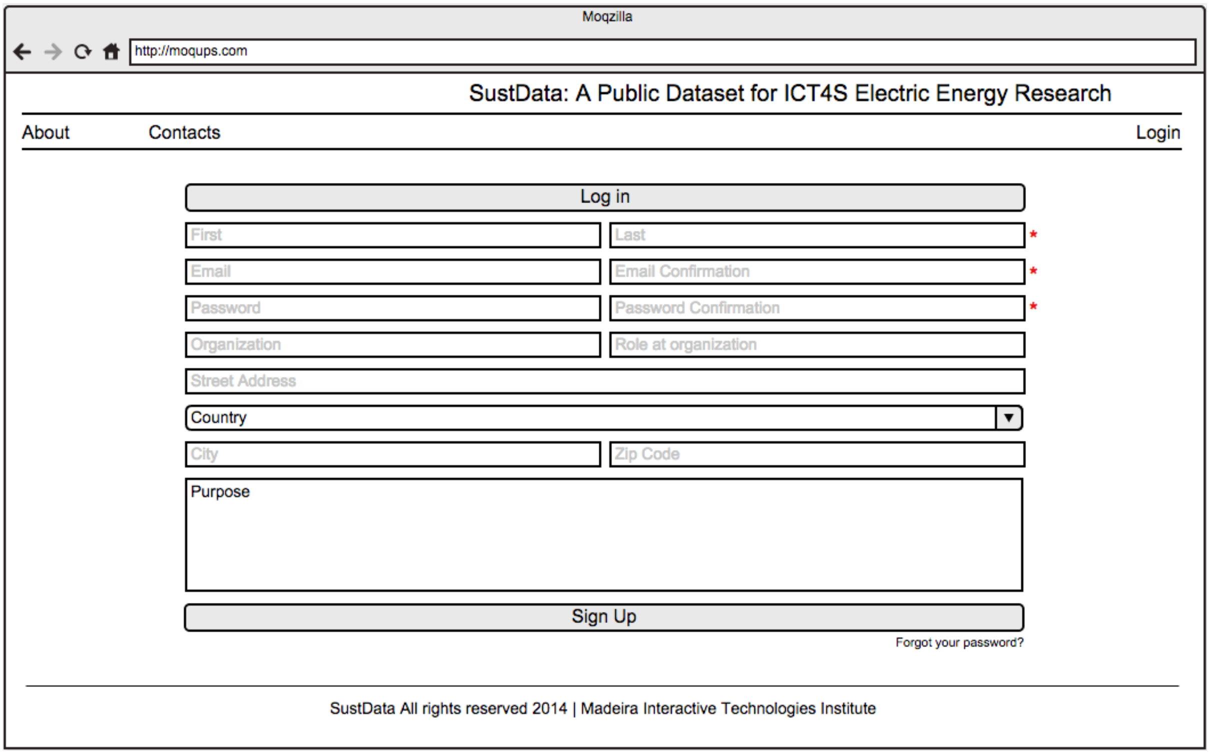
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.3 – Contacts

Users need to register in order to gain access to the dataset collections. For this we have the registration page presented by Figure 4.4, asking users for some basic demographic information.


Moqzilla

http://moqups.com

SustData: A Public Dataset for ICT4S Electric Energy Research

About Contacts Login

Log in

First | Last *

Email | Email Confirmation *

Password | Password Confirmation *

Organization | Role at organization

Street Address

Country ▼

City | Zip Code

Purpose

Sign Up

Forgot your password?

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.4 – Sign up

Once registration is done, users are able to authenticate in the system and therefore access all the public data available. Figures Figure 4.5 and Figure 4.6 show the authentication page and the password recovery form.

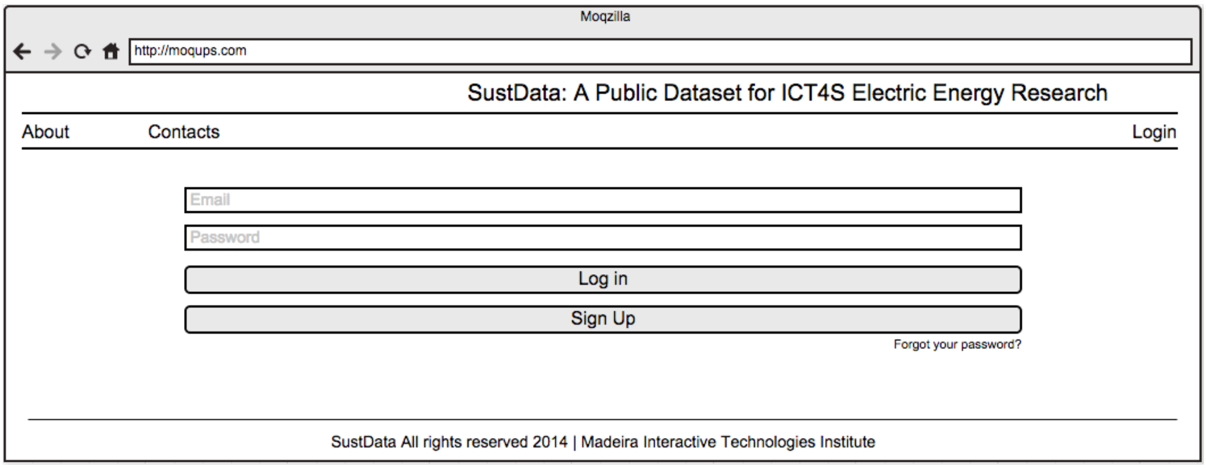

Moqzilla
http://moqups.com
SustData: A Public Dataset for ICT4S Electric Energy Research
About
Contacts
Login
Email
Password
Log in
Sign Up
Forgot your password?
SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.5 – Login

