

Hermes: Distributed social network monitoring system

Daniel Cea Royes

June 2014

BACHELOR THESIS

Department of Computer Architecture

Universitat Politècnica de Catalunya

Director: Marc Solé Simó (CA Labs)

Reporter: Jordi Nin Guerrero (UPC)

Preface

English

Nowadays, social network services play a very important role in the way people interact with each other and with the world. This generates big amounts of data that can be used to study social relationships and extract useful information about preferences and trends.

When analysing this information, two main problems emerge: The need to aggregate different data coming from multiple sources, and hardware limitations due to the incapability traditional systems have to deal with large amounts of data. In order to solve the problems mentioned before, this project aims to implement a distributed, scalable social media analysis tool, ready to connect and gather data from multiple sources and show the aggregated results in real-time.

Español

Hoy en día, las redes sociales juegan un papel muy importante en la manera como las personas interactúa entre ellos y con el mundo. Esto genera grandes volúmenes de información que pueden ser utilizados para estudiar las relaciones sociales y extraer información útil acerca de gustos y tendencias.

Cuando se analiza esta información, surgen dos problemas principales: La necesidad de agregar diferentes datos provenientes de múltiples fuentes, y las limitaciones hardware por la incapacidad de los sistemas tradicionales de manejar grandes cantidades de datos. Para poder solventar estos problemas, este proyecto propone implementar una herramienta de análisis de redes sociales distribuida y escalable, preparada para conectarse y recolectar datos de múltiples fuentes y mostrar los resultados agregados en tiempo real.

Català

Avui en dia, les xarxes socials juguen un paper molt important en la manera com les persones interactua entre ells i amb el mon. Això genera grans quantitats de dades que poden ser utilitzats per estudiar les relacions socials i extreure informació útil sobre gustos i tendències.

Quan s'analitza aquesta informació, sorgeixen dos problemes principals: La necessitat de agregar diferents dades provinents de múltiples fonts, i les limitacions hardware per la incapacitat dels sistemes tradicionals de gestionar grans quantitats de dades. Per poder solucionar aquests problemes, aquest projecte proposa implementar una eina d'anàlisi de xarxes socials distribuïda i escalable, preparada per connectar-se i recollir dades de múltiples fonts i mostrar els resultats agregats en temps real.

Acknowledgment

Thanks to my mom for being the support in my failures and the celebration in my achievements.

Thanks to CA Labs and Universitat Politècnica de Catalunya for providing everything I needed to develop this project.

Finally, thanks to Jordi, Victor and Marc for their teaching and support.

Contents

Preface	1
Acknowledgment	3
1 Introduction	13
1.1 Problem formulation	13
1.2 Project scope	14
1.3 Thesis organization	15
2 Analysis	16
2.1 Objectives	16
2.2 Risks	17
2.3 Stakeholders	18
2.3.1 Developers	18
2.3.2 Project Director	18
2.3.3 Project Tutor	19
2.3.4 Project Manager	19
2.3.5 Universitat Politècnica de Catalunya	19
2.3.5.1 Facultat d'Informàtica de Barcelona	19
2.3.5.2 Computer Architecture Department	19
2.3.5.3 CIT UPC	20
2.3.6 Barcelona Supercomputing Center	20
2.3.7 CA Technologies	20
2.3.7.1 CA Labs	20
2.3.8 End clients	20

2.4	Project description	21
2.4.1	User Interface	21
2.4.2	Master node	22
2.4.3	Crawling nodes	22
2.4.4	Database nodes	23
3	State of art	24
3.1	Social networks	24
3.1.1	Introduction to social networks	25
3.1.2	Trends and applications of social networks	25
3.1.3	Issues of social networks	27
3.1.4	Current social networks	28
3.2	Social media analysers	29
3.2.1	Introduction to social media analysers	30
3.2.2	Current social media analysers	30
3.3	Technologies	32
3.3.1	Data source technologies	33
3.3.1.1	Twitter API	33
3.3.1.2	Foursquare API	35
3.3.1.3	Instagram API	35
3.3.1.4	Google Maps API	35
3.3.2	Data gather and process technologies	36
3.3.2.1	Node.js	36
3.3.2.2	Tornado	36
3.3.2.3	Spring MVC	38
3.3.2.4	Play	38
3.3.3	Data storage technologies	39
3.3.3.1	Couchbase	40
3.3.3.2	MongoDB	41
3.3.3.3	Cassandra	42

3.3.3.4	Redis	43
3.3.3.5	Neo4j	43
3.3.3.6	HBase	44
4	Implementation	45
4.1	Data Access Layer	45
4.1.1	Proxy Layer	47
4.1.2	ElasticSearch Layer	47
4.1.3	Couchbase Layer	50
4.2	Business Logic Layer	50
4.2.1	Master Node	51
4.2.2	Crawling Node	52
4.2.2.1	Device enricher	54
4.2.2.2	Geo enricher	54
4.2.2.3	Spanish autonomous communities enricher	54
4.2.2.4	Gender enricher	55
4.2.2.5	Stopwords enricher	55
4.2.2.6	Stem enricher	56
4.2.2.7	Sentiment enricher	58
4.3	Display Interface Layer	59
4.3.1	Query interface	59
4.3.2	Host interface	60
4.3.3	History interface	61
4.3.4	Results interface	62
5	Testing	64
5.1	Data Access Layer tests	64
5.2	Business Logic Layer tests	65
5.2.1	Master Node	65
5.2.2	Crawling Node	66
5.3	Display interface layer tests	67

<i>CONTENTS</i>	7
5.3.1 Query interface tests	67
5.3.2 Host interface tests	68
5.3.3 History interface tests	69
5.3.4 Results interface tests	70
6 Results	71
6.1 Use case 1: UEFA Champions League Final	71
6.1.1 Scenario	71
6.1.2 Global results	72
6.1.3 Results by time	76
6.1.4 Conclusions	78
6.2 Use case 2: Miley Cyrus concert in Barcelona	80
6.2.1 Scenario	80
6.2.2 Global results	80
6.2.3 Concert only results	85
6.2.4 Spain results	90
6.2.5 Conclusions	92
7 Planning	97
7.1 Methodology	97
7.2 Schedule	97
7.3 Action plan	98
7.3.1 Project planning	98
7.3.2 Project analysis	99
7.3.3 Project development	99
7.3.4 Project maintenance	100
7.4 Deviations	103
8 Budget	104
8.1 Budget Analysis	104
8.2 Budget Estimation	104
8.2.1 Human resources	105

8.2.2	Hardware	105
8.2.3	Software	106
8.2.4	Other expenses	106
8.2.5	Total budget	106
8.3	Budget control	107
8.4	Deviations	107
9	Conclusions	109
9.1	General conclusions	109
9.2	Personal conclusions	110
9.3	Future work	110
10	Social and environmental impact	112
10.1	Social impact	112
10.2	Economic impact	113
10.3	Environmental impact	113
A	System classes	124
B	Dashboard	137
C	Paper	147

List of Figures

3.1	Radian6 dashboard	31
3.2	Sysomos dashboard	31
3.3	Trackur dashboard	32
3.4	Schema of Twitter REST API	34
3.5	Schema of Twitter Stream API	34
3.6	Execution time VS CPU consumption during a test Node.js VS Apache+PHP	37
3.7	Execution time VS memory consumption during a test Node.js VS Apache+PHP . .	37
3.8	Tornado performance under different contexts	38
3.9	Comparison between non-relational database model VS relational database model	40
3.10	Replica-set example in MongoDB	42
4.1	Schema of the data layer architecture	46
4.2	Schema of the business logic layer	51
4.3	Flowchart of Lancaster stemming algorithm	57
4.4	Schema of the display interface layer	59
4.5	Schema of the query interface	60
4.6	Schema of the host interface	61
4.7	Schema of the history interface	62
4.8	Schema of the results interface	63
6.1	Tweet count, Use Case 1	72
6.2	Devices used, Use Case 1	74
6.3	Sentiment analysis, Use Case 1	74

6.4	Sentiment over time, Use Case 1	74
6.5	Arousal over time, Use Case 1	75
6.6	Tweet map, Use Case 1	75
6.7	Tweet and retweet count from worldwide tweets, Use Case 2	81
6.8	Devices used from worldwide tweets, Use Case 2	81
6.9	Language from worldwide tweets, Use Case 2	81
6.10	Sentiment analysis over time from worldwide tweets, Use Case 2	83
6.11	Arousal over time from worldwide tweets, Use Case 2	83
6.12	Tweet map from worldwide tweets, Use Case 2	84
6.13	Sentiment analysis from worldwide tweets, Use Case 2	84
6.14	User's gender from worldwide tweets, Use Case 2	84
6.15	Tweet and retweet count from tweets about the gig, Use Case 2	85
6.16	Devices used from tweets about the gig, Use Case 2	86
6.17	Language of tweets about the gig, Use Case 2	86
6.18	Sentiment over time from tweets about the gig, Use Case 2	88
6.19	Arousal over time from tweets about the gig, Use Case 2	88
6.20	Tweet map from tweets about the gig, Use Case 2	89
6.21	Sentiment analysis from tweets about the gig, Use Case 2	89
6.22	User's gender from tweets about the gig, Use Case 2	89
6.23	Tweet count from Spanish tweets, Use Case 2	90
6.24	Devices used from Spanish tweets, Use Case 2	91
6.25	Language from Spanish tweets, Use Case 2	91
6.26	Sentiment over time from Spanish tweets, Use Case 2	92
6.27	Arousal over time from Spanish tweets, Use Case 2	93
6.28	Tweet map from Spanish tweets, Use Case 2	93
6.29	Sentiment analysis from Spanish tweets, Use Case 2	94
6.30	User's gender from Spanish tweets, Use Case 2	94
7.1	Planning and feedback loops in extreme programming	98
7.2	Planning Gantt chart from February 17 to April 20	101

7.3 Resources diagram from February 17 to April 20 101

7.4 Planning Gantt chart from April 21 to June 17 102

7.5 Resources diagram from April 21 to June 17 102

B.1 Query interface 138

B.2 Query interface, language view 139

B.3 Query interface, time view 140

B.4 Query interface, location view 141

B.5 Host interface 142

B.6 Host interface, slot info view 142

B.7 Host interface, stop stream modal 143

B.8 History interface 144

B.9 History interface, delete query modal 145

B.10 Results interface, part 1 145

B.11 Results interface, part 2 146

List of Tables

6.1	Top 10 most common words, use case 1	73
6.2	Top 10 most common hashtags, use case 1	73
6.3	Top 10 most common words from worldwide tweets, use case 2	82
6.4	Top 10 most common hashtags from worldwide tweets, use case 2	82
6.5	Top 10 most common words from tweets about the gig, use case 2	86
6.6	Top 10 most common hashtags from tweets about the gig, use case 2	87
6.7	Top 10 most common words from Spanish tweets, use case 2	91
6.8	Top 10 most common hashtags from Spanish tweets, use case 2	91
8.1	Human resources budget	105
8.2	Hardware budget	105
8.3	Software budget	106
8.4	Other expenses budget	106
8.5	Total budget	107

Chapter 1

Introduction

This project is carried out as the bachelor thesis of the Informatics Engineering degree at Universitat Politècnica de Catalunya during the spring semester of 2014, and it is enclosed inside a collaboration agreement within the company CA Labs.

1.1 Problem formulation

Nowadays, a growing proportion of human activities, such as social interactions, job relationships, entertainment, collaborative working, shopping, and in general, gathering information, are mediated by social networks and web services. Such digitally mediated human behaviours can easily be recorded and analysed, fuelling the emergence of computational social science, new services such as tuned search engines, social recommender systems or targeted on-line marketing. Due to this, public and private sector actors expect to use big data analytics to aggregate all this information, extract knowledge from it, and identify value to citizens, clients and consumers.

According to one research study from the University of Maryland's [1], this growing allows Facebook, Twitter and other social media sites to create between 182,000 and 235,000 jobs in US and have added between \$12.19 billion and \$15.71 billion in wages and salaries. A similar study funded by Facebook finds that, in Europe, Facebook added a similar number of jobs (approximately 232,000). All these business opportunities have been possible thanks to the possibility to mine social media insights through development Application Programming Interface (API).

Platforms for social media analytics are proliferating rapidly nowadays (Radian6 [2], Trackur [3], etc.) However, most of these platforms are private initiatives and the ones that are freely available present important hardware restrictions and, therefore, limitations to perform complex queries.

This project is born in order to overcome the aforementioned limitations, aiming to create a distributed, elastic and scalable social media analytics application. Our architecture enables clients to query different social networks with different queries at the same time and display, in real time, the aggregated mining results.

1.2 Project scope

The project scope has three different parts. The first one, a web interface that allow users to:

- Set a new query where geolocation, language and word filters can be added.
- Set the end time for the new query to stop gathering information and free the slot automatically.
- Initiate the query to start gathering information from social networks.
- Review the status of the system crawling slots, see how many available slots are and the running queries in the busy ones.
- Review a history of all queries executed in the past.
- Retrieve information about past queries, including start time, end time and filters used.
- Delete past queries.

The second part consists on another web interface that allows users to:

- See the aggregated results for one or more queries in real-time.
- Set the update time for new results to be displayed.
- Set time filters.

- Perform querying and filtering over the results using Lucene syntax [4].
- Add, modify and delete the way information is displayed.
- Provide a way to store the custom interfaces created by users.

Finally, both parts are controlled by a web server responsible of:

- Listen Hypertext Transfer Protocol (HTTP) requests and render both user interfaces.
- Control the interactions of the users with the system.
- Store queries and the data gathered by them.

1.3 Thesis organization

The rest of this document is organised as follows: Firstly, we list the project objectives and risks, analyse the scope and stakeholders and introduce state of the art, where the current status of technologies and similar projects are described. Secondly, a detail of the implementation process for the system, followed by the tests realized to ensure correctness and the results obtained from two different use cases. Finally, we talk about the planning and budget, explain the project conclusions, and detail the social and environmental impact of the system.

Chapter 2

Analysis

Once formulated the problem and the scope of the project, in this chapter we analyse into more detail the system. This includes, firstly, listing the objectives and risks related to its application; secondly, describing all the stakeholders related to this project; and lastly, giving a brief explanation of all the parts that implemented in it.

2.1 Objectives

The main objectives that we must ensure our system fulfil once it is finished are:

- Automatic: The system must handle disconnections and errors, being able to stop automatically if needed, and retrieving, processing and storing the data without human intervention. Once a query is started, user does not need to worry anymore.
- Reliable: Because no human intervention is a must, the system must be extremely reliable, so the system does not stop unless critical error occurs. This includes handling error messages and disconnections, non-vital node failures, and network problems.
- Scalable: The system must be fully scalable, following a "Hot-plug" philosophy. This includes not only adding or removing database or system nodes, but also new data sources, without affecting the system performance and distributing the workload between the old nodes and the new ones.

- Usable: We must provide a Software as a Service (SAAS) software model that can be accessed anytime, anywhere, from any device, and because we expect no technical knowledge from the users, it must be simple and intuitive enough to have an almost flat learning curve, not requiring any special course or training, and displaying the result data in a friendly, visual, graphical way.
- Real-time: The system must be able to update itself in real-time based on events occurring. This includes updating the interfaces with the nodes status, and the result interface with the new indexed data. This way, the user can be sure he is looking at the latest information anytime.

2.2 Risks

Every new software product has to assume some risks even before starting, and this project is not an exception. There are some important weak points that must be taken into account at all times if we want to achieve a product that meets the previous requirements and objectives, and thus they have been divided into five main risks:

- Use of the newest technologies: Riding high with the latest available technologies allows us to make a faster, more distributed and more reliable system, but has also some downsides. The testing period of those technologies has not been as long as other legacy ones, so they may contain bugs. Besides, usually documentation is poorer, and problem solving is harder because less people have expertise in them.
- Third-party dependence: The entire project depends on the quality of the data source. We must ensure that the data source is deep enough, and it does not have limitations too restrictive. Also, we must be prepared to add, modify or remove the different data sources anytime, as no one can ensure they remain active and unchanged forever.
- High maintenance cost: Another problem related to the third-party dependence. As we mentioned before, data providers may change the accessing rules or the data format anytime, so our project may need an update to meet the new requirements.

- Hardware limitations: This problem affects specially the database nodes, because making them scalable, distributed, and being able to turn them on and off without affecting the system causes a big overhead which makes the system not worthy for small amounts of data, where old-style relational databases are much more effective. This overhead makes it harder to implement our solution using old hardware.
- Software licences: Is important to ensure that all the software used is open source, but we have to be careful that no library, product or system we use comes under a licences that force the software developed under them to be also open source, like GNU, because this is an end-product that may be used for private purposes in the future.

2.3 Stakeholders

Project stakeholders are entities that have an interest in a given project, and may have a positive or negative influence in the project completion. Every stakeholder related to this project is described below.

2.3.1 Developers

I, Daniel Cea, student at Facultat d'Informàtica de Barcelona, am the only developer for this project. This means I am responsible to analyse, develop, maintain and test the whole system. Also, I am responsible for the installation, configuration and maintenance of all the software used in servers (but not of the servers themselves, responsibility of Computer Architecture Department), and the software and hardware of the develop laptop (including maintenance of the Operative System (OS), software, etc.)

2.3.2 Project Director

The project director is Marc Solé, lecturer in the Computer Architecture Department of the Universitat Politècnica de Catalunya. His role as director is to advise and assist me during the whole project analysis and development.

2.3.3 Project Tutor

The project tutor is Jordi Nin, lecturer in the Computer Architecture Department of the Universitat Politècnica de Catalunya. His role as a data analytics expert is to help during the analysis, development and testing processes, and also he is responsible of communicating with the project manager, with Barcelona Supercomputing Center, and with Computer Architecture Department.

2.3.4 Project Manager

The project manager is Victor Muntés Mulero, vice president and director of CA Labs (R&D section) based in Barcelona and, prior to joining CA Labs, associate professor at Universitat Politècnica de Catalunya. His role, as CA Labs director, is to analyse the requirements for this project, communicating with external people from CA Technologies, dealing with the legal issues, and everything related with the budget (salaries, hardware acquisitions, taxes, etc.).

2.3.5 Universitat Politècnica de Catalunya

The Universitat Politècnica de Catalunya in Barcelona, Spain, is where this project is presented as a bachelor thesis. There are several stakeholders contained within UPC.

2.3.5.1 Facultat d'Informàtica de Barcelona

The teachers affiliated with this school are the ones responsible of evaluating this bachelor thesis. Also, they were responsible to make the first contact between me and Jordi Nin so we could engage this project.

2.3.5.2 Computer Architecture Department

I am working in their building, and they provide me with the needed hardware for this project. Also, they are the ones helping with the development, not only Jordi Nin, but also other teachers from the department may provide their help occasionally.

2.3.5.3 CIT UPC

CIT UPC (Centre d'Innovació i Tecnologia) is a department belonging to UPC acting as intermediary between students and private companies, in this case, me and this project by CA Labs. The funds are given to CIT UPC so they can grant this project.

2.3.6 Barcelona Supercomputing Center

Barcelona Supercomputing Center hosts the biggest supercomputer in Spain. Due to an agreement with UPC, they provide me with the hardware for the production deployment using part of the Minerva cluster.

2.3.7 CA Technologies

CA Technologies is an enterprise software company based in Islandia, New York. His most successful products are business-to-business mainframes and distributed information technology infrastructure applications. They are aiming to improve their knowledge about cutting the edge technologies related with big data analytics, and a good starting point for that is to create a social data analyser.

2.3.7.1 CA Labs

CA Labs is an external company acting as the R&D department of CA Technologies. They receive the funds from CA Technologies to engage the European project LeanBigData, and part of that budget is designated to CIT UPC to hire a student to join them temporary.

2.3.8 End clients

The direct end client of this project is CA Technologies, but also all the companies included in the partnership of the European project LeanBigData. This project does not have any end users apart from these companies, and is not released as a product to the general public.

2.4 Project description

Our project is divided into four different parts:

2.4.1 User Interface

The user interface is responsible of providing the end-user control over the system, including setting the query based on different filters, starting and stopping the crawling nodes, retrieving a query history from the database, and display in real time all the statistics related to an actual or a past query.

This interface is entirely web-based and following a responsive design, which means that users are able to access from any device provided with a web browser (smartphones, tablets, laptops, computers...). It is divided in four different sections:

- Query Interface: In charge of starting the data harvesting based on three different filters: words contained within the message, language of that message, or location where it was sent from. Also, there is control on setting the time when the data gathering must stop automatically.
- Host Interface: In charge of controlling the crawling nodes. Here, the user sees all the information about each crawling node. If that node is busy collecting data, the interface shows the information about the filters that are being used, and also lets the user to stop the stream.
- History Interface: In charge of showing the query history and letting the user to show the results of each one, or delete them. In case a deleted query is still running in one of the crawling nodes, it is stopped.
- Result Interface: In charge of displaying the aggregated results of one or more queries. The information is represented using char pies, bar graphics, histograms and maps.

2.4.2 Master node

The master node is the main point of the project, acting as a message router between the input interface and the other nodes. Its functions include:

- Answer the HTTP GET queries from the client, rendering the adequate interface.
- Keep a live connection with each client, updating the interface in real-time to reflect the changes from the broadcast messages sent by database and crawling nodes.
- Communicate the user input to the crawling nodes (start/stop queries) or the database nodes (create/retrieve/delete query).

2.4.3 Crawling nodes

The crawling nodes are responsible of requesting and managing the data from external sources. Unlike the previous nodes, there can be more than one crawling node, and they can be added and removed dynamically and easily. Its functions include:

- Authenticate with the external API.
- Receive the query filters from the master node and initiate the data harvesting based on them.
- Process the raw response through the different enrichment modules.
- Send the processed response to the database nodes.

There may be several enrichment modules, each one responsible of one concrete topic, and this concrete project has the following ones:

- Device enricher: Determines the device used to write the message.
- Spain enricher: For messages coming from Spain, determines the autonomous community.
- Stop words enricher: Remove all stop words. Needed to perform sentiment analysis.

- Stemmer enricher: Applies a stem to the prior filtered words. Needed to perform sentiment analysis.
- Sentiment enricher: Determines the sentiment and arousal of the previous stemmed tweet in a scale of 1 (sad/calm) to 9 (happy/excited). It only works for messages in Spanish and English.
- Geo enricher: Filters messages by geolocation, removing those outside a specified bounding box.
- Gender enricher: Guesses the gender of the user who sent the tweet based on the full name of his/her profile.

2.4.4 Database nodes

The database nodes are responsible of storing and indexing the data. They are running a scalable, distributed, NoSQL document-based database, with one of them acting as master and the rest as slaves.

As the previous nodes, they may be added and removed dynamically, depending on the needs of search/indexing speed (more/less CPU) and space (more/less memory).

Chapter 3

State of art

From the previously introduced ideas, we conclude that we need certain knowledge about the current social media monitoring tools in order to continue with our project, to ensure its novelty and originality. We need to investigate the current solutions in social data analysis, and what are their strengths and lacks.

To do that, first we give a formal definition of social networks, their most interesting applications but also their negative aspects, and the most important social networks nowadays.

Secondly, we talk about the current social media analysers in the market, remarking how they process, enrich and store the data, and what are the most common problems and challenges they are facing.

An important factor for this project is the capability of managing enormous amounts of data quickly enough, which makes necessary to introduce technologies focused on big data processing and storing. That is why, in that direction, the last part of this chapter studies the different technologies existing for big data, including non-relational databases and big data collectors.

3.1 Social networks

Before describing social media monitoring tools, first we have to talk about social networks, the root of any software tool specialized in social analysing due to their capability to provide big amounts of data from a lot of people. Therefore, it is interesting to deeply describe what are the pros and cons of social networks, and decide which social networks we should focus on, as the

one with fewer users do not report us enough data to be worth of implementation.

3.1.1 Introduction to social networks

A social network, understood as the networking service and not as the theoretical concept of relationships between people, is a platform to build those social relations mentioned before among people who share interests, activities, backgrounds or real-life connections. Consist of a representation of each user (often called a "profile"), his social links, and a variety of additional services. Most current social network services are web-based and provide means for users to interact over the Internet. This includes create a public profile, create a list of users with whom to share connection, and being able to view and cross messages, photo and video within the system. [5]

3.1.2 Trends and applications of social networks

While the popularity of social networking increases, new uses for the technology are frequently being observed. There are two main points that make social networks different and novel respect legacy online communities: Real-time, the fact that content contributed by users which is broadcasted as is being uploaded as it happens with to TV or radio, and the attached metadata, the fact that every message, video and photo has multiple related content (geolocation, user, language, etc.) that opens a wide range of opportunities to be analysed in multiple ways.

Among the newest trends and applications for social networks, we can highlight the following ones:

- Science: Researchers use social networks to communicate research results with the science community, and also as a public communication tool with scientists sharing the same professional interests and fields [6]. Social networks like LinkedIn, Facebook or Researcher Gate give the possibility to join professional groups and pages, share papers and results, publicise events, discuss issues and create debates.
- Education: Educators and advocates of new digital technologies are confident that social networking encourages the development of transferable, technical, and social skills of

value in formal and informal learning [7]. Learners use social networks to improve professional and curriculum education, giving new added values and making the apprenticeship easier. For example, the use of online social networks by school libraries is increasingly prevalent and they are being used to communicate with potential library users, as well as extending the services provided by individual school libraries.

- Employment: A recent trend in social network use is being driven by college students using the services to network with professionals for internship and job opportunities. Many studies have been done on the effectiveness of networking online in a college setting, and one notable one is by Phipps Arabie and Yoram Wind published in *Advances in Social Network Analysis* [8]. LinkedIn is a notable example, being a social network fully focused in work relationships.
- Trading: Many social networks allow selling and buying products, which has arisen as a convenient way for many companies to sell their products or services, and for many users to buy, sell and exchange second hand objects. eBay is a good example.
- Social movements: Social networks are being used by activists as a means of low-cost grassroots organizing. The most notorious and recent example was the Egyptian revolution of 2011, where Facebook and Twitter both played a pivotal role in keeping people connected to the revolt, and Egyptian activists credited social networking sites as a platform for planning protest and sharing news from Tahrir Square in real time [9] [10]
- Health: Social networks are beginning to be adopted by health care professionals as a means to manage institutional knowledge, disseminate peer to peer knowledge and to highlight individual physicians and institutions [11]. As an example, SoberCircle gives alcoholics and addicts in recovery the ability to communicate with one another and strengthen their recovery through the encouragement of others who can relate to their situation.
- Crowdsourcing: Crowdsourcing is a recent trend born to subdivide tedious work or to fund-raise startup companies and charities, and can also occur offline. Kickstarter is an example of social network where users can upload their projects or ideas and people can fund them if they like it.

3.1.3 Issues of social networks

Social networks are not exempt of downsides, some of them really dangerous. The most notorious are:

- Privacy: This has been a recurrent problem since social networks were born, and involves multiple factors. The main problem involves users giving too much personal information publicly, information that can be misused by the wrong people, like sexual predators, or simply a violation of the person's privacy from harmless but unwanted people [12]. Privacy problems also include social network owners selling private information to companies to improve their sales and profitability, creating customer profiles containing demographics and online behaviour [13].
- Unauthorized access: There are different forms where user data in social networks are accessed and updated without a user's permission, including bugs in the software, malicious third-party plugins or applications from the social network, virus and malware installed in the user's computer, or simply a user's neglect. This unauthorized access allows the wrongdoer to violate user's privacy, or update the profile and send messages without user's consent [14] [15].
- Risk for child safety: Citizens and governments have been concerned with misuse by child and teenagers of social networking services, in particular in relation to online sexual predators [16]. Overuse of social networking may also make children more susceptible to depression and anxiety [17].
- Cyber-bullying: Online bullying is a relatively common occurrence occurring specially between the youth. There are not many limitations as to what individuals can post when online. Individuals are given the power to post offensive remarks or pictures that could potentially cause a great amount of emotional pain for another individual, resulting in emotional trauma for the victim.
- Interpersonal communication: As social networking sites have risen in popularity over the past years, people have been spending an excessive amount of time on the Internet in

general and social networking sites in specific. This has led researchers to debate the establishment of Internet addiction as an actual clinical disorder [18].