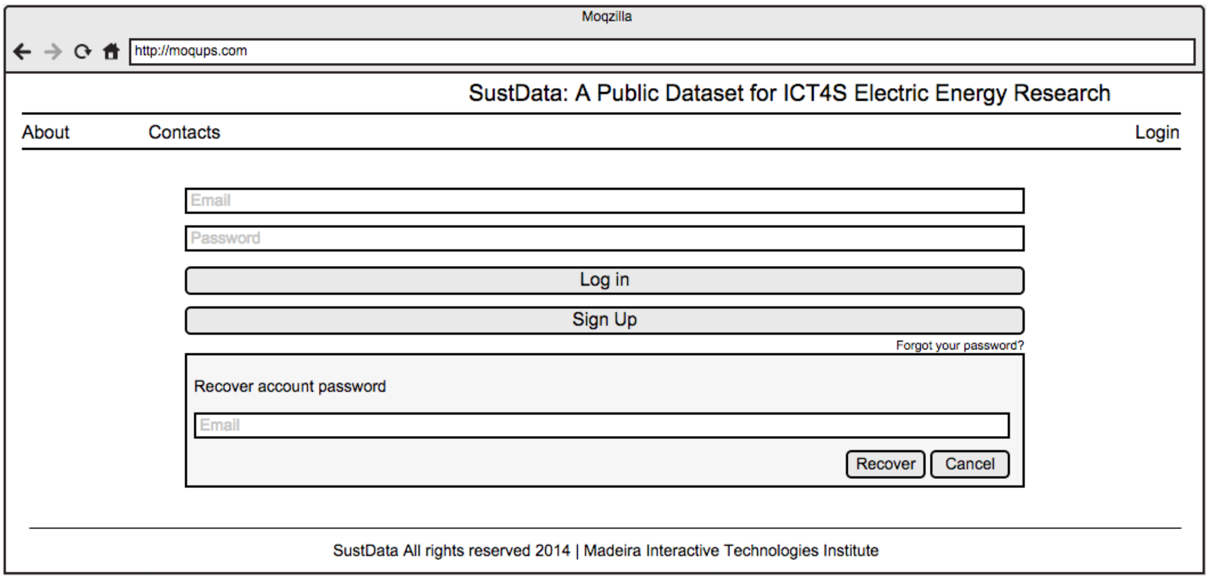
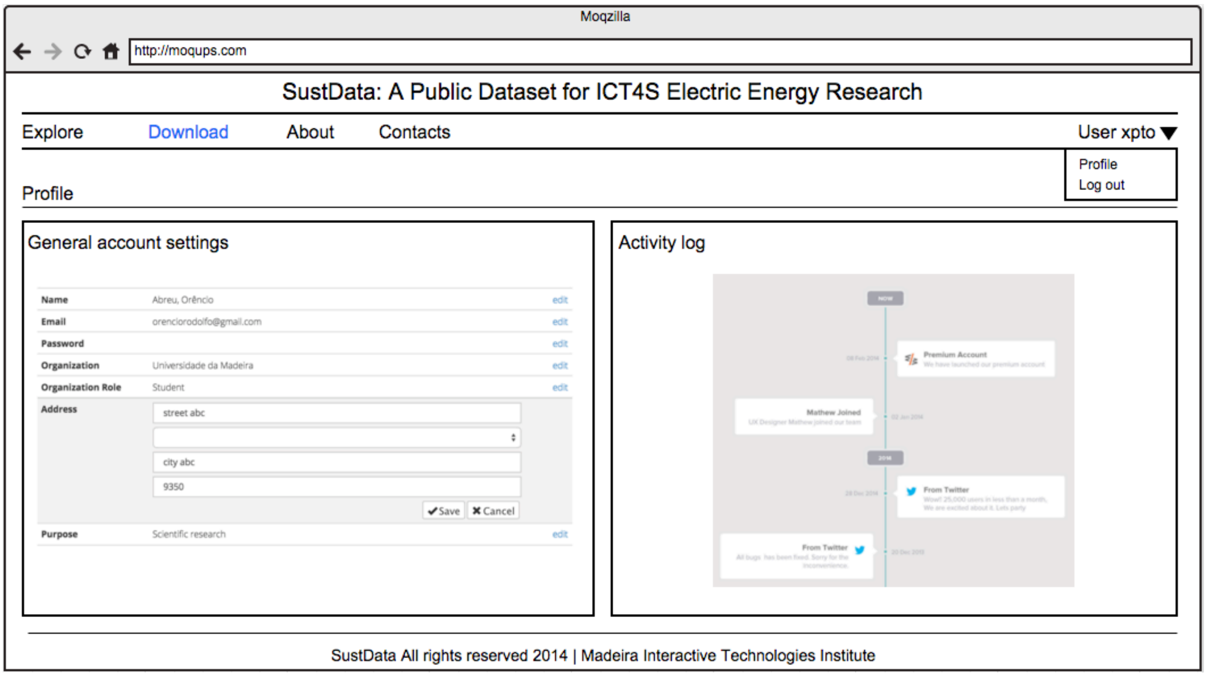

Moqzilla
http://moqups.com
SustData: A Public Dataset for ICT4S Electric Energy Research
About
Contacts
Login
Email
Password
Log in
Sign Up
Forgot your password?
Recover account password
Email
Recover
Cancel
SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.6 – Password recovery

When a user authenticates there are some new functionalities available. One of them is the user profile. In the user's profile, presented by Figure 4.7, all the basic information is available to consult and update. Also the user activity history is presented, showing the main tasks performed by each user on the platform. This can be interesting to consult previous performed queries and reproduce them for instance.



Name	Abreu, Orêncio
Email	orenciorodolfo@gmail.com
Password	
Organization	Universidade da Madeira
Organization Role	Student
Address	street abc city abc 9350
Purpose	Scientific research

08 Feb 2014 Premium Account — We have launched our premium account

02 Jan 2014 Mathew Joined — UX Designer Mathew joined our team

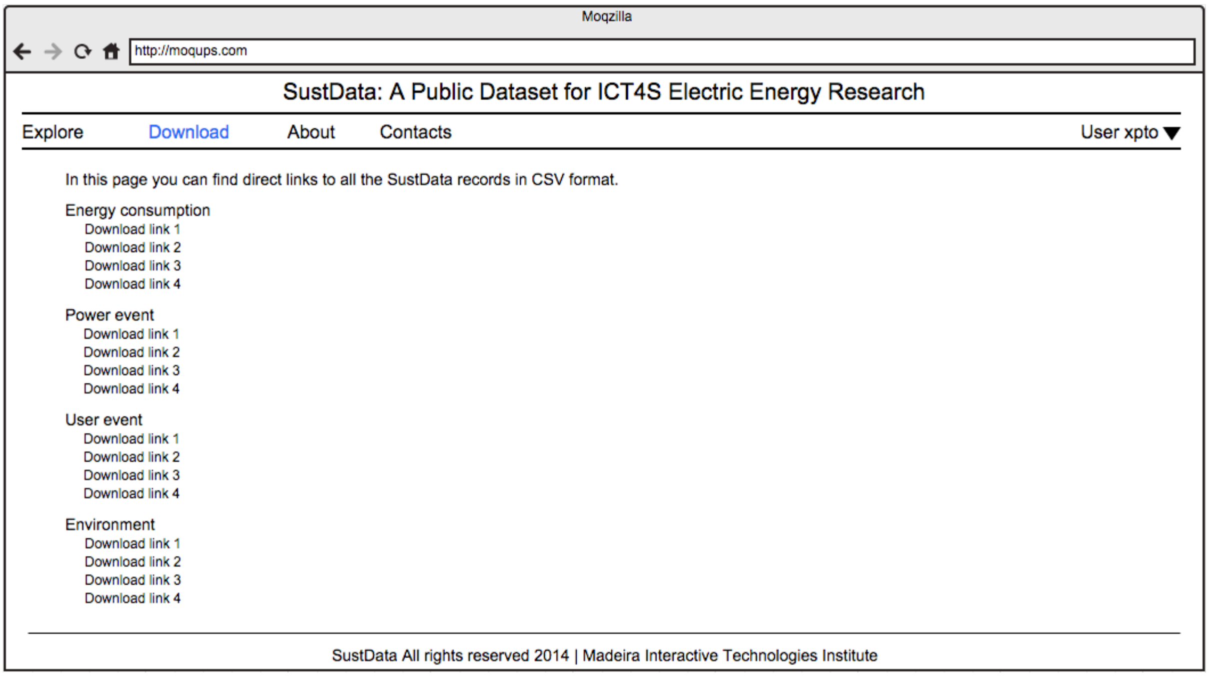
28 Dec 2014 From Twitter — Wow! 25,000 users in less than a month, We are excited about it. Lets party

20 Dec 2013 From Twitter — All bugs has been fixed. Sorry for the inconvenience.

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.7 – User profile

Users can be interested in downloading the whole dataset and manage its data with their own data management system. For this we provide the download page, represented on Figure 4.8, where users can download big CSV compressed files containing all the raw data of SustData.



SustData: A Public Dataset for ICT4S Electric Energy Research

In this page you can find direct links to all the SustData records in CSV format.

Energy consumption
- Download link 1
- Download link 2
- Download link 3
- Download link 4

Power event
- Download link 1
- Download link 2
- Download link 3
- Download link 4

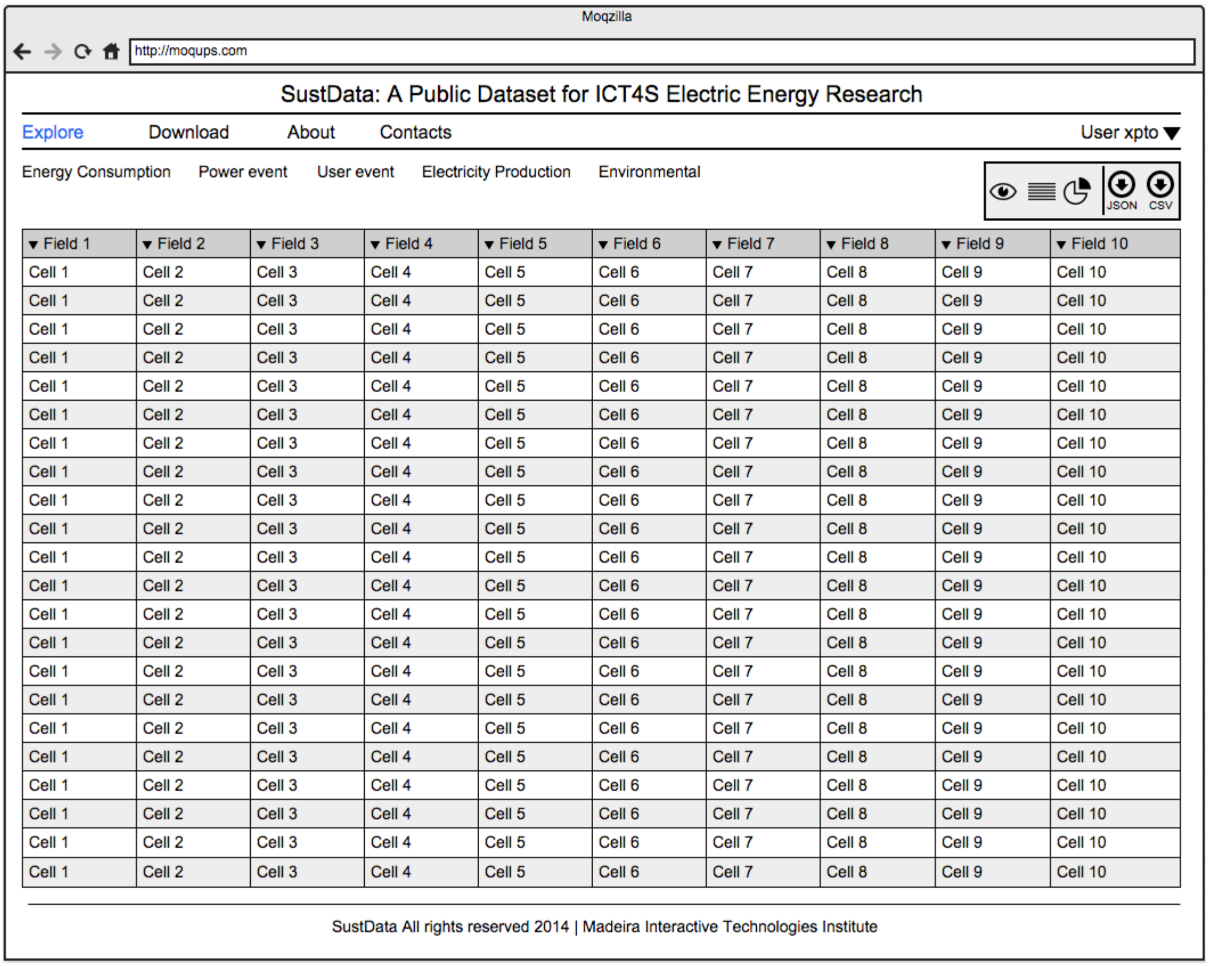
User event
- Download link 1
- Download link 2
- Download link 3
- Download link 4

Environment
- Download link 1
- Download link 2
- Download link 3
- Download link 4

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.8 – Download

The “Explore” page, offer dataset users access to all the public data available on SustData for query, visualization and download results on CSV and JSON formats. Here users can navigate between the different collections, filter the data by performing different queries and having a preview of results in the form of lists or charts.



SustData: A Public Dataset for ICT4S Electric Energy Research

▾ Field 1	▾ Field 2	▾ Field 3	▾ Field 4	▾ Field 5	▾ Field 6	▾ Field 7	▾ Field 8	▾ Field 9	▾ Field 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10
Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7	Cell 8	Cell 9	Cell 10

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.9 – Table data visualization

Like shown in Figure 4.9 and Figure 4.10, dataset users are given the opportunity to preview their query results. Where, in a graphical mode, time is represented on X axis and the remaining variables on Y axis.

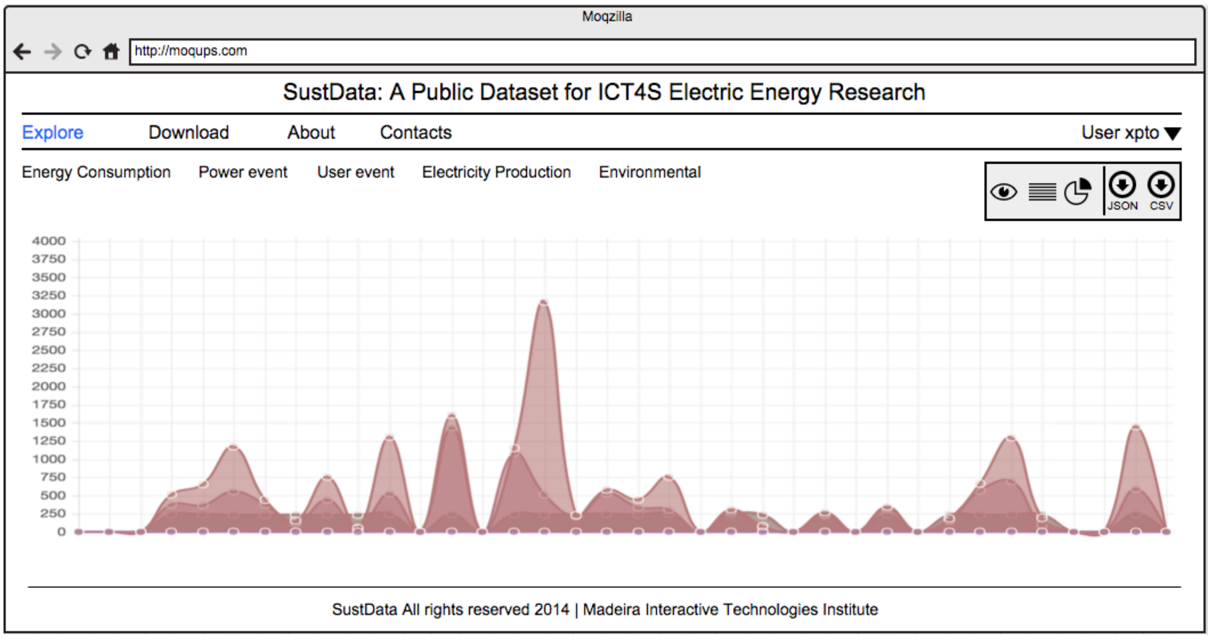


Figure 4.10 – Graphical visualization

The query builder pop-up is presented in Figure 4.11 and enables users to apply several filters to the different field that compose a particular collection. For each type of data stored on each field, different types of filters are available.

Users can select which fields they wish to view toggling the available checkboxes and can aggregate the results by different time units, like “Minute”, “Hour”, “Day”, “Week” and so on.