3.1.4 Current social networks

To end our social network analysis, we give a brief overview to those social networks with enough impact to be used, to point out those that produce the best data for our system:

- LinkedIn: LinkedIn is a social network service born in May 2003 and completely oriented for professional networking [19]. It allows users to create a profile that works like a professional resume, then connect with other profiles (users or companies) to apply jobs, seek employees for an open position or keep contact with clients or companies. The data offered by LinkedIn, even not being user-oriented (it is allowed to look for data from any user or topic, not just the authenticated one), it is very specific, people does not share hobbies or talk about topics that are not related to work here, so the data obtained might be useful, but not as good as expected.
- Facebook: Currently the social network with more users, it was founded on February 2004 to offer Harvard students a way to communicate and share private messages and pictures between them [20]. Users must register before using the site, and then they create a personal profile, add other users as friends, and are able to add new content to their profile or review other friend's profiles. Even the data containing would be useful for our application, their API is very user-oriented, which means you can only get information from the logged user. Would be really interesting for studies like social graphs, but it is out of the scope of this project.
- Twitter: Twitter is a microblogging social network born in March 2006 that enables users to send and read "tweets", text messages limited to 140 characters [21]. Everyone is able to read those messages, but only registered users are able to post them. Their API is probably the one that fits best our needs: their Stream API allows to receive a big amount of tweets in real-time, and those tweets include a lot of metadata to be analysed.
- Foursquare: Foursquare is a location-based mobile social network founded in March 2009,

where users check in at venues from a list the application locates nearby, giving badges and points for each checked place. Those scores can be later compared with other friends [22]. Due to its millions of geolocated places and the open API, Foursquare data would be really useful for our application.

- Instagram: Instagram is a social network born in October 2010 focused on photo and video sharing. Users are able to take pictures and videos, apply one of the multiple digital filters that they offer, and share not only in Instagram, but also in other social networks like Twitter or Facebook [23]. The interesting part about Instagram is that their API offers a public stream of pictures containing a lot of useful metadata for the system.
- Google+: The second social network by users, founded in May 2013, aims to offer a social layer that enhances many of the online products Google already had. It works similarly to Facebook, allowing users to create a personal profile with friends and content, but users do not have to register specifically to the social network, an account it is automatically created with Google credentials [24]. Because it has similar features to Facebook, it also shares the same problem with its data: It is very user-oriented, so it is also out of the scope of this project.

Once analysed some of the most popular social networks, we conclude that the ones who allow retrieving information from all users are the only ones that fit in our needs. Therefore, we discard Facebook and Google+ as data sources. Due to time limitations, we implement our solution just using data coming from Twitter, because it offers the data that better fits our needs, but Instagram and Foursquare are also interesting options to consider in the future.

3.2 Social media analysers

Once established what is a social network, analysed their positive and negative points, and chosen Twitter as our data source, now it is time to talk about social network analysis software, first giving a brief overview about them, and then analysing the current solutions other companies are offering, focusing on their strengths and, specially, on the weaknesses we can exploit.

3.2.1 Introduction to social media analysers

The fast proliferation of social networks and consumer-generated content has created an emerging shift in power from traditional institutions and mainstream media to the consumer, which everyday generates massive amount of freely-available sentiment on the Internet.

For that reason, Social Network Analysis (SNA) software was born, facilitating quantitative or qualitative analysis of social networks by describing features of the data flowing that network either through numerical or visual representation. Those networks can consist of anything from families, classrooms, sport teams, social networks or even the Internet, but the bigger the network, the more data has to offer and, therefore, the more reliable the results.

3.2.2 Current social media analysers

Among the most popular social monitoring tools, we can find the next ones:

- Radian6: Radian6 belongs to the Exacttarget marketing cloud, a series of products from the company Salesforce aiming to improve marketing campaigns and brand position for companies [2]. This product is centered in identifying and analysing conversations about companies, products and competitors in order to route important insights to sales, customer service and community managers. The product allows aggregating and analysing data from multiples sources, but it has three downsides: It has a monthly fee, there is no sentiment analysis (only statistics about demographics, age, gender, etc.), and it is a closed product, which means that clients cannot modify or add new sources. A picture of Radian6 dashboard can be seen in figure 3.1.
- Sysomos: Sysomos is a product suite that provides customers with the tools to measure, monitor, understand and engage with the social media landscape [25]. Their solution covers a lot of social networks, numerous Facebook pages and Twitter accounts, and classify the information by sentiment, demographics, influence of the author, or text analytics. It shares a lot of features with Radian6, which makes it share also the same downsides: The high monthly fee and the fact of being a closed product. A picture of Sysomos dashboard can be seen in figure 3.2.



Figure 3.1: Radian6 dashboard

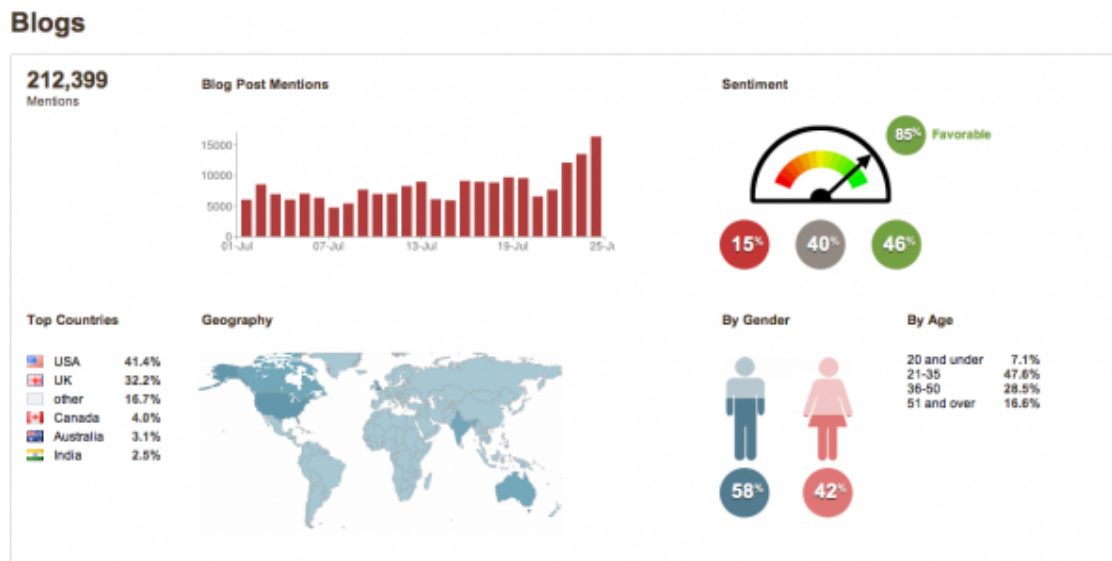


Figure 3.2: Sysomos dashboard



Figure 3.3: Trackur dashboard

- **Trackur:** Trackur is an online reputation and social media monitoring tool designed to track what is said on the Internet about people, organizations and companies [3]. They aggregate and analyse multiple sources, including news, videos and blogs, and they also have efficient and complete sentiment analysis tools. Their worst downsides are their poor input filter, allowing only to search by words, and the few output data displayed, only giving information about the source, influence and sentiment. Also, it has a monthly fee (with a free trial period). A picture of Trackur dashboard can be seen in figure 3.3.

3.3 Technologies

Finally, after analysing current online social networks (and, hence, the data sources) and describing other social media analysers, it is time to take a look at the technologies we can use to develop our system. We have divided the following section into three parts, which are the three mainstays of this project: Technologies allowing us to obtain the data (API and geolocation systems), technologies to gather and process that data, and technologies to store it.

3.3.1 Data source technologies

The data source technology used for every social network is even more important than the number of interactions per day the social network has, because it is useless to know that they have a lot of information if we cannot access it in real-time and with no restrictions. That is the reason why this section presents the main API services available to connect and retrieve social information from those social network services.

3.3.1.1 Twitter API

Currently, Twitter API is one of the most used for social data analytics in applications for phones, desktop and websites. With more than 500 million tweets being sent every day, this API provides a very useful data source for big data analytics in social networks [26]. It is divided into two different services:

- REST API: Based in HTTP protocol, all resources available are accessible thanks to a global Uniform Resource Identifier (URI) and exchanged between server and client using representations of those resources. Each request is stateless (there is no relationship between two different request), and the data is exchanged using JavaScript Object Notation (JSON) format. This API allows gathering information about timelines (collections of tweets, ordered with the most recent first), tweets, private messages, friendship relations, users, favourites, lists, places and trends. See figure 3.4 for a detailed schema of the Twitter REST API.
- Stream API: Also based in HTTP protocol, this API does not response to user request like the REpresentational State Transfer (REST) API, instead, the first HTTP connection sets a series of filters and remains open, and Twitter starts sending real-time tweets matching those filters as a response using a data stream. This API is really useful for real-time projects like this, the problem is that the filters are limited to only words, geoposition, users and language, and to avoid system saturation, Twitter does not ensure that all the fields of each tweet are always populated, sometimes they are sent empty and must be retrieved in the REST API using the tweet URI. See figure 3.5 for a detailed schema of the Twitter Stream API.

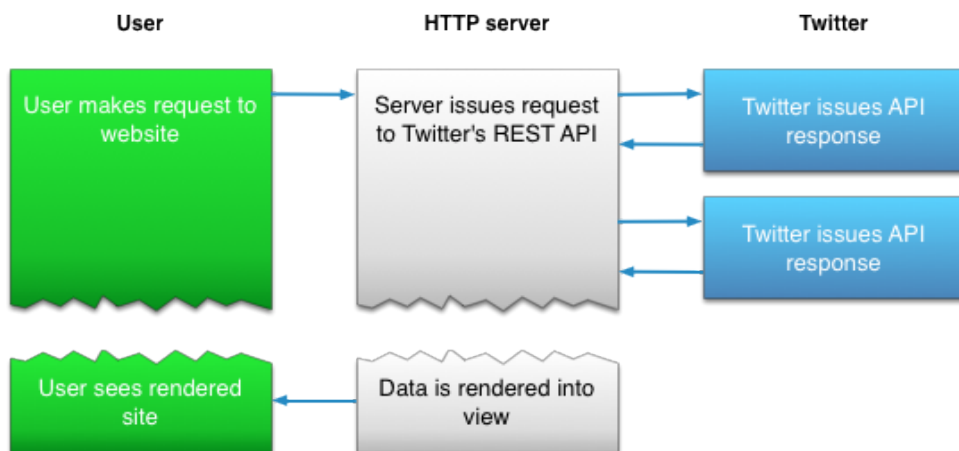


Figure 3.4: Schema of Twitter REST API

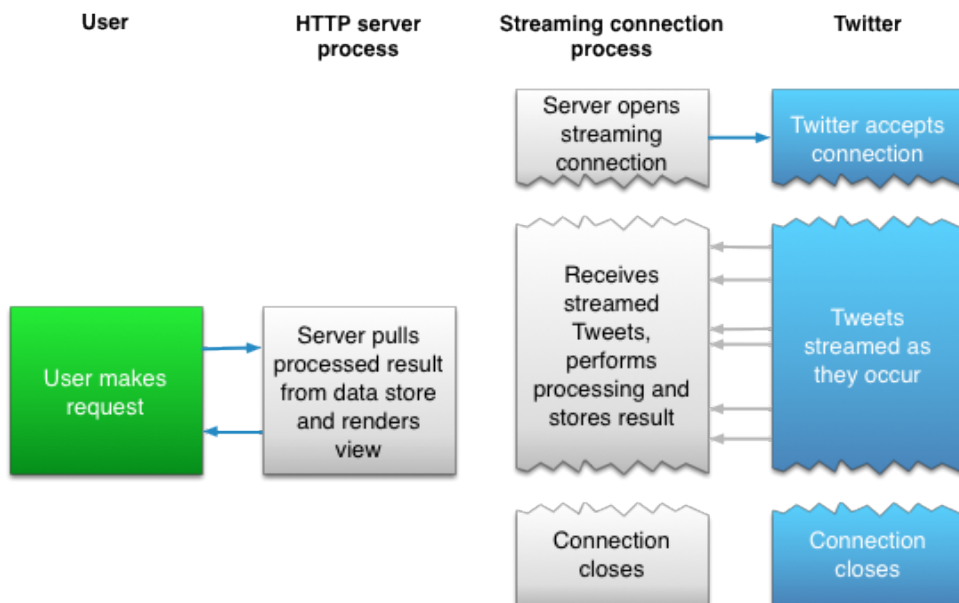


Figure 3.5: Schema of Twitter Stream API

3.3.1.2 Foursquare API

Foursquare API is based in a REST architecture and provides 4 different connection points: Core API, for the main operations, Business owner API, Business API and a Real-time API [27].

The Real-time API works different than the Twitter stream API, because it requires a server accessible through Internet with a Uniform Resource Locator (URL). Once the URL is registered into Foursquare systems, each new notification is sent to this server using HTTP POST requests in JSON format.

This API, despite being really useful, is not as good as the Twitter Streaming API for this project due to only providing geospatial and places information. Although this, it may be considered in the future to implement an enricher with precise geospatial information coming from this API.

3.3.1.3 Instagram API

Instagram API is also based on a REST architecture, and provides information about users, relationships, resources, comments, geoposition, photos and their tags [28].

As Foursquare and Twitter do, Instagram provides a Real-Time API with a method similar to Foursquare. A server accessible through Internet is required, and then this server initiates a subscription with a set of filters and Instagram returns HTTP POST requests in real-time. More than one subscription is allowed, and they have 4 different types: user, tags, location and geography. Those subscriptions can be listed and deleted using HTTP GET and HTTP DELETE methods.

This API is not implemented because of time limitations, but it may be considered a future implementation to enrich the stored data with pictures, tags and more related messages.