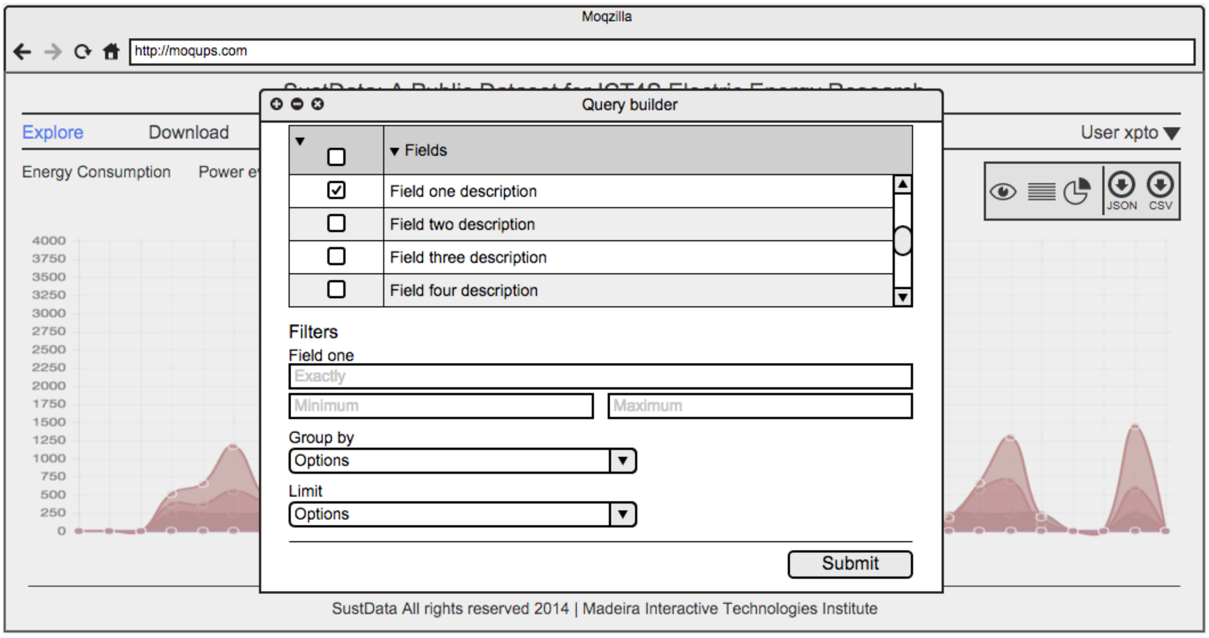
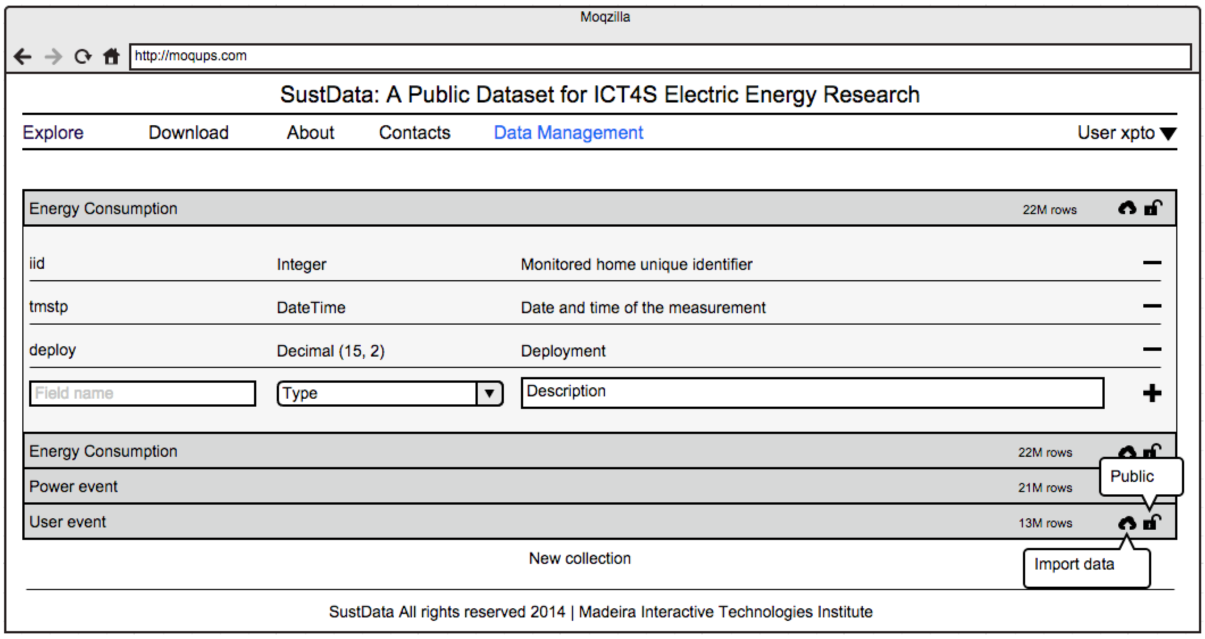


Figure 4.11 – Query builder UI

On Figure 4.12 we have the data management area. Here, dataset providers are able to manage their data collections and publish them.

Here, each existing collection is presented to the user that is also given the opportunity to import new data or even change the specifications (i.e., the metadata) of the already existing collections. Through this mechanism, dataset creators can identify which fields shall be available to query and the data type that each one should hold. For a easier data interpretation, there is the description of each field.

There is as well the possibility of making collections public or private, where private collections will only be available for dataset providers.


Moqzilla

http://moqups.com

SustData: A Public Dataset for ICT4S Electric Energy Research

Explore Download About Contacts Data Management User xpto

Energy Consumption		22M rows	
iid	Integer	Monitored home unique identifier	−
tmstp	DateTime	Date and time of the measurement	−
deploy	Decimal (15, 2)	Deployment	−
Field name	Type	Description	+
Energy Consumption		22M rows	
Power event		21M rows	
User event		13M rows	

New collection

SustData All rights reserved 2014 | Madeira Interactive Technologies Institute

Figure 4.12 – Data Management

4.4 Technologies

From the analysis in Chapter 2 about some of the most known and used technologies for web development and data storage (see section 2.2) and having the problem statement of SustData, we are ready to evaluate which technologies can serve our purpose and attend our needs.

In the next sections we will present the main technologies that support SustData, starting from the storage, passing by the server language and finally ending at the client side technologies that will build the user interface.

4.4.1 Storage

In this kind of projects we are dealing with large volumes of data that can assume several different structures and can grow very fast. It is therefore important that the final solution provides very good scalability.

When dealing with this kind of problems, it is inevitable not to talk about NoSQL databases. In this particular scenario MongoDB has proven to be a really good candidate to handle datasets like SustData (please refer to the Chapter 3 for mode details about this). It is easy to scale as it supports horizontal scaling, allows the choice of the consistency level of the data, it is schema-less and the data is in BSON format, a very flexible and simple to understand data structure.

Thanks to these features and all the advantages mentioned before when talking about No-SQL databases, MongoDB is our database of election. Later on this chapter we will see how data will be stored on MongoDB and which decisions over *scalability*, *replication* and *consistency* were made.

4.4.2 Server side

Performance and scalability are the major concerns for this platform. Having our MongoDB setup set, is important to select a server-side technology that offers a good interface with the database and that can offer good performance and scaling features.

There are several technologies that offer good database drivers like Python for instance, so this was not a major concern for our decision. However, besides the need for a good database driver, we need a technology that presents good performance values, manages physical resources efficiently and is easy to scale.

Node.JS has emerged in web development world as one of the lightest and easiest to scale technologies. It is open source, has a large and growing community producing every day new different resources and scales very well. Additionally, it is very fast thanks to the V8 Javascript engine and it is very suitable to deal with JSON data structures, which is a great plus when dealing with MongoDB for compatibility purposes.

Working together with Node.JS we have Mongoose[26], a MongoDB object modelling tool that help developers to validate, cast and perform some business logic. Mongoose also enables developers to define the database schema.

It can be somewhat contradictory to the schema-less approach of MongoDB but sometimes it is necessary to guarantee that the application will always deal with the same data structure, specific data types, and with data that meets some specific requirements. Mongoose comes to provide a layer to address these needs.

In short, having all these in consideration, Node.JS is naturally our server side election.

4.4.3 Client side

Finally, we present client side technologies. Here **HTML**, **CSS** and **JavaScript** are obvious choices. Nevertheless, several other technologies come to speed up and guide development to standard and better-accepted solutions by the community.

Instead of HTML we will use **Jade**[27]. Jade is an HTML template engine that supports dynamic code, reusability (DRY), requires less code overall and offers mechanisms that can conduct to better productivity.

Instead of using pure CSS, we will use **Less**. Less and Sass are CSS pre-processors that have proven to be great for building code easier to maintain, extend and customize.

For icons, since we selected **Font Awesome**[28], which provides hundreds of vectorial icons through fonts, which makes the introduction of icons very easy while enabling the possibility of scaling and colouring the whole user interface using CSS like a text font.

Concerning JavaScript, **AngularJS** along with **jQuery** and **Angular Material** were our choices. These technologies come to provide more abstract layers that make JavaScript development more intuitive and therefore easier to produce.

[26] Mongoose, http://mongoosejs.com/ [last accessed 12/11/2015]
[27] Jade, http://jade-lang.com/ [last accessed 14/11/2015]
[28] Fontawesome, https://fortawesome.github.io/Font-Awesome/icons/ [last accessed 09/02/2015]

A clear advantage of using JavaScript on the client side is that with this, we have a homogeneous solution. With both JSON and JavaScript languages in the database, server and in the frontend user interface we have a homogeneous solution that implements all its logic and data presentation using the same core technologies.

4.5 Development tools and frameworks

There are several different tools that make developers' work easier and more practical to perform. In this section we provide a very brief overview of the different public resources used in OpenDataHub development.

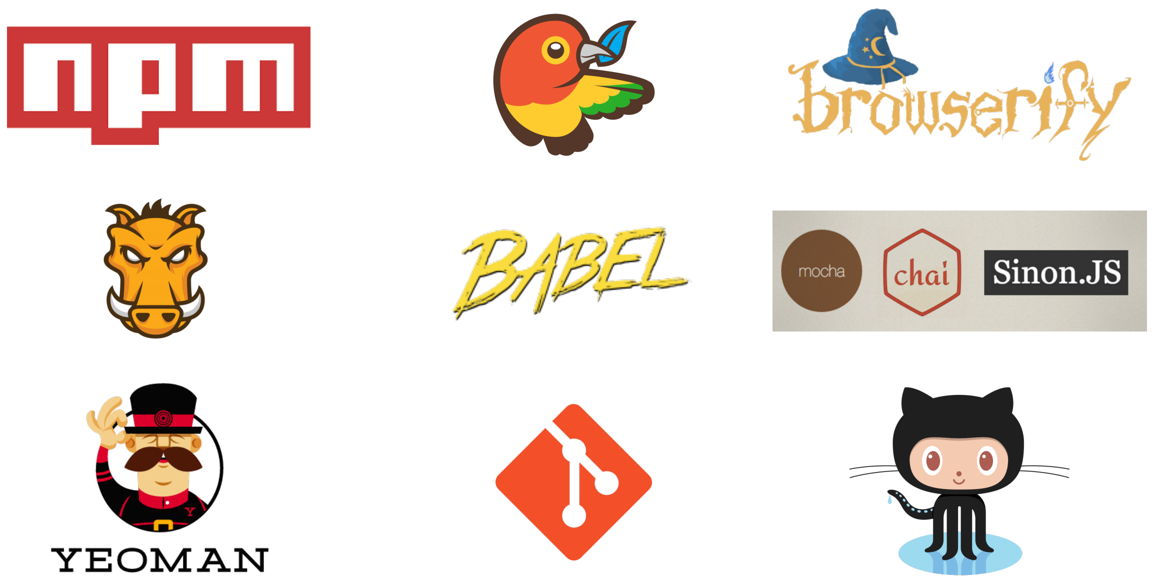


Figure 4.12 – Tools and frameworks

NPM and Bower

For our Node.JS dependency management we use **NPM**[29] a package manager that is responsible for referring to all the external dependencies in our project and enabling a fast and simple installation of all of these whenever a new developer wishes to collaborate to OpenDataHub project.

29 https://www.npmjs.com/ [last accessed 18/012/2015]

NPM stores references to production and development dependencies as well as the versions in use. For deployment and in order to guarantee availability and operability of the platform we use npm-shrinkwrap[30] for locking down dependency versions and therefore maintain a homogeneous deployment installation in any different environment.

Very similar to NPM, we have **bower**[31]. A package manager for the frontend dependencies, storing reference to all the external public resource dependencies ant its versions as well.

Browserify

Browserify[32] is a Javascript module loader where we can create JavaScript bundles that are independent of any external resources that are not required.

For instance, if in our application we have globally available both jQuery and Moment.js and our bundle only needs jQuery, in our bundle we write "require('jQuery')" and only jQuery will be available within our bundle code.

This makes our implementation much more modular and non-fragile to the external JS world.

Grunt and Babel

Grunt[33] automatizes several tasks in our project. In our case, we are using *ES2015*[34] in order to write JavaScript in a nicer way and taking advantage of the different recent JavaScript features. Yet ES2015 is not fully supported on many browsers, so we use **Babel** JavaScript compiler to compile ES2015 to the original JavaScript such that the different web browsers can interpret it.

This compilation is a very common task to perform during the implementation phase, so we use **grunt** and **grunt watch** to do it for us in the background.. We also use it to *minify* and

[30] Shrinkwrap, https://docs.npmjs.com/cli/shrinkwrap [last accessed 04/02/2016]
[31] Bower, http://bower.io/ [last accessed 09/02/2016]
[32] RequireJS, http://requirejs.org/ [last accessed 10/02/2016]
[33] Grunt, http://gruntjs.com/ [last accessed 10/02/2016]
[34] ES2015, https://babeljs.io/docs/learn-es2015/ [last accessed 11/02/2016]

uglify our source code for production environment, in order to enhance speed on the browser and prevent the interpretation of our client-side implementation through the DOM query.

There are thousands of tasks that can be performed through Grunt, compile, minify and uglify are just some examples. In our project we also use it for compiling Less into native CSS, to concatenate our JS bundles, to deploy the application and several other tasks.

Mocha + Chai + Sinon

Mocha[35] is a unit test framework very suitable to test driven development and for testing our NodeJS API. Mocha will enable us to run several tests over the API and guarantee its correct behaviour.

Chai[36] is simply an assertion library that comes to complement Mocha, it enables us to write tests over Mocha in a more intuitive manner.

Sinon[37] also comes as a complement to the Mocha framework. As it is natural, our application modules communicate between them and sometimes with other resources that are external to the application. As our goal is to write unit tests, we need to guarantee that our code is tested without being affected by external resources. For that we use Sinon, an API that enable us to fake the behaviour of external resources, giving us the opportunity to explore the different responses that the unit that is being tested can face in a real scenario.

Yeoman

Yeoman[38] is a code generator tool that combined with grunt and bower helps to achieve higher rates of productivity and code quality. With this we can easily build new applications or JavaScript components. Yeoman helped us especially on our server side development.

[35] Mocha, https://mochajs.org/ [last accessed 12/02/2016]
[36] Chai, http://chaijs.com/ [last accessed 12/02/2016]
[37] Sinon, http://sinonjs.org/ [last accessed 12/02/2016]
[38] Yeoman, http://yeoman.io/ [last accessed 12/02/2016]

Git, GitHub and GitLab

For the versioning and deployment we use **Git**[39] version control system and as remote servers we have **GitHub**[40] and **GitLab**[41], a *Git* repository where we can easily store our projects and manage not only the code versions but also handle tasks like *issues*, *wiki* and others.

4.6 Database architecture

The database is the most critical part of our system due the big-data nature of SustData data. MongoDB is very suitable to large volumes of data but it is important to build a well-defined strategy to take the best out of the many advantages that MongoDB can provide to our problem. Therefore, we designed the database architecture and decided how to configure our database server in order to address all our needs.

To achieve better performance and reduce the risk of data loss, we divided our biggest collections over three different *shards*. Each *shard* is constituted by a replica set of three databases, one of them acting as primary and the other two as secondary databases.

The use of *replication* reduces the probability of down time since on a scenario where the primary database becomes unavailable one of the secondary will be selected as the new primary. Then, in the meantime, the old primary will come back up and when it does, it will perform tasks to synchronize its data according to the data stored on the primary database.

This can be done through the *oplog* or by copying the entire dataset from the primary. It depends of the failure scenario, but the first approach is the most common.

By default, *replication* sets only permit reads and writes on the primary database for consistency reasons. There are some mechanisms such as *write concern* and *journal* that are essentially mechanisms that enable database users to choose what level of consistency they want, depending on the value of the data.

[39] Git, https://git-scm.com/ [last accessed 19/09/2015]
[40] GitHub, https://github.com [last accessed 19/09/2015]
[41] GitLab, https://about.gitlab.com/ [last accessed 19/09/2015]

If we need higher performance, we can write on a single node and return a response to the application and the *oplog* will be responsible for enabling to rest of the nodes to acknowledge the new updates. On the other hand, if we need higher consistency, we can ensure that a write may happen on the majority or even on all the nodes that constitute the replica set and then return a response to the application. Naturally this second approach is slower than the first one.

Due the nature of SustData dataset, having *eventual consistency* is not a major problem, so we decided to enable read operations over secondary nodes and make writes only over the primary node and let the *replication* set synchronize the rest of the nodes.

For instance, if one user gets data from the primary database with 5 000 results from a specific query and another user gets 4 980 for the exact same query it is not critical. It is public data for posterior analysis and has no immediate impact on any external environment. Furthermore, public datasets generally are published when the study ends and therefore, new updates to an existing collection are not very usual.

This decision drives our solution for higher performance and reduces the workload over the primary nodes of each *replication* set.

MongoDB is very well known for its capability to scale horizontally, and this can be achieved through data *sharding*. For SustData we have created a cluster constituted by three *shard* nodes, each node with the *replication* set also with the tree nodes and three *config* servers containing metadata for the *sharded cluster*.