3.3.1.4 Google Maps API

Google Maps API offers geolocalization and geocodification through HTTP requests [29], limited to 50.000 requests per day. This API, even it does not provide information at its own, is really useful to enrich the geodata coming from our sources, or to provide easy interfaces to add filter location to our requests without asking the user for concrete latitude/longitude coordinates, which would be really unfriendly and unusable.

3.3.2 Data gather and process technologies

Currently, there are many frameworks that allow to develop applications capable of supporting big amounts of concurrent requests and wide streams of data without that affecting the performance of the global system, a very important feature when gathering and processing the data because it must, on the one side, capture different streams and analyse them and, on the other side, provide several input interfaces and real-time data display. Here we describe them.

3.3.2.1 Node.js

Node.js is a framework to create web applications using JavaScript in the server side following an event-oriented pattern. It works over the JavaScript Google Chrome engine [30] based on an asynchronous model with non-blocking request, offering high concurrency and, therefore, scalability.

A simple "Hello World" benchmark, compared to alternatives like Apache + PHP, shows that, even the memory consumed is higher, the system works way faster [31]. Figures 3.6 and 3.7 show a comparison between both systems of the CPU and memory usage for a thousand concurrent request and a total of 100.000 request. Here, we can appreciate that Node.js has a lower execution time with a little bit higher memory consumption.

The popularity of this framework is increasing exponentially during the last years, being used by companies like LinkedIn [32]. The community appears to be very active, developing new libraries every day supporting this asynchronous model, which ensures a wide spectrum of tools, support and documentation.

3.3.2.2 Tornado

Tornado is a non-blocking framework for web applications written in Python [33]. It was first developed by the company FriendFeed, which was bought by Facebook at 2009 and, the same year, was set free as open source project. As Node.js, it is a framework that stands out for his high performance for web applications that need to run with a large number of connections or big streams of data.

When Facebook liberated Tornado, they published a benchmark showing the performance

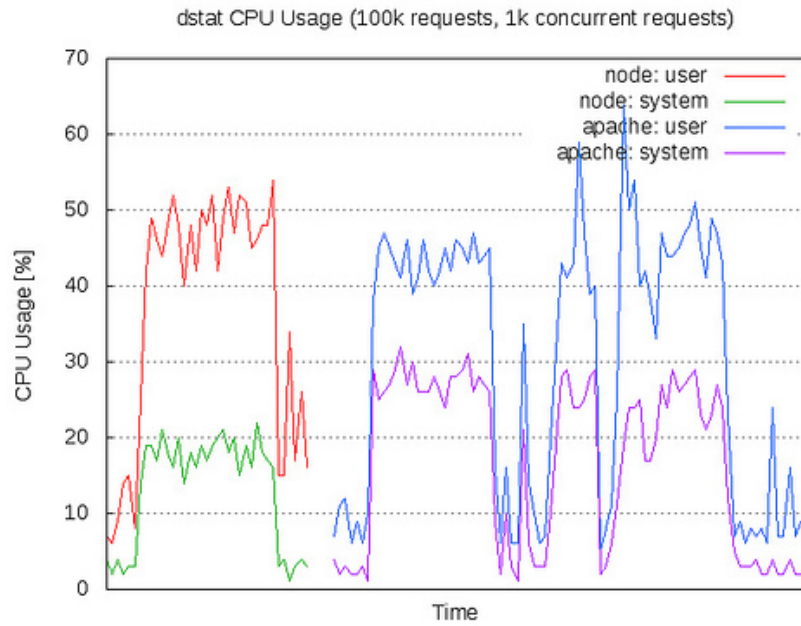


Figure 3.6: Execution time VS CPU consumption during a test Node.js VS Apache+PHP

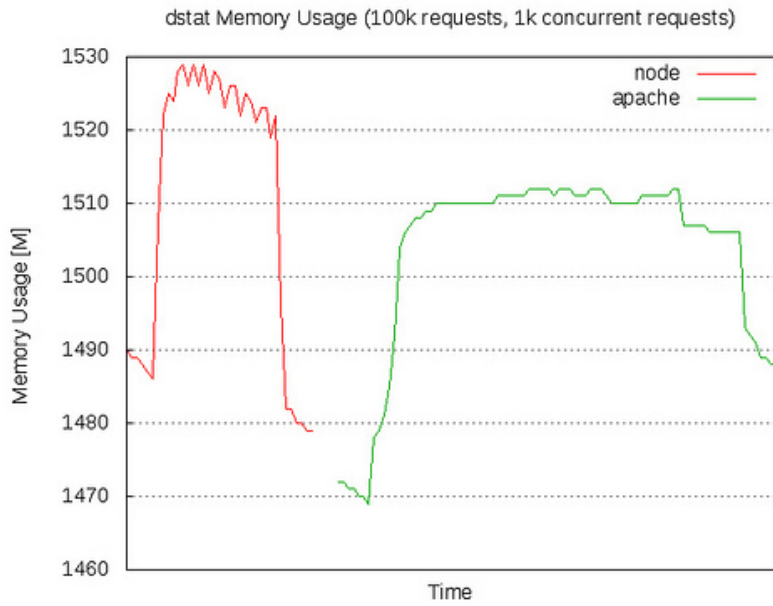


Figure 3.7: Execution time VS memory consumption during a test Node.js VS Apache+PHP

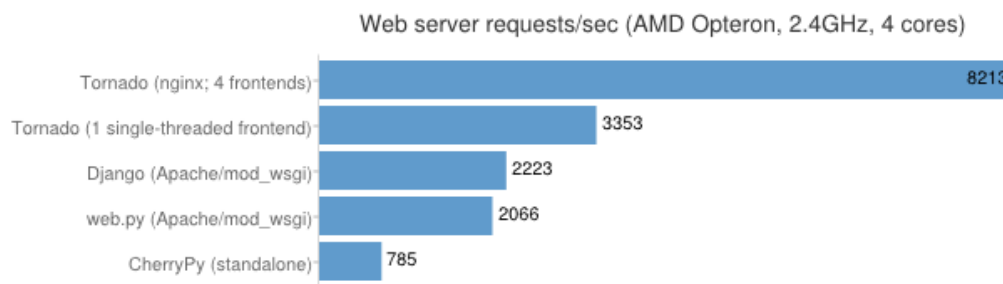


Figure 3.8: Tornado performance under different contexts

offered by Tornado in different contexts, as can be seen in Figure 3.8. They proved that, with a nice Tornado configuration, it is able to achieve 8000 request per second, much better than other frameworks also written in Python.

The main uses for this framework are the same as the ones described before for Node.js.

3.3.2.3 Spring MVC

Spring MVC is a component from the big framework Spring, one of the most used frameworks by Java developers. It is based in a Model View Controller (MVC) pattern, and being used for years by a lot of companies, has a wide community and an excellent support and documentation [34].

Spring offers multiple options to parallelize processes and also to develop multi-thread applications. Thanks to Spring Data, another part of the framework, the integration with non-relational databases like MondoDB, Redis or Neo4j is complete.

Another interesting point about Spring is that allows modularizing applications and running those modules in different hosts. There are many options to escalate applications using memory-caching systems or connecting those modules with messaging systems like Java Message Service (JMS).

Spring MVC may be very interesting when there is a need to connect different components at the same time, allowing developers to combine and switch those components easily and quickly depending on different configurations, environments or needs.

3.3.2.4 Play

Play is a web-oriented stateless framework written in Scala and Java and following a MVC pattern. It is inspired in other frameworks like Ruby on Rails (for Ruby) or Django (for Python),

and presume about offering high scalability thanks to their Input/Output (I/O) asynchronous model provided by Netty, a base-server able to attend a high number of concurrent HTTP request thanks to the reactor pattern, a thread-based pattern similar to Node.js [35]

Play is still in development, but so far it has been very well accepted thanks to its easiness to deploy web applications fast, prioritizing conventions over configuration, and to its lightness.

3.3.3 Data storage technologies

Currently, almost every database management system existent follows either a relational model or a non-relational model. On the one hand, the relational model organizes data using relations, defined by rows and columns and with the capability to interrelate with others using foreign keys. The structure of that relationships and other metadata are stored in the database schema, responsible of maintaining the consistency of the database status at all times [36]. About the non-relational models, the data model here is different and the information is usually aggregated using key-value pairs. Unlike the relational model, which is always structured using rows and columns, the non-relational databases may have different data models [37]:

- Key-value based
- Document based
- Column based
- Graph based

The main problem of maintaining a schema in relational models is that is rigid and very reticent to new changes. On the one hand, relational technologies require a defined relationship structure before starting to insert new data in the database, and changing those relationships is usually a heavy and hard process. On the other hand, non-relational models are more flexible because the schema allows changes anytime easily, but it is much harder to maintain the Atomicity Consistency Isolation Durability (ACID), the set of properties that guarantee that database transactions are processed reliably [38].

Moreover, another big difference is the way both models escalate. On the one hand, relational models, due to being centralized models, they escalate vertically ("scale up"), adding

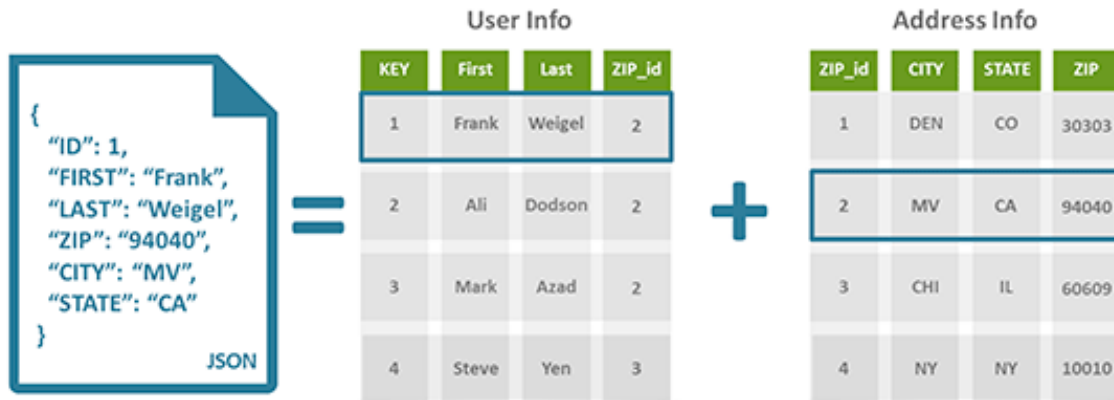


Figure 3.9: Comparison between non-relational database model VS relational database model

more resources to the system (CPU, memory and disc) in order to absorb the storage of new data and relationships. Maintaining those servers is expensive and complex. On the other hand, non-relational models were born to be distributed systems from the beginning, so they can scale both vertically and horizontally ("scale out"). When there is an increase of the data from the application, that system can scale by adding more resources, like we mentioned before, but also by adding new servers to the cluster, which makes it easier and cheaper to deploy and maintain rather than adding more and more hardware resources to a single server.

Once analysed the two different database models, we can conclude that non-relational databases fit better for our project. For that reason, here we describe the currently most used non-relational database management technologies, highlighting their main features.

3.3.3.1 Couchbase

Couchbase, originally Membase, is a document-based distributed non-relational database that was born first as a key-value. It is optimized for interactive applications with an elevated number of requests, and it is designed to escalate easily without much configuration needed.

Each node inside a cluster is equal to the others, and it has two components [39]:

- Cluster manager: It is responsible to supervise the configuration and operation of all the other nodes of the cluster, and also the replication and recovery in case of failure. This manager is written in Erlant/OTP, an ideal environment for distributed systems.

- Data manager: It is responsible to store and retrieve the data requested by the applications. All write operations are asynchronous, and the documents are stored in JSON format.

We choose this as our storage technology because, apart from the benefits described before, it has a stack of software specifically designed for Couchbase that make our system development much easier:

- ElasticSearch: ElasticSearch is a flexible and powerful open source, distributed, real-time search and analytics engine [40]. Architected from the ground up for use in distributed environments where reliability and scalability are must haves, ElasticSearch gives the ability to move easily beyond simple full-text search. Through its robust set of APIs and query DSL, plus clients for the most popular programming languages, ElasticSearch delivers a high-end search technology that becomes the cornerstone of this project.
- Kibana: Kibana is ElasticSearch's data visualization engine, allowing to natively interact with all the data in ElasticSearch via custom dashboards [41]. Kibana's dynamic dashboard panels are saveable, shareable and exportable, displaying changes to queries into ElasticSearch in real-time. Users can perform data analysis in Kibana's beautiful user interface using pre-designed dashboards or update these dashboards in real-time for on-the-fly data analysis.

3.3.3.2 MongoDB

MongoDB is an open-source, document-based database management system written in C++ that manages files in Binary JSON (BSON) format [43]. It has multiple drivers for different technologies, enabling multiple systems and languages. The system allows users to execute all kinds of queries, with ranks and regular expressions. It supports Map Reduce and geospatial indexation and is widely used in web environments because all queries are plain JavaScript expressions. The system offers a replica-sharding system. There are two replica systems:

- Master-slave system: Configures a master host where write and read operations happen. Each time master executes a write operation, the new data is copied to the slaves.