These *shards* split the SustData's biggest collections in equal or almost equal parts, dividing the database into parts and therefore requiring lower physical resources for performing the data queries.

Ideally these shards and each node of the *replication* sets should live in different physical servers, as only in this way we would be able to take full advantage of all the benefits of data sharding, dividing the workload and the resource consumption over different machines. This does not only provide higher rates of availability as well it leads to higher performances.

Yet, due the lack of resources and given that SustData is a case study, we developed this architecture on the same server, launching different services on different ports.

Figure 4.13 shows the architecture that sustains SustData's database.

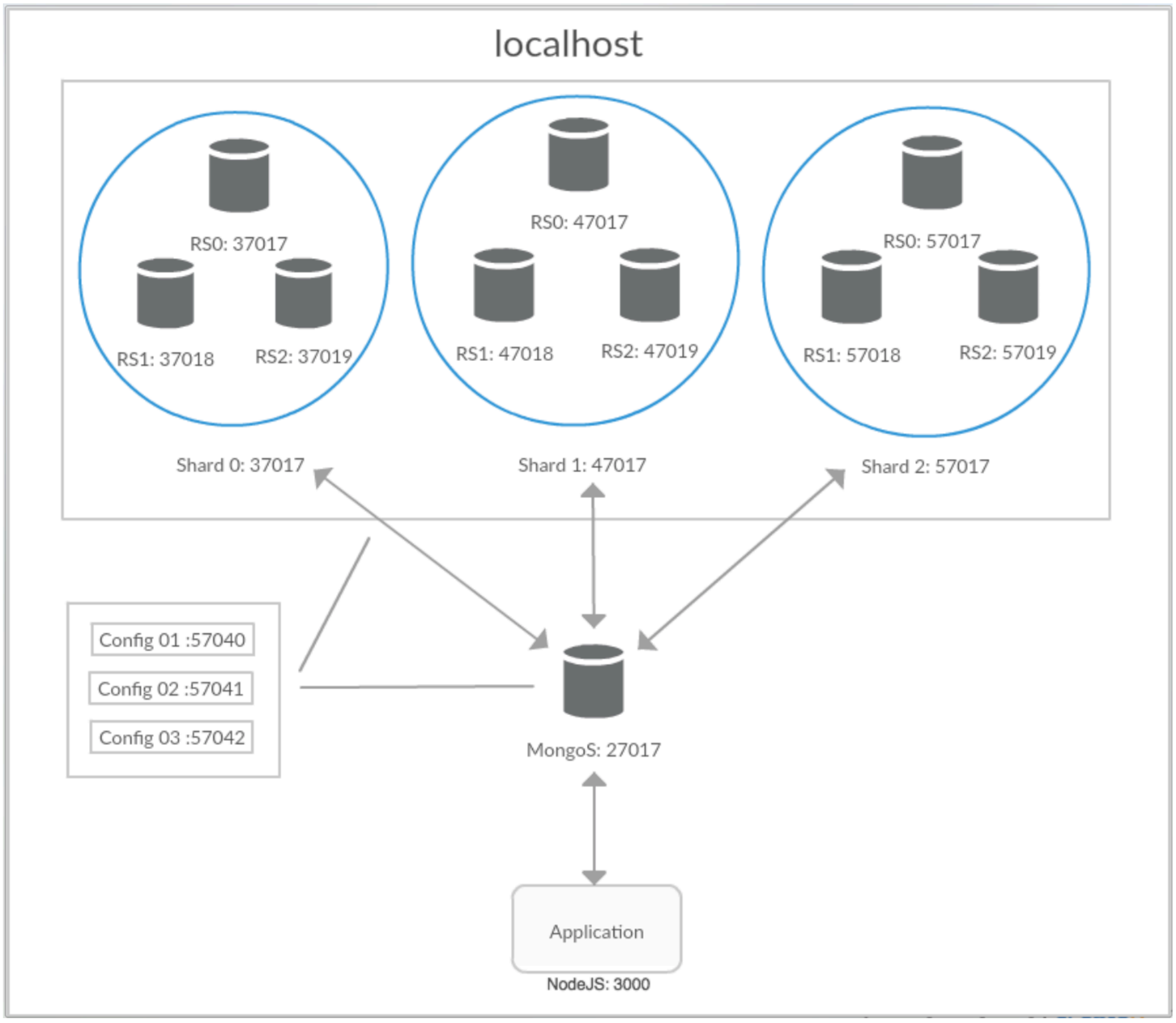
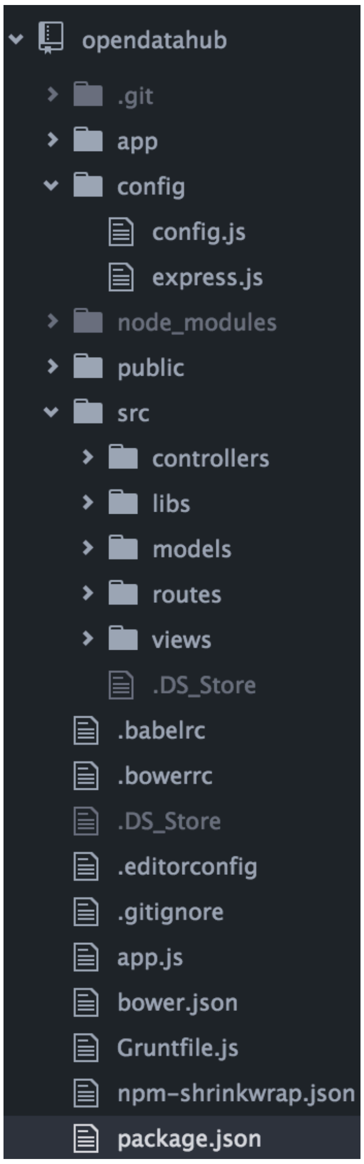


Figure 4.13 –Data *sharding* and *replication*

On ports *57040*, *57041* and *57042* we have our config servers and on ports 37017, 47017, 57017 we have the primary database of each shard, and on ports *18* and *19* of each of these ranges would be the secondary databases. MongoS, running in port *3000*, is our database driver. The one responsible for receiving requests from the database and forwarding queries for the respective shards.

4.7 Application structure



```
opendatahub
  .git
  app
  config
    config.js
    express.js
  node_modules
  public
  src
    controllers
    libs
    models
    routes
    views
    .DS_Store
  .babelrc
  .bowerrc
  .DS_Store
  .editorconfig
  .gitignore
  app.js
  bower.json
  Gruntfile.js
  npm-shrinkwrap.json
  package.json
```

Figure 4.14 - Application structure (server)

Concerning to the server side, there are several Node.JS frameworks that aim to facilitate and speed up the development process. Express[42] is our framework of election, offering great functionality and flexibility over the Node.JS itself.

Our entry server's point is the *app.js* present at our project's root directory. Next we have *src, app* and *config* folders.

The *config* folder contains all the configurations to our express (default responses for *404* and common middlewares for instance), core dependencies load and the configurations to our application, such as the server port, the database connection details and other configurations that might vary on *development, test* and *production* environments.

Regarding to the *src* and *app* folders, the *src* contains all the server-side logic implementation and main HTML templates where implementation is written in ES2015[43] standard, while the *app* folder has the exact same implementation but it's the result of the compilation of the *src* code. This folder is dynamically generated by a Grunt task when compiling the code using Babel.

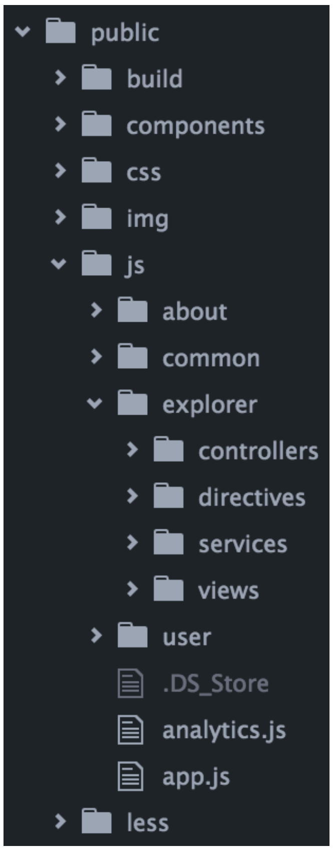
Another important components of our server-side implementation are the *package.json* and the *npm-shrink-wrap.json*. These files are responsible for storing information about our application, such as the application name, author, version and more importantly, all the project dependencies.