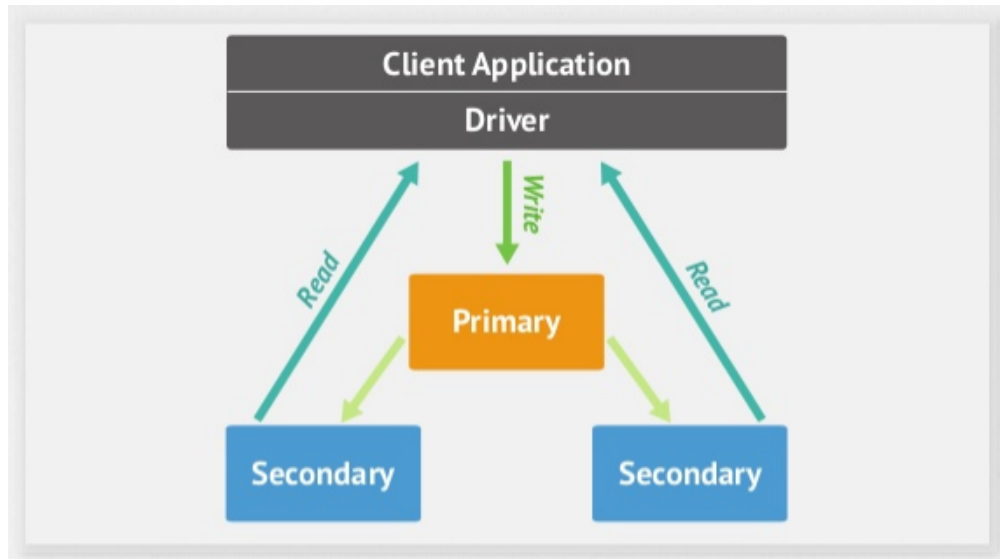


Figure 3.10: Replica-set example in MongoDB

- Replica set system: Defines a system similar to master-slave, but with the added feature to change the master in case it goes down. A schema of the replica-set system can be found in Figure 3.10

About the sharding system, it is based on Big Table, essential to ensure the scalability of data distributing it based on data rank over the sharding keys. This way, read operations are faster. Finally, the file system is GridFS, perfect for storing and loading big files.

3.3.3.3 Cassandra

Cassandra is a column-oriented database management system created by Facebook to improve their messaging systems and was given to Apache foundation once the support was abandoned [44]. It is conceived as a Big Data system to manage large amounts of data prioritizing the write operations, where this system excels with respect to other NoSQL systems, over the lecture operations. It also provides a query language similar to Structured Query Language (SQL) called Casandra Query Language (CQL).

Casandra is a decentralized system where all cluster nodes have the same role, and the data is distributed between those nodes and any of them can be accessed for read operations. Each row, accessed using a key, contains one or more columns, and each column is a key-value pair with a timestamp, used to update the data inside a cluster. The columns within a row are stored

sorted by key, and even if a row contains a large number of columns, the system is capable to manage them very efficiently.

The main problem with Cassandra is that the system restricts the query operations, the data must be added in the order it needs to be queried. Also, it does not offer geospatial indices natively.

3.3.3.4 Redis

Redis is an open-source key-value database which uses the memory to store the values (although it also offers disc persistence), making it extremely efficient [45]. This database has clients for almost any current language, including C, C#, C++, Java, Python, Ruby, Perl, PHP, Scala, Objective-C and Node.js.

The main point of Redis is their Publication/Subscription system that allows users to subscribe to channels, so every time an update happens in the system, this update is send to all the clients subscribed to that channel, acting this way as a real-time system.

The limitations of this system are the type of data allowed for storage (only strings, lists, string sets or hashes), and the fact that is memory-oriented makes it only useful for short datasets, even it has external plugins to add disc persistence.

3.3.3.5 Neo4j

Neo4j is a graph-based database where data is stored as nodes connected between them with oriented relationships containing metadata and properties [46]. It has clients for Java, Ruby, Python, PHP and .Net.

This database is fully scalable, offers persistence, and high availability of the data. Also, another key feature is the query languages Cypher and Gremlin, both graph-oriented to execute efficient queries with millions of node graphs.

It is a really interesting database management system, but only when representing data structured as a graph, like the friendship relations on a social network, and this is out of the project topic.

3.3.3.6 HBase

HBase is a column-oriented database management system aiming to offer scalability using the Big Table project [47]. It works over Hadoop, a framework for distributed calculation over big amounts of data using the Map Reduce paradigm, and using Hadoop Distributed File System (HDFS).

It uses Apache Zookeeper to do a very well executed load balancing, allowing a linear scalability each time a new node is added to the system. But it has the same downsides as Cassandra, being column-oriented limits a lot the type of queries that can be performed.

Chapter 4

Implementation

Once defined the state of art of the current technologies, here we specify the implementation process of this project. As stated before, the system has to be able to extract social data information from different data sources. Despite this, because of time limitations, the system only uses Twitter as the data source, as we mention below in Section 3.1.4.

The implementation is divided in 3 different layers, following a MVC pattern, and this chapter is structured starting from the lowest layer in upward direction.

4.1 Data Access Layer

We have already seen the advantages in scalability and flexibility of non-relational databases over relational database in Section 3.3.3. Therefore, the implementation of the Data Access layer uses Couchbase, a document-based NoSQL database, with an Elasticsearch system over it, a Java application that connect natively with Couchbase for indexing and searching purposes.

The database nodes run in Minerva, a cluster allocated in the Barcelona Supercomputing Center consisting of 24 cores Intel Xeon E5-2620 2.00 GHz, a 64GB of RAM memory, and a Redundant Array of Independent Disks (RAID) storage formed by 1.4TB of Solid-State Drive (SSD) and 3TB of Hard Disk Drive (HDD). The machine is shared with other systems, so the system administrator decided to install a proxy server that routes the connections by port number.

The data access layer architecture used in this project is represented in Figure 4.1. Here, we can see three differentiated layers.

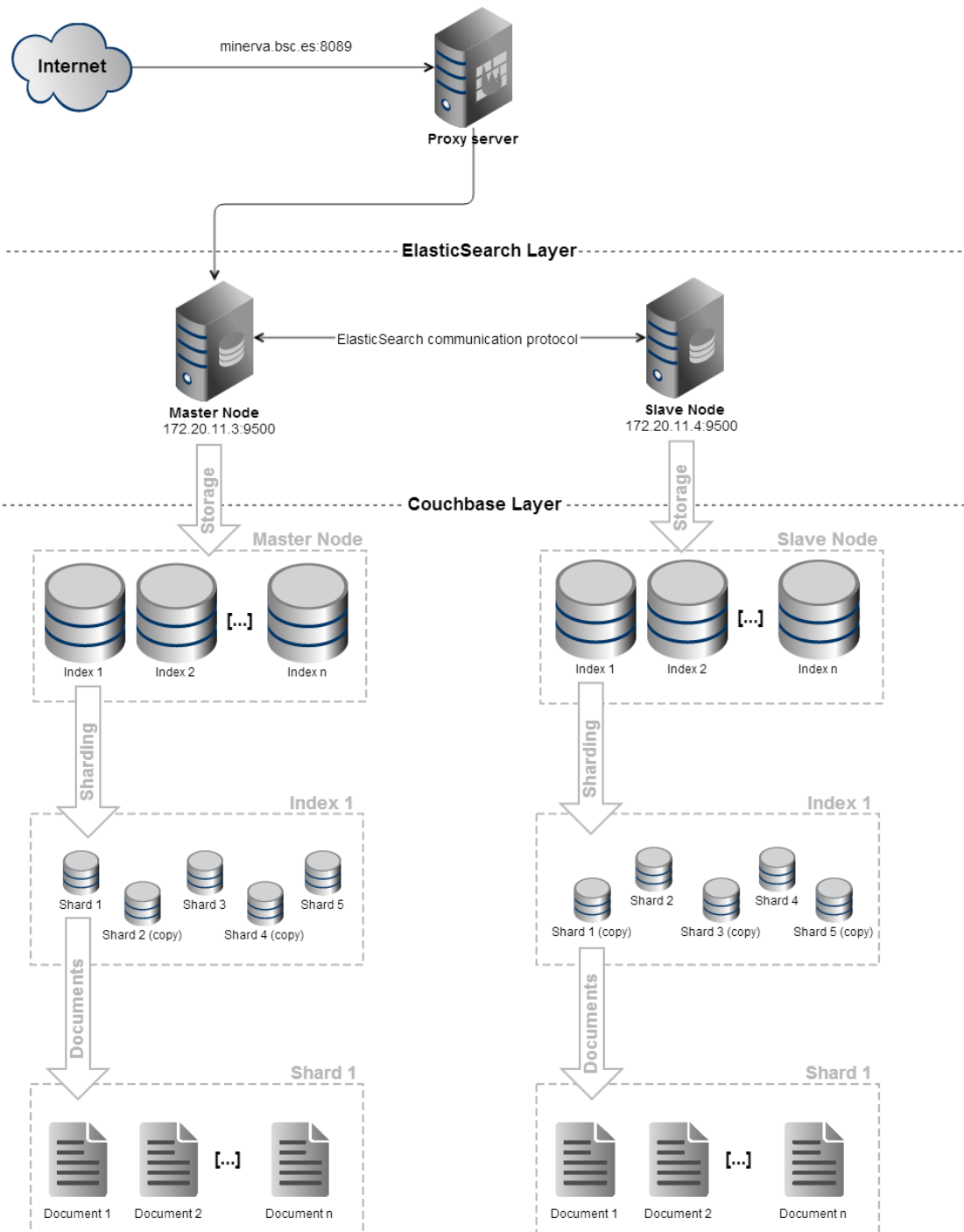


Figure 4.1: Schema of the data layer architecture

4.1.1 Proxy Layer

As we mentioned before, the entrance point from Internet is a proxy server. Elasticsearch follows a REST architecture, which means all database actions are managed through HTTP GET, POST, UPDATE or DELETE connections.

All HTTP connections sent by the business layer to the database public URL (`http://minerva.bsc.es:8089`) are redirected by the proxy server to the Elasticsearch layer, which is running a private Internet Protocol (IP) address (`172.20.11.3:9500`).

ElasticSearch, by default, listens for HTTP connections on port [9200-9300] (the range means that if the port is busy, it automatically tries the next port). Because other instances may be already running in the Elasticsearch nodes, the proxy server is configured to send HTTP traffic to ports 9092 to avoid collisions.

4.1.2 Elasticsearch Layer

The Elasticsearch layer contains one or more nodes within the same private network able to communicate between them using broadcast messages with the Elasticsearch HTTP protocol. In our case, the system consists of two nodes, one of them acting as a master and the other as slave:

- Slave node: Slave nodes act as data nodes, receiving HTTP connections from the proxy server and executing them properly.
- Master node: Is responsible of maintain the global cluster state, and acts if other nodes join or leave the system by reassigning shards. To do that, broadcast messages are sent through the private network to communicate between all nodes using port 9500. In this system, the master node also acts as a data node the same way slave nodes do.

There are four different types of APIs, all of them using HTTP REST architecture:

- Cluster API: Cluster level API allows to specify which nodes to execute on (for example, getting the node stats for a node). Nodes can be identified in the API either using their internal node id, the node name, address, custom attributes, or just the `_local` node receiving the request. The following example retrieves the state of a node:


```
GET 'http://minerva.bsc.es:8089/_cluster/state'
```

- Indices API: The indices API are used to manage individual indices, index settings, aliases, mappings, index templates and warmers. This includes creating, deleting, updating and opening/closing indices, flush the database, clear cache or create index aliases. The following example creates a new index called "tweets" with 5 shards and 1 replica:

```
PUT 'http://minerva.bsc.es:8089/tweets' {
  settings : {
    number_of_shards : 5,
    number_of_replicas : 1
  }
}
```

- Documents API: This section describes the following Create, Read, Update and Delete (CRUD) APIs:

- Index API: The index API adds or updates a typed JSON document in a specific index, making it searchable. The following example inserts the JSON document into the "twitter" index, under a type called "tweet", with id valued 1:

```
PUT 'http://localhost:9200/twitter/tweet/1' {
  user : "Dani",
  post_date : 2014-11-15T14:12:12,
  message : "New document!"
}
```

- Get API: The get API allows to get a typed JSON document from the index based on its id. The following example gets a JSON document from an index called twitter, under a type called tweet, with id valued 1:

```
GET 'http://localhost:9200/twitter/tweet/1'
```

- Delete API: The delete API allows to delete a typed JSON document from a specific index based on its id. The following example deletes the JSON document from an index called twitter, under a type called tweet, with id valued 1:

```
DELETE 'http://localhost:9200/twitter/tweet/1'
```

- Update API: The update API allows to update a document based on the script provided. The operation gets the document from the index, runs the script (with optional script language and parameters), and index back the result (also allows to delete, or ignore the operation). The following example updates the "end time" field of the tweet with id equal 1:

```
UPDATE 'http://localhost:9200/twitter/tweet/1/_update' {
  script: 'if (ctx._source.id == id) { ctx._source.end = endTime } else {
    ctx.op = "none" }',
  params: {
    id: 1,
    endTime: moment().utc().valueOf()
  }
}
```

- Search API: Search API is a multi-index, multi-type API used to retrieve data from the indices. It allows to define queries using a Query DSL based on JSON to create filters. The following example searches all documents in index "tweets" of type "tweet" which contain a field called "geo.coordinates":

```
GET 'http://minerva.bsc.es:8089/tweets/tweet/_search' {
  query: {
    filtered: {
      filter: {
        bool: {
          must: [
            {
              "match_all": {}
            },
            {
              exists: {
                field: geo.coordinates
              }
            }
          ]
        }
      }
    }
  }
}
```

ElasticSearch, by default, binds itself to the 0.0.0.0 address, and listens on port [9300-9400] for node-to-node communication. (again, the range means that if the port is busy, it automatically tries the next port). As we said, to avoid collisions, the node-to-node communication is broadcasted using port 9500.

4.1.3 Couchbase Layer

The last layer is the Couchbase Layer, responsible of storing the documents following the orders of ElasticSearch nodes. Documents are organized using indices, a common ID attached to all documents to identify them. When ElasticSearch creates a new index, each data node creates the index and stores the information and data related to that index, so in case one of the nodes fails, the system automatically recovers the data from the other nodes.

Each index is distributed in shards, which allows ElasticSearch to distribute the data between all the nodes for scalability and reliability purposes. Our system uses 5 shards per index, which means the data stored in each index is divided in 5 shards, and those shards are then stored in each node, some of the shards acting as primary shards (where the data is loaded from), and the copies acting as a backup shards in case primary shards become unavailable.

Each shard contains an unknown number of documents. Which shard the documents are stored at is determined by ElasticSearch. The document is the lowest level item, and consist on a single unit of information (like a row in a SQL table).

4.2 Business Logic Layer

BEcause the system works using an asynchronous model with non-blocking request, offering high concurrency and scalability, the implementation of the Business layer uses Node.js, relying on multiple external nodes.

The back-end nodes forming the business layer runs in two machines: Sawyer, acting as the master node, consist of 2 cores Intel Xeon E5620 2.40GHz, a 2GB memory, and a 40GB HDD storage. Zipi, acting as the crawling node, consist of 1 core Intel Xeon E5620 2.40GHz, a 1GB memory, and a 40GB HDD storage. Both systems are virtual machines, which means the physical machines have superior specs but are shared with other systems.

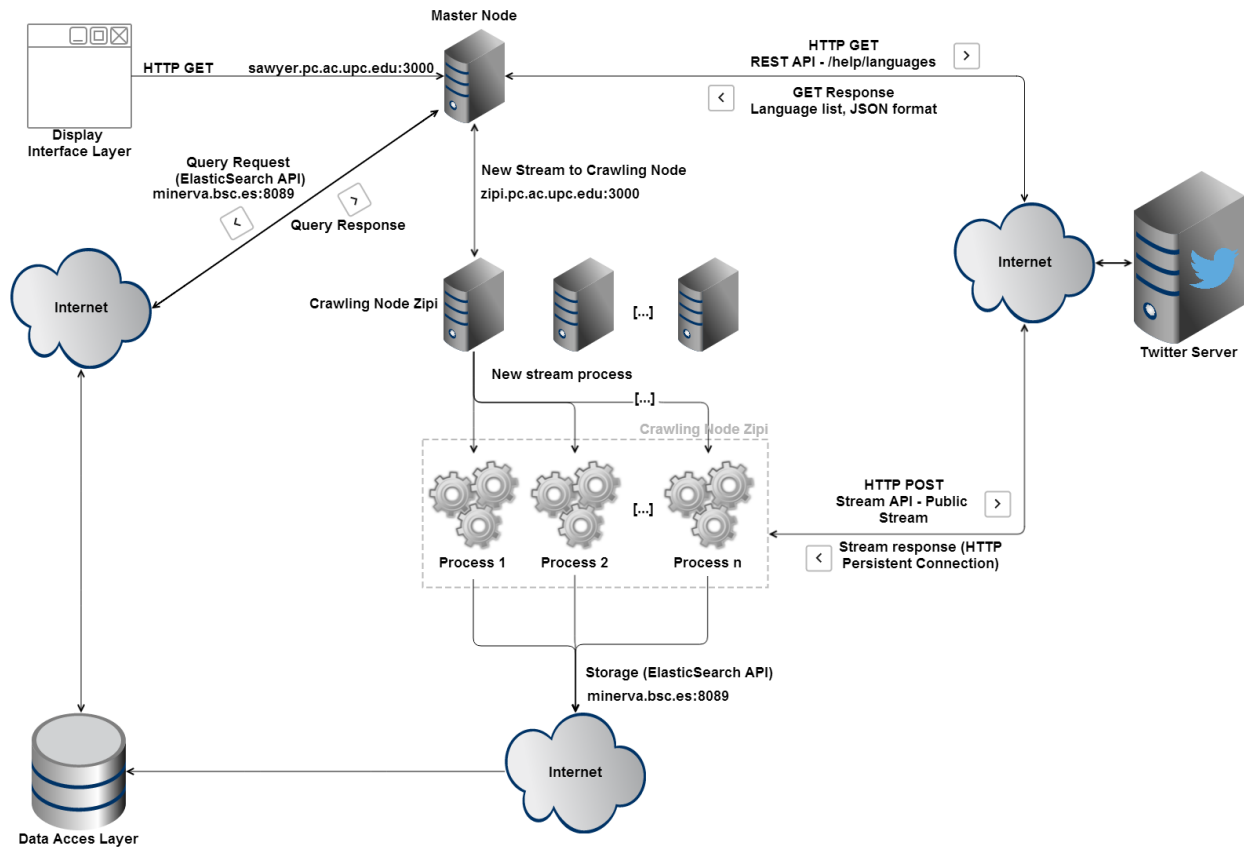


Figure 4.2: Schema of the business logic layer

The business layer architecture used in this project is represented in Figure 4.2. Here, we can see the components described below.

4.2.1 Master Node

The master node acts as the entrance point of the system. It is responsible to render the web interfaces when requested by the browser, and also acts as communication point between the Display Interface and the proper crawling node, which is responsible to initiate the connection with Twitter.

A detailed list of the master node functions include:

- Load Twitter languages: When setting the filters for the query, one of the available filters is the tweet language. To know which languages are available, Twitter offers the API endpoint GET help/languages (<https://api.twitter.com/1/help/languages.json>), which re-

sponds with a list of supported languages. The master node executes this request only once, when the system starts, and stores the result in memory.

- Retrieve query history: In order to visualize the data, users must select in the display interface which queries want to retrieve data from. To do that, when the master node receives the request, it access the "queries" index in the database, which stores information about the queries executed (filters used, index identifier, and start and end timestamps), and sends the response to the interface.
- Receive crawlers updates: To show the appropriate data about the status of crawler nodes in the display interface, the master node has a configuration file with the URL of all crawling machines available. When the system starts, the master node opens a socket with each crawling machine, and then the crawling machines send information about their status every 5 seconds. In case one of the crawling machines appears unreachable, the master node tries to contact again every 5 seconds and send the stream request to other crawling nodes, so the system automatically recovers from crawler node failures. For our project, we only have one crawling node, but the system is ready to have multiple ones by just adding them in the configuration file.
- Route stream requests: The master node processes the start stream requests and redirects them to the first crawling node with available slots, and notifies the display interface if it started correctly or all of them are busy. The master node also redirects the stop stream requests to the right crawling node.

4.2.2 Crawling Node

The crawling nodes are responsible of starting and stopping the streaming processes. One crawling node may have one or more slots, each slot capable to initiate a new stream using a unique set of Twitter API keys, configurable using the crawling configure file.

A detailed list of the crawling node functionalities include:

- Start streaming process: Whenever a new stream request arrives from the master node, the crawling node creates a new streaming process using the credentials of an empty slot.

- Stop streaming process: When stop requests arrive from the master node, they are already identified by the stream ID, so the crawling node checks that ID and destroys the appropriate process, releasing the slot.
- Send status updates: Whenever the master node makes the first contact and opens a socket, crawling nodes use that socket to send updates about the slot status every 5 seconds.

The crawling nodes also host the streaming slots created. They are separated processes running parallel to the main thread, and have the following functionalities:

- Establish the Twitter connection: The stream process is responsible for establishing a new connection using the credentials and the query filters given by the crawling node. The established connection remains open, and Twitter servers use it to send the tweets fitting the filters in real-time.
- Manage disconnection and reconnection: During the streaming connection, it is important to avoid disconnections and, in case they occur, handle the reconnection. This is entirely managed by the streaming processes. Once an established connection drops, the stream process attempts to reconnect immediately. If the reconnect fails, it slows down the following reconnect attempts according to the type of error experienced: Backing off linearly for TCP/IP level network errors, and exponentially for HTTP errors and HTTP 420 errors (usually caused by Twitter restricting the connection due to saturation problems).
- Process data: Once the connection is established, the streaming process fires a set of functions every time a new tweet arrives. These functions include a set of data enrichment modules, and the storage of the enriched tweet.
- Enrich tweets: Once the connection has been established, the stream process sends the tweet to a group of enrichers. Enriching a tweet consists of adding new related information to the raw response coming from the Twitter streaming API.
- Store enriched tweets: Once the tweet has been enriched, the streaming process stores it in the right index sending an HTTP POST connection to the ElasticSearch API.

The enrichers used to aggregate more information to each tweet during the enrichment step are described below.

4.2.2.1 Device enricher

The device enricher parses the "source" field in the response from Twitter to obtain the device that was used to send the tweet. The parsed field is always the HyperText Markup Language (HTML) code of the widget that sent the tweet. The device enricher eliminates the HTML tags and gives it a friendlier name.

4.2.2.2 Geo enricher

The geo enricher provides two functions lacking from the filter capability of the Twitter streaming API related with the tweet geolocation:

- AND filtering: Whenever a query is set with words and geolocation as filter, the Twitter streaming API treats that as an "OR" filter, sending all tweets containing the words or all tweets from that geolocation. This enricher filters tweets that have the words but not the geolocation, and also those which have the geolocation but not the words.
- Geolocation error correction: In order to avoid bandwidth saturation, Twitter sometimes sends tweets which are not inside the location specified by the filters. This filter also ensures that all tweets are inside the right bounding box, and discards those which are outside.

4.2.2.3 Spanish autonomous communities enricher

This enricher is used to classify tweets with geolocation and coming from Spain by autonomous community. The enricher has a detailed bounding box of the 17 Spanish autonomous communities, so whenever a tweet with the "place.country_code" field equal to "ES" arrives, the stream process sends it to the Spain enricher, who see which autonomous community the tweet coordinates fall in using the geolib node module [64].

In case there is a matching, the enricher adds a new field "geo.spain_code" with the code of the autonomous community. Those codes follow the ISO 3166-2 standard [52], which defines codes for the names of the subdivisions of all countries coded in ISO 3166-1.

4.2.2.4 Gender enricher

The gender enricher classifies tweets by the user gender. In order to achieve that, the enricher look at the "user.name" field, where Twitter users set their complete name (not the account name, but their real name). The enricher parses the first name of that field, and then uses a REST API service provided by the Statistical Institute of Catalonia (IDESCAT) for onomastic with the Catalanian census. This service provides a response of the number of people registered with the provided name in Catalonia and the gender. Because it looks more than 7.5 million names, the most typical names are always matched, including foreign names.

Once the gender information is retrieved asynchronously, it is added to the tweet under the "user.gender" field. That field may have 4 different values: undefined, when the API has no entries for the name, unknown, when the API has entries for the name but does not provide gender, and finally male and female, when the API has entries for that name and those entries include gender. The call executed to obtain the gender is:

```
'http://api.idescat.cat/onomastica/v1/noms/dades.json?id=' + name
```

4.2.2.5 Stopwords enricher

The stopwords enricher filters the stopwords from the "text" field of the tweet. Those stop words are filtered to provide sentiment analysis after. In order to filter those words, a set of stop words dictionaries are included in 20 different languages, including Arabic, Catalan, Czech, Danish, Dutch, English, Finnish, french, German, Greek, Hungarian, Italian, Japanese, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish and Turkish.

Whenever a new tweet arrives, the stopwords enricher receives the text and the language code (from the "lang" field). If the language is supported, then the text is filtered checking word by word if it matches one of the dictionary entries. The response is the tweet text with the stop words filtered.

4.2.2.6 Stem enricher

The stem enricher applies a stemming to the remaining text once the stopwords filter has been applied. In linguistic morphology, stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form, generally a written word form.

The stemmer uses an implementation of the Lancaster algorithm using the Node.js module "natural". The Lancaster algorithm (also called Paice/Husk algorithm) was published in 1990, and is a conflation based iterative stemmer which utilises a single table of rules, each of which may specify the removal or replacement of an ending. The rules are indexed by the last letter of the ending to allow efficient searching and are of the following form:

- An ending of one or more characters, held in reverse order
- An optional intact flag '*'
- A digit specifying the removal total (zero or more)
- An optional append string of one or more characters
- A continuation symbol, '>' or ''

These rules are divided into four steps, described in the flowchart in Figure 4.3:

1. Select relevant section: Inspect the final letter of the term and, if present, consider the first rule of the relevant section of the rule table.
2. Check applicability of rule: If final letters of term do not match rule, or intact settings are violated or acceptability conditions are not satisfied go to stage 4.
3. Apply Rule: Remove or reform ending as required and then check termination symbol, and either terminate or return to stage 1.
4. Look for another rule: Move to the next rule in table, if the section letter has changed then terminate, else go to stage 2.

Due to the need of a complex set of rules for each language, only Spanish and English implementation are provided.

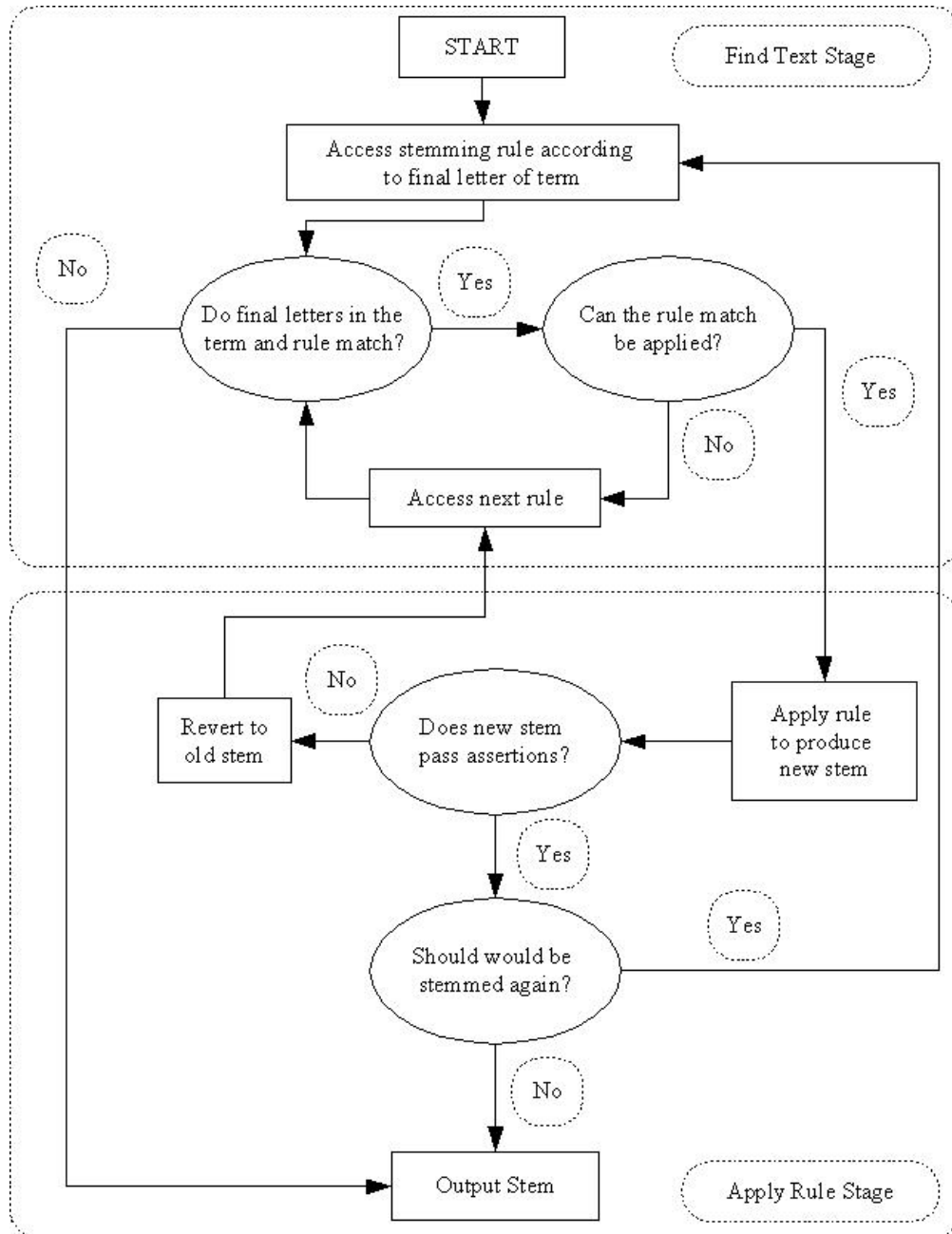


Figure 4.3: Flowchart of Lancaster stemming algorithm

4.2.2.7 Sentiment enricher

The sentiment enricher applies a sentiment analysis over the stemmed tweet. This enricher adds two new fields to the tweet: valence, the pleasantness of the tweet, and arousal, the intensity of emotion provoked by the tweet.