[42] Express, http://expressjs.com/ [last accessed 03/12/2015]
[43] ECMAScript 6, https://babeljs.io/docs/learn-es2015/ [last accessed 03/12/2015]

The *package.json* stores both development and application dependencies giving the opportunity to receive new versions of this dependencies by a simple “npm install”.

On the other hand, *npm-shrinkwrap.json* stores only the application dependencies, not giving the opportunity to receive newer versions of the specified dependencies and respective version. This is important in a production environment in order to avoid the installation of versions of our dependencies that were not tested and can compromise our solution availability, consistency and reliability.

Finally we have the *node_modules* folder, storing all the external dependencies to our project that are installed via NPM.



```
public
  build
  components
  css
  img
  js
    about
    common
    explorer
      controllers
      directives
      services
      views
    user
    .DS_Store
    analytics.js
    app.js
  less
```

Figure 4.15 - Application structure (client)

Regarding to the client side, our entry point is also the *app.js,* but this time the one that is in the root of the *public* folder. The *public* folder contains all the implementations and resources that are referent to OpenDataHub client side implementations.

Here, the *js* folder contains all our implementations with AngularJS, dividing it into a component per module. I.e., a component for the *data explorer* module, another to the *user profile, registration and authentication*, another to the *about* and so on. Here, controllers, directives, services and views compose each one of these components and each one of these components will be translated into a JS bundle through *browserify*.

The *img* folder contains all the images that are used and the less folder contains all our application style sheets written in Less that are compiled through a Grunt task. The result of this compilation is stored in the *css* folder in order for the browser be able to interpret it.

In the *build* folder we have all the resulting native JavaScript from the ES2015 code that is implemented in the *js* folder. This folder contains the resulting bundles that are ready to inject into the different HTML pages. In case we are in a production environment, this folder will store all the code in a minified version in order to enhance the traffic in our pages.

Finally, and similarly to the *node_modules* folder, there is the *components* folder containing all the external resources to our client side that are installed via Bower. The *bower.json* file, analogous to the *package.json,* stores data about the frontend application like its name, version and all its dependencies such that is it possible install of them through a simple "*bower install*" command.

4.8 Summary

The presented architectures were not a result from our first attempt. In a first iteration the MongoDB database was a single database with no replication. Yet, while this would responds to our functional needs it would not guarantee high availability and data integrity. Consequently, we had to change our strategy introducing data clusters with the data sharding and replication techniques.

Lastly, it is important to stress once again that at its current state, we are not taking full advantage of the sharding and replication benefits. This will happen once it is possible to distribute the shards in different machines.

Chapter 5 Conclusions

In this chapter we aim to expose the different challenges that were presented to us during the development of OpenDataHub as well as the different solutions that were found to overcome such challenges.

5.1 The challenge

Starting with our problem, the need for a tool to enable researchers to easily share their research data and, dataset users to easily have a brief introduction and preview of these before downloading it for posterior analysis. This was a need that emerged when the SustData research team had to publish their dataset on the Internet.

The research team did not find a platform where they could publish the data produced during the SINAIS project and at the same time offer to the dataset users novel means to query, manipulate and preview the dataset contents. Furthermore, due the data dimension there were several difficulties when dealing with the data. The data was originally stored in different databases or CSV files, making the data difficult to maintain.

Given all this constraints we built our vision over this problem and all the revolving issues around it and started to explore and evaluate existing solutions to learn how these could help us build our proposed solution.

We found several platforms that shared our main goal and vision, of improving scientific data sharing. Each of them using distinct techniques but always producing the same output, a web page containing a brief description of the data and download links for downloading large files in formats like html, csv or xml.

Yet, ultimately, none of the already existing solutions served our purpose. The main reason for this was the fact that data resulting from scientific research can assume different structures, data types and is very likely to assume very large dimensions. Consequently, learning how to deal deal with such issues was one of our main tasks before building any kind of proposal for solution.

This kind of problems is not really new as of today, and several technologies have already been created and progressively improved to overcome barriers on big data storing. After our technological background research we conclude that NoSQL would be centre of our solution and using SustData as a test case for the realization of our vision.

Yet, before selecting NoSQL as the way to go, we had to prove in practice that this was really the best alternative. To this end, we conducted and extensive benchmark between SQL and NoSQL technologies that come to confirm that for this kind of problems NoSQL technologies would offer better overall performance.

However, from this analysis another problem has emerged. If we would follow the NoSQL approach we had to guarantee high capability of physical storage since our test shown that NoSQL requires higher rates of physical storage for storing the same portion of data. We could not afford from a super machine that could be able to address the needs that a public dataset can require.

Fortunately, we found the technique of splitting the data form the same database across as many servers as necessary through data sharding and the ability of building clusters. At this point, before building our proposed solution, we had pretty much the answers to our main problems.

5.2 Developed work

NoSQL vs SQL Benchmark

Although there are several articles and studies comparing SQL and NoSQL databases, we built our own study using the SustData dataset as a use case, what was important to evaluate which option will be the best for our purpose.

We used PHP for the MySQL database and the NodeJS for the MongoDB database. This could raise doubts about the final results, but in our study we only measured the time consumed by the database operations, so the programming language used has no impact over the final results.

The reason why we use PHP for the MySQL and NodeJS for the MongoDB was thanks to its drivers to interact with relational and non-relational databases. PHP offers great drivers for MySQL databases, such as PDO[44] and active records from frameworks like Laravel[45], Yii[46], CodeIgniter and so on. On the other hand, NodeJS offers as well a great driver and modelling tools like Mongoose.

As final result, we had MySQL with better results on write operations and MongoDB with better results on read operations. On write operations MongoDB consumed more disk space and took longer storing the same data, although several articles mention MongoDB as being faster both on read and write operations. But we believe that these results on writes were thanks the tool that we used for importing data directly from a CSV file, the *mongorestore*.

Having the results, we built our conclusion based on our applications purposes. Since we are dealing with dataset publication and sharing, there will be much more data reads than writes. Because once the data is uploaded and shared, it will be unusual to suffer changes or

[44] PDO, http://php.net/manual/en/book.pdo.php [last accessed 21/02/2015]
[45] Laravel, https://laravel.com/ [last accessed 21/02/2015]
[46] Yii, http://www.yiiframework.com/ [last accessed 21/02/2015]

updates, on the other way, it will be very likely to be queried in different ways. And, for reading, MongoDB is the leader.

Another advantage of taking MongoDB is its schema-less features. Public datasets are very likely to have non-homogeneous data structures and being able to easily handle different data structures within the same data collection is a big plus.

OpenDataHub

In our work definition we followed the waterfall sequence for software analysis, starting on requirements definition, use cases, prototypes and so on. In this phase, we decided to introduce two additional steps, the definition of the application and database architectures.

Having this phase concluded we start a more interactive cycle between implementation and the application technologies. This interactive process was the result of the continuous increase in our familiarity with the used technologies, which meant that different visions were emerging during the development.

After some interactions we found that our core of OpenDataHub application should suffer some changes in order to introduce REST concepts and standardize our client/server communication. It would turn the communication with our server side more homogeneous and standardized. This would also conduct our solution to a more modular structure, which made the implementation easier to understand , maintain, scale and test.

At this time we also find unit tests as being important to guarantee the right behaviour of our components, so we decided to introduce Mocha, plus Chai and Sinon and with these a new manner to produce our code. First we write our unit tests and then we proceed with the solution development. This way we have a moment to stop, think in our component needs and responsibilities, describe the component through the tests and then implement it. Avoiding the implementation of solutions that at the end does not attend all our needs because we did not think about them from the beginning.

Another major change in our solution along these interactions was the introduction of AngularJS along with Angular Material. At the beginning our frontend was implemented with jQuery and jQuery UI libraries but along the time we had the opportunity to take a closer look over AngularJS framework and Angular Material and the advantages that it could bring to our project. This way we proceed with the refactor of our frontend implementation, building a more modular and clean solution. It also enabled us to create responsive interfaces and to introduce unit tests on the frontend implementation more easily.