This enricher calculates both values using a sentiment dictionary recollected in McMaster University, Canada, which classified 13,915 English words via 1,085,998 ratings collected by participants in Amazon Mechanical Turk's crowdsourcing website [51]. Those participants were asked to rate 350 English words by valence and arousal from 1 (sad/calm) to 9 (happy/excited). We proceeded to translate it using automatic translators to have those words also in Spanish. Here is an example of an entry from the sentiment dictionary, showing the English word, the Spanish word, the sentiment score and the arousal score:

abrupt , abrupto , 3.28 , 2

The sentiment analysis is divided in three steps:

1. Language select: From the "lang" field of the tweet, we extract the language, and discard those that are not in Spanish or English.
2. Word analysis: The stemmed tweet is checked word by word with the appropriate dictionary (English or Spanish). If there is a match, their valence and arousal values are stored.
3. Calculate sentiment: Once all the words are processed, the mean of valence and arousal is calculated and added to the fields "score" and "arousal" of the tweet. Also, another field "sentiment" is calculated from the valence mean in the following way:
 - Very Positive: Valence mean is higher than 7
 - Positive: Valence mean is between 7 and 5.5
 - Neutral: Valence mean is between 5.4 and 4.5
 - Negative: Valence mean is between 4.4 and 3
 - Very negative: Valence mean is lower than 3

4.3 Display Interface Layer

The display interface layer is responsible of provide users interaction with the system. It consist on four different web pages rendered by the master node. Whenever a new page is requested via a HTTP GET request, the master node renders it and opens a permanent web socket, which is used to listen for further user interactions with the system and send updates to the view. For a schema of the display interface layer, see Figure 4.4.

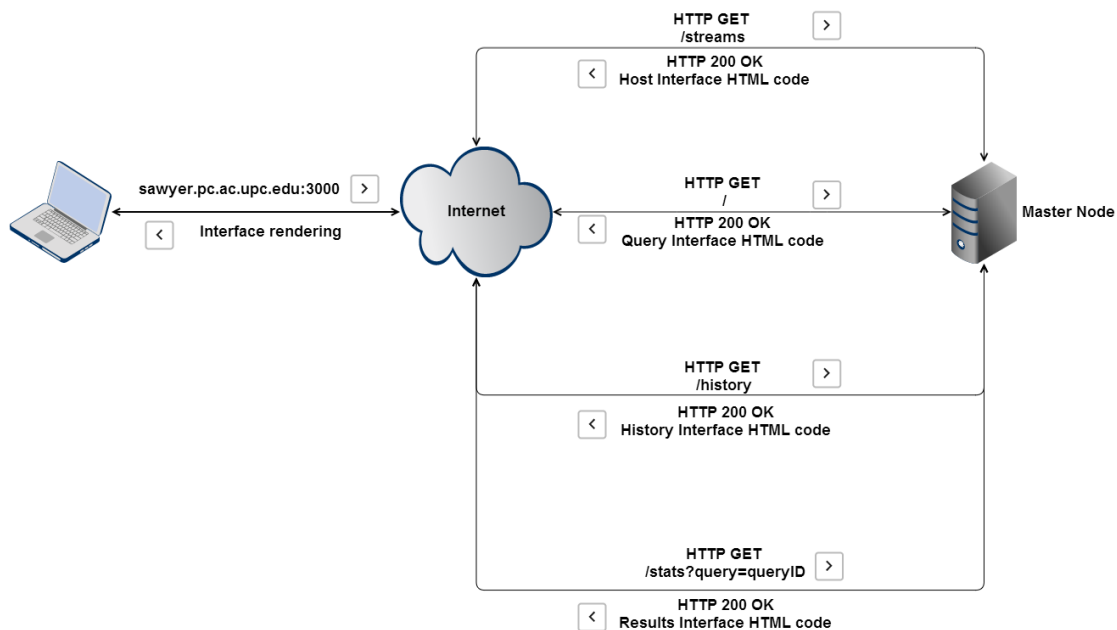


Figure 4.4: Schema of the display interface layer

4.3.1 Query interface

The query interface is the view that allows users to start a new stream. It provides controls to set the language, time, location and tracking words for the new stream request. For a schema of the query interface, see Figure 4.5.

The query interface is formed by four different sub views:

- Main view: The first view displayed to the user. It includes the text bar to write down the words of the filter, 3 different links to show the other 3 views, and the search button to

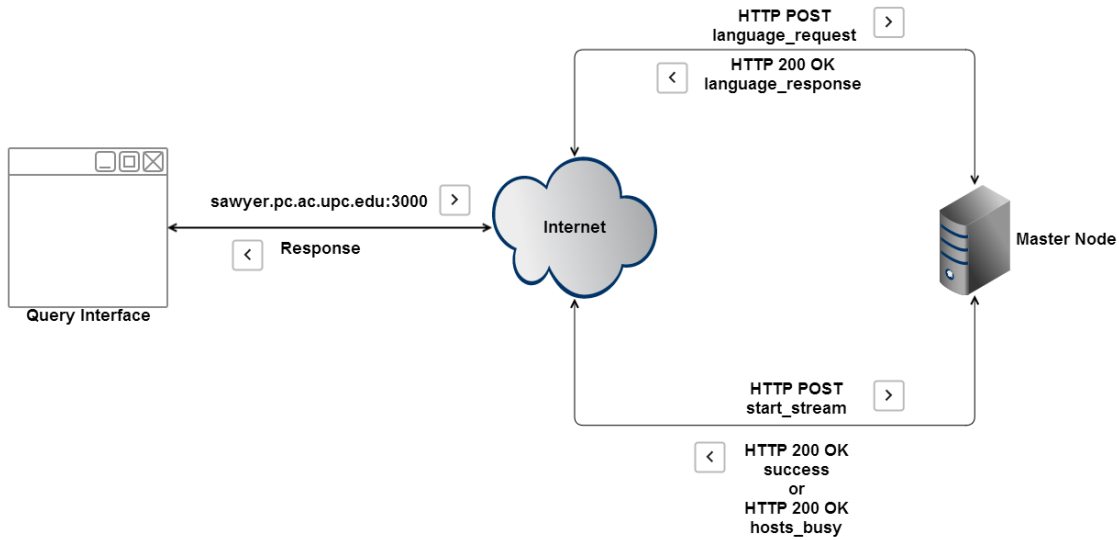


Figure 4.5: Schema of the query interface

start the stream. The half bottom of the page, initially empty, shows all the current filters as they are selected. See Figure B.1 in Annex B.

- Language view: The language view shows a multi-select drop list of all twitter supported languages. Users may select one or more languages to add to the filters. See Figure B.2 in annex B.
- Time view: The time view allows users to set a date when the stream stops automatically. By default, the check-box is disabled, but when enabled, a calendar and time picker are shown, and users may set the date and time. See Figure B.3 in annex B.
- Location view: The location view shows a map view where users can set the locations of the filter. The view renders a map from the Google Maps API [29], and this map provides a bounding box drawing tool. Each location is generated with a random colour, and can be erased from the right side of the view or from the main view. See Figure B.4 in annex B.

4.3.2 Host interface

The host interface is the view that allows users to see the status of the crawling nodes. Users can check how many available slots crawling nodes have, and what streamings are currently running in those which are busy.

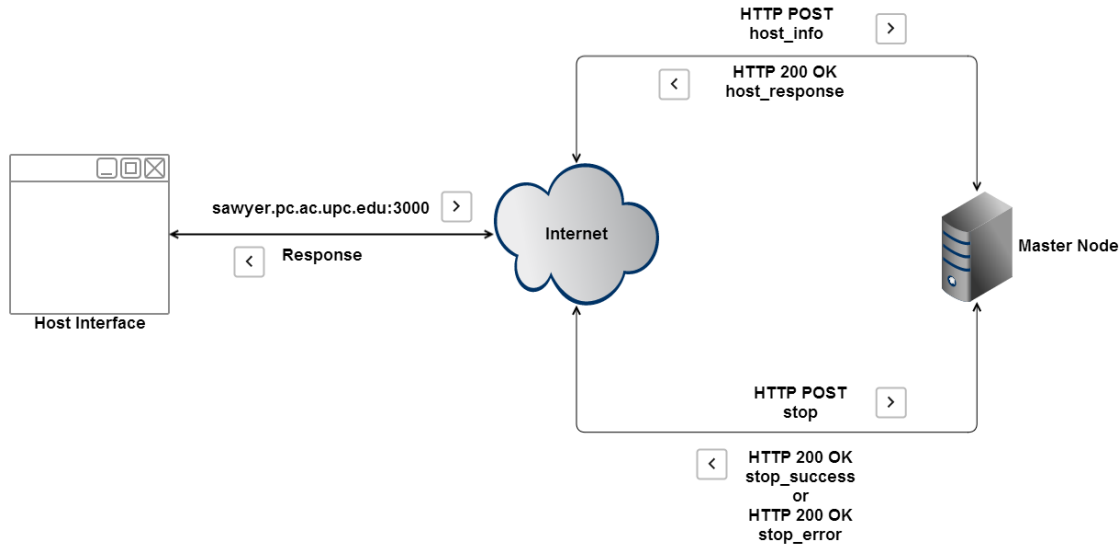


Figure 4.6: Schema of the host interface

This view, when rendered, request the current status of crawling nodes to the master view. Once received, each crawling node and their slots are rendered using an accordion view. Each slot may have a green check if it is free, or a red stop sign plus an information sign if it is busy. For busy slots, the red stop sign stops the stream in that slot, and the information sign shows another sub-view with information about the query.

For a schema of the host interface, see Figure 4.6. For host interface captures, see Figures B.5, B.6 and B.7 in annex B.

4.3.3 History interface

The history interface shows a table with a record of all the queries executed. Users can check the information about those queries, retrieve the data using the results view, and delete them.

This view, when rendered, request the query record to the ElasticSearch index "queries", where a record of each index and their filters, start time and end time are stored. Once the data is received, the table is filled.

For a schema of the history interface, see Figure 4.7. For history interface captures, see Figures B.8 and B.9 in annex B.

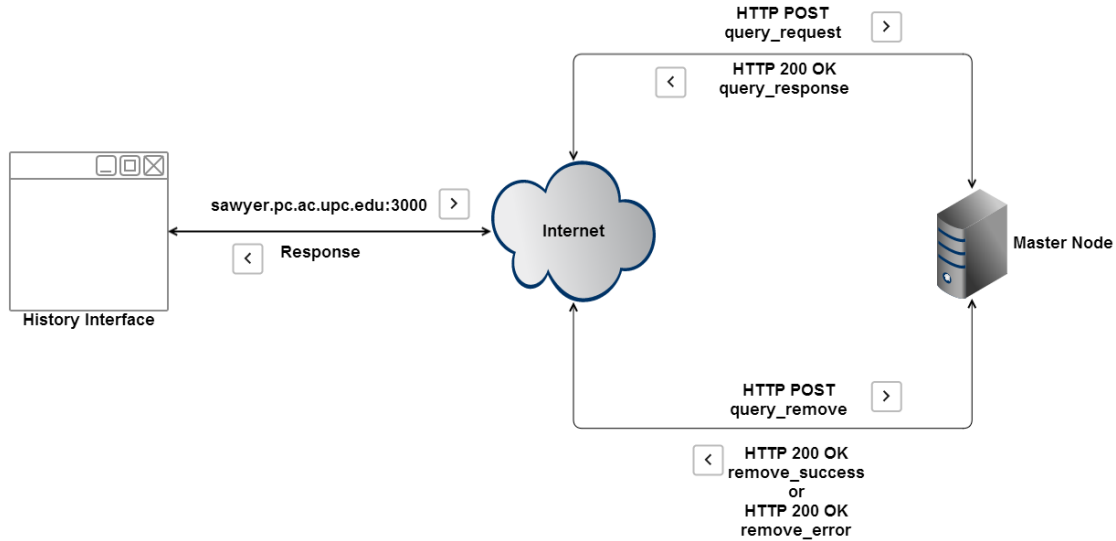


Figure 4.7: Schema of the history interface

4.3.4 Results interface

The result interface shows the data related to the queries selected in the history interface, using editable, movable and erasable panels provided by Kibana, containing char pies, char bars, histograms, tables and maps that are refreshed every minute by default.