Considering the database, initially we had a single database with a set of collections that would sustain persistent data about users, users' activity and SustData's data.

However, although this was enough for SustData dataset we had in mind build a solution that could serve many datasets as possible with different sizes. For that, we had to build a scalable solution and with that goal, we introduced data sharding. Furthermore, availability and data loss were major concerns as well so we proceed with data replication as well.

As result, we came up with a cluster with three shards and in each shard a replication with three databases. This would enable us to divide our data into chunks across the shards and we could easily add more shards as we wish and data would automatically rebalance between these.

On an ideal environment all these database nodes, both replication databases and shards, should be present on different servers, on different physical machines in order to guarantee that an unavailable server would not compromise data availability or integrity. But we did not have all this resources, so we had to build this structure on a single server launching mongo instances on different ports.

This would not actually give us the advantages on data sharding but served as a use case for the used technology.

Having our strategy set and a first stage of the project concluded, it was time to deploy it. Several hostage options were explored, including Amazon AWS for instance. But on most of them we could not find a free service that could serve our needs. This way *M-ITI, Madeira*

Interactive Technologies Institute[47] provide us a server where we could install all our project dependencies, including Git who turn the deployment tasks much easier and clean to perform.

OpenDataHub is an open source project and can be acceded on GitHub: https://github.com/OrencioRodolfo/OpenDataHub.git

Here users will find the project sources and how can they download it, install and launch the application.

For accessing SustData dataset, users can simply register on SustData's web page and access all its public data (http://aveiro.m-iti.org:3000/).

5.3 Future work

NoSQL vs SQL Benchmark

Like the OpenDataHub source code, the source code for the benchmark is also publicly available. So other users can use our source code and build their own studies. For instance, since the write operations presented results that maybe would not be the expected for the most of the people reading this study, some changes could be introduced and data could be written into the database through other techniques that could enhance the final results.

The source code is available together with OpenDataHub source code and can be downloaded by anyone.

OpenDataHub

OpenDataHub until now has just SustData as use case but was designed to sustain any kind of dataset which different collections.

However, at this point, in order to publish a new dataset, it would be necessary to install another project and all its dependencies and manage its data. All this is not pratical or easy to do.

[47] M-ITI, Madeira Interactive Technologies Institute, http://www.m-iti.org/ [last accessed 12-12-2015]

As future work we visualize a more abstract layer consisting on a dataset search engine where users could search for different datasets and access its details. Where this details area would be exactly or at least very similar to SustData presentation. In this way, any researcher could easily create as many datasets with as many collections as desired.

Having users' activity log, it would be interesting to build a dashboard area where data providers could find which operations data users perform more often over the published data, which kind of queries, which kind of users, what is the geographical location of the users interested on their data, and many other metrics. Basically it would be an analytics service for public scientific data.

References

[1] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame, “Data Sharing by Scientists: Practices and Perceptions,” *PLoS ONE*, vol. 6, no. 6, p. e21101, Jun. 2011.

[2] Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. Pratim, T. Marie, J. Fern, and O. Pereira, “The ORCHESTRA Collaborative Data Sharing System,” *SIGMOD Rec.*, p. 2632, 2008.

[3] H. van der Kuipers,Tom, “Insight into digital preservation of research output in Europe,” Report D3.4.

[4] M. I. Nrusimham Ammu, “Big Data Challenges.”

[5] L. Pereira, F. Quintal, R. Gonçalves, and N. Nunes, “SustData: A Public Dataset for ICT4S Electric Energy Research,” in *Proceedings of the 2nd International Conference on ICT for Sustainability*, 2014.

[6] F. Quintal, N. J. Nunes, A. Ocneanu, and M. Berges, “SINAIS: Home Consumption Package: A Low-cost Eco-feedback Energy-monitoring Research Platform,” in *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, New York, NY, USA, 2010, pp. 419–421.

[7] “About UCI Machine Learning Repository.” .

[8] “Open Science Data Cloud (OSDC) Documentation.” .

[9] “About Data.gov,” *Data.gov*. [Online]. Available: http://www.data.gov/about. [Accessed: 24-Dec-2015].

[10] “Understanding SQL And NoSQL Databases And Different Database Models,” *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models. [Accessed: 19-Sep-2015].

[11] “Introduction to Database Technology and DBMS.” .

[12] “Database model,” *Wikipedia, the free encyclopedia*. 07-Aug-2015.

[13] “NoSQL Databases Defined & Explained,” *Planet Cassandra*. [Online]. Available: http://www.planetcassandra.org/what-is-nosql/. [Accessed: 03-Oct-2015].

[14] “What is NoSQL?” MongoDB, Inc.

[15] “Wide Column Stores,” *DB-Engines*. .

[16] F. Stroud, “Top 10 Enterprise Database Systems to Consider in 2015.” .

[17] “Hadoop,” *Apache Hadoop*, 10-Sep-2015. .

[18] “Novell MySQL Administration Guide,” 09-Nov-2009. .

[19] V. Gananathan, “Structured Query Language,” *Advantages & Disadvantages*. .

[20] “ACID Properties of Sqlserver 2005,” *Sqlserver Blog*. .

[21] “Databases, ACID Compliance, NoSQL, and More.” [Online]. Available: http://www.idmworks.com/blog/entry/databases-acid-compliance-nosql-and-more. [Accessed: 11-Oct-2015].

[22] “What is Java? Server-Side Programming Language - Android - Object-Oriented Programming,” *Hiring | Upwork*. [Online]. Available: https://www.upwork.com/hiring/development/the-java-platform/. [Accessed: 24-Oct-2015].

[23] A. Ateeque, “What are the pro and cons of using Java compared to other server side programming languages?,” 15-Oct-2015. .

[24] “What is Python? - Definition from WhatIs.com,” *SearchEnterpriseLinux*. [Online]. Available: http://searchenterpriselinux.techtarget.com/definition/Python. [Accessed: 30-Oct-2015].

[25] P. Christensson, “Python,” *techterms*, 15-Jun-2010. .

[26] “A developer’s guide to the pros and cons of Python,” *InfoWorld*, 24-Feb-2015. [Online]. Available: http://www.infoworld.com/article/2887974/application-development/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html. [Accessed: 30-Oct-2015].

[27] “PHP,” *PHP*, 06-Sep-2015. .

[28] “What Are the Pros and Cons of PHP? (with picture),” *wiseGEEK*. [Online]. Available: http://www.wisegeek.com/what-are-the-pros-and-cons-of-php.htm. [Accessed: 02-Nov-2015].

[29] “Chrome V8,” *Google Developers*. [Online]. Available: https://developers.google.com/v8/intro. [Accessed: 02-Nov-2015].

[30] “Init.js: A Guide to the Why and How of Full-Stack JavaScript,” *Toptal Engineering Blog*. [Online]. Available: http://www.toptal.com/javascript/guide-to-full-stack-javascript-initjs. [Accessed: 24-Oct-2015].

[31] R. Posa, “Node JS Processing Model – Single Threaded Model with Event Loop Architecture,” Apr-2015. .

[32] “HTML,” *Wikipedia, the free encyclopedia*. 04-Nov-2015.

[33] “Cascading Style Sheets,” *Wikipedia, the free encyclopedia*. 29-Oct-2015.

[34] “JavaScript,” *Wikipedia, the free encyclopedia*. 06-Nov-2015.

[35] “jQuery,” *jQuery*, 06-Nov-2015. .

[36] “Introduction - Material design,” *Google design guidelines*. [Online]. Available: https://www.google.com/design/spec/material-design/introduction.html#introduction-goals. [Accessed: 14-Feb-2016].

[37] E. McNulty, “SQL vs. NoSQL- What You Need to Know,” *Dataconomy*. .

[38] “XAMPP,” *Wikipedia, the free encyclopedia*. 28-Aug-2015.

[39] “Aggregation Concepts — MongoDB Manual 3.0,” *https://github.com/mongodb/docs/blob/master/source/core/aggregation.txt*. [Online]. Available: http://docs.mongodb.org/manual/core/aggregation/. [Accessed: 07-Sep-2015].