We select the queries to display using GET parameters and preparing the Kibana panels through a JavaScript loader that sets the right IDs to each panel. The ones displayed by default are:

- Device: A pie chart with the top 10 devices used to send the tweet.
- Language: A bar chart with the top 10 tweet languages.
- Tweet and retweet count: A histogram with the number of tweets and retweets per time unit. The time unit defaults to 1 minute, but may be changed.
- Most common words: A bar chart with the top 10 most common words in tweets.
- Most common hashtags: A bar chart with the top 10 most common hashtags in tweets.
- Most mentioned users: A bar chart with the top 10 most mentioned users in tweets.
- Gender: A pie chart with the gender of the users that wrote the tweet.

- Sentiment: A pie chart with the different sentiments that a tweet may contain, as seen in section 4.2.2.7.
- Sentiment over time: A histogram with the valence score of the tweets over time, as seen in section 4.2.2.7. The time unit defaults to 1 minute, but may be changed.
- Arousal over time: A histogram with the arousal score of the tweets over time, as seen in section 4.2.2.7. The time unit defaults to 1 minute, but may be changed.
- Geo-position: A map geo-positioning each tweet, which a pop-up when hovering over the point showing the tweet text. Kibana uses the library Leaflet [48] to create an interactive, zoomable, movable map. Kibana uses CloudMade API, a private service to provide the Map tiles, but we customized the panel to use Open Street Maps API [50], which is much more detailed and open source.
- World tweet count: A shape-form map with a count of the number of tweets per country. Kibana uses the library jVectorMap [49] to create maps from Scalable Vector Graphics (SVG).
- Spain tweet count: The only custom-made panel. Kibana provided 3 shape-maps: whole world, Europe and USA. In order to classify tweets by Spanish autonomous community, we added a Spanish map to the map panels, adding the Spanish shape from jVectorMap website.

For a schema of the results interface, see Figure 4.8. For history interface captures, see Figures B.10 and B.11 in annex B.

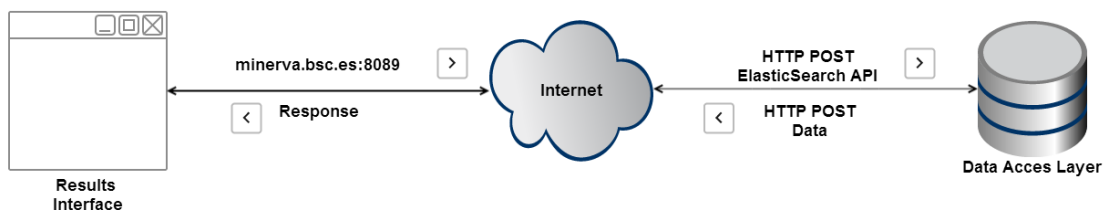


Figure 4.8: Schema of the results interface

Chapter 5

Testing

In this chapter we present the results obtained from the functional evaluation of the system. The test methodology followed, as part of the extreme programming methodology [58], is to test each part separately once finished, and then perform a final generic test on the whole system.

Firstly, we present a series of tests to ensure the database is correctly configured. Secondly, we test the whole business logic layer, using unitary tests on the different classes. Finally, we ensure the display interface correctness using different use cases.

5.1 Data Access Layer tests

To test the data layer, we applied using the Elasticsearch API, to ensure that read, update and delete operations (CRUD) work as expected. Those tests are:

- Having an empty database, after creating a new index using the Indices API [53], that index has been created correctly.
- Once the index is created, new data can be added using the Index API [54].
- Once data has been added, it can be retrieved using the Search API [55].
- When an element is stored inside an index, the element properties can be modified using the Update API [56] using the same element ID.

- When an element is stored inside an index, the element can be removed from the index using the Delete API [57]

As we can see, we are just making sure that simple operations over the database work as expected, in order to make sure the data layer has no conflicts or future problems when the system pipes the data. Those tests have been executed in a local database located temporarily in the Master Node, in order to not alter the data locate in the production database.

Apart from testing the integrity of the database, we also ensured that the two classes that manage the data retrieval and storage worked correctly applying the following unitary tests:

- Save: To test the save object, we tried to add a new stream by calling `newQuery()`, then add a new tweet by calling `newtweet()`, then simulate we end the query calling `endQuery()`, and finally delete the index by calling `deleteQueries()`.
- Load: To test the load object, we added a new query into the "queries" index using the Indices Elasticsearch API [53], and then called `loadQueries()` to see if the created element is loaded.

A deep explanation of the classes from the data access layer can be found in annex A.1

5.2 Business Logic Layer tests

The business logic layer has been tested using unitary tests on each one of the classes that compose the system. A deep explanation of the classes from the business logic layer can be found in annex A.2.

5.2.1 Master Node

In order to test the Master Node, the following unitary tests have been applied:

- Twitter Loader: `loadLanguages()` was called, and the response was compared to the official list of languages provided by Twitter.

5.2.2 Crawling Node

- Container Controller: To test the container controller, we called `create()` simulating the filters from the input interface, then tried `get()` and `getAll()` methods to see if they work, and finally destroyed the container with `destroy()`.
- Container: To test the container object, we instantiated the object in the terminal with fake attributes, and then tried all its getters and setters.
- Stream Controller: To test the stream controller, we instantiated a new controller, then called `set()` simulating the filters from the input interface, and then `get()` to see if the filters are correctly retrieved.
- Stream: To test the stream object, we instantiated the object in the terminal with fake attributes, and then tried all its getters and setters.
- Device enricher: The device enricher was tested with a real tweet obtained from the stream API. We checked the field "device" was added correctly.
- Geo enricher: The geo enricher was tested with a real tweet obtained from the stream API. We checked that adding tweets with coordinates outside the defined bounding box, and tweets inside the defined bounding box but without the defined words in the text, and ensured both tweets were filtered.
- Spain enricher: The Spain enricher was tested with a real tweet obtained from the stream API. We checked the field "geo.spain" was added correctly.
- Gender enricher: The gender enricher was tested with a real tweet obtained from the stream API. We checked the field "user.gender" was added correctly.
- Stopwords enricher: The device enricher was tested with a real tweet obtained from the stream API. We compared the original text with the new text, added in the field "words".
- Stem enricher: The stem enricher was tested with a real tweet obtained from the stream API. We compared the original text with the new text, added in the field "words".

- Sentiment enricher: The sentiment enricher was tested with a tweet already filtered with the stopwords and the stem enricher. We checked the fields "sentiment" and "arousal" were added correctly.

5.3 Display interface layer tests

To test the display interface, we have executed a deep list of all the possible use cases that our interface allows.

5.3.1 Query interface tests

The query interface tests ensure that users can create any type of query, and cannot create queries that goes against Twitter rules. Those tests include:

- Create a new query with the tracking word "Monday". The created stream recollects tweets containing the word "Monday".
- Create a new query with two words, coma separated (OR), "Monday, Tuesday". The created stream recollects tweets containing either Monday or Tuesday, or both.
- Create a new query with two words, space separated (AND), "Monday Tuesday". The created stream stores only tweets containing both words.
- Create a new query with 3 words, first two space separated, the other two coma separated, "Monday Tuesday, Wednesday". The created stream recollects tweets containing either "Monday" and "Tuesday", or "Wednesday", or all of them.
- Create a new query with one geolocation. The created stream recollects tweets with geolocation whose geolocation is inside the specified bounding box.
- Create a new query with two geolocations. The created stream recollects tweets with geolocation whose geolocation is inside one of the specified bounding boxes.

- Create a new query with word "Monday" and, as geolocation, a bounding box covering USA. The created stream recollects tweets with geolocation, inside the specified USA bounding box, and containing the word "Monday".
- Create a new query with word "Monday", a bounding box covering USA, and English language. The results are the same as before, except for those tweets that are not in English.
- Create a new query with word "America", a bounding box covering USA, languages English and Spanish, and end time 3 hours after the query starts. The created stream recollects tweets with word "America", written in English or Spanish, and geo-located inside USA, and it ends automatically at the expected time, 3 hours after it starts.
- Try to create a new query without word or geolocation. The stream is not created, and an error message is shown.

5.3.2 Host interface tests

The host interface tests ensure that hosts and streams are listed correctly, stream query information can be retrieved, and also that streams can be stopped. Those tests include:

- Get host information when no hosts are on. The host list is shown empty.
- Get host information when one host is on with all slots empty. The host list shows the available host and its available slots.
- Get host information when one host is on with 2 empty slots and 1 busy slot. The host list shows 2 available slots and one busy slot.
- On a busy slot, retrieve information about the slot. A new dialog shows the query information for that slot.
- On a busy slot, stop the stream. A confirmation dialog is shown.
 - When stop confirmation dialog is shown, cancel it. The dialog is closed, and nothing happens.

- When stop confirmation dialog is shown, accept it. The dialog is closed, the list is refreshed to show the new slot status, and the stream stops.
- On a busy slot, when stop confirmation dialog is shown, accept. The dialog is closed, the list is refreshed to show the new slot status, and the stream stops.

5.3.3 History interface tests

The history interface tests ensure that the queries are listed correctly, and they can be retrieved (using the results interface) and deleted. Those tests include:

- Get queries when the database is empty. The query table is shown empty.
- Get queries when the database is not available. The query table is shown empty.
- Get queries when the database contains at one query. The query is shown on the table.
- Get queries when the database contains eleven queries. First 10 queries, ordered by time, are shown, and the eleventh is paginated into the second page.
- Select a finished query and delete it. A confirmation dialog is shown.
 - When delete confirmation dialog is shown, cancel it. The dialog is closed, and nothing happens.
 - When delete confirmation dialog is shown, accept it. The dialog is closed, the deleted row is erased, and the query is deleted at the database.
- Select a query still running and delete it. A confirmation dialog is shown.
 - When delete confirmation dialog is shown, cancel it. The dialog is closed, and nothing happens.
 - When delete confirmation dialog is shown, accept it. The dialog is closed, the deleted row is erased, the stream is stopped and the query is deleted at the database.
- Select a finished or unfinished query, and retrieve its data. The result interface is opened, including as a GET parameter the ID of the selected query.

- Select two finished or unfinished queries, and retrieve their data. The result interface is opened, including as a GET parameter both IDs from the selected queries.

5.3.4 Results interface tests

Because we use an already existent solution to the results interface (Kibana), we do not need to fully test the interface, if there are new bugs, we expect Elasticsearch team to solve them. But we have to test the interface loader, responsible of filling the panels with the appropriate IDs in order to show the data from the selected queries. Those tests include:

- Interface open with no "query" GET parameter. The panels from the interface are shown empty.
- Interface open with one "query" GET parameter. The panels from the interface show data from that query.
- Interface open with two "query" GET parameters. The panels from the interface show data from both queries, aggregating the results in all of them except for the histograms, which show 2 different colours for both queries.

Chapter 6

Results

In order to test the system as a whole, we have proposed two different scenarios: The first one, the Football UEFA Champions League final between Real Madrid and Atlético de Madrid, and the second one, the singer Miley Cyrus' concert played in Barcelona.

6.1 Use case 1: UEFA Champions League Final

6.1.1 Scenario

As the first use case, we used the Champions League football final between Real Madrid and Atlético de Madrid (played 24th May 2014, at 20:45, in Lisbon) to try to analyse the reactions before, during and after the match.

The input filters were set to take all tweets from Spain, written in Spanish, and whose text contained at least one of the following set of words:

- champions
- liga campeones
- atletico madrid
- real madrid

The data gathering was initiated 24th May 2014, at 09:00, and ended 25th May 2014, at 23:59.

6.1.2 Global results

The global results are:

- The total number of tweets sent matching our criteria was 14451. From those 14451 tweets, 4281 were sent from the 12 hours prior to the match (1018 one hour before the match), 2552 were sent during the match, and 7618 were sent after the match (3406 during the first hour). See Figure 6.1 for more details.

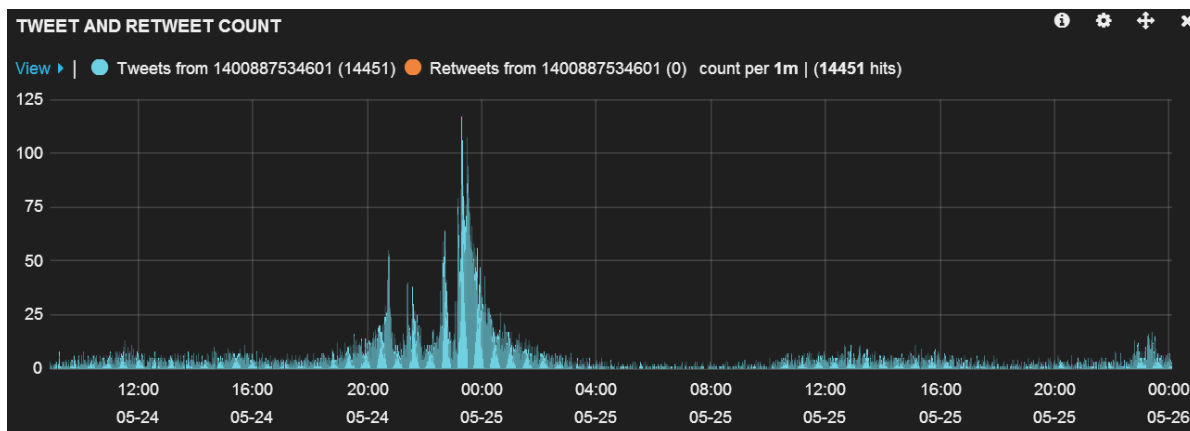


Figure 6.1: Tweet count, Use Case 1

- Handheld devices were the main source. 10593 used an Android device (73%), and 3035 used an iPhone (21%). Only 16 tweets came from a device other than a smartphone or a Tablet (less than 1%). See Figure 6.2 for more details.
- The top 10 words, from high to low, can be seen in Table 6.1.

Comparing the total occurrences between both teams, Real Madrid was mentioned 7883 times (With "real" and "décima", referring to the tenth trophy they could win), while Atlético de Madrid was mentioned 2456 times (the sum of words "atletico", "atleti" and "atlético").

- The top 10 hashtags, from high to low, can be seen in Table 6.2.

Comparing the total occurrences between both teams, Real Madrid was mentioned 1537 times in 6 different hashtags, while Atlético de Madrid was mentioned only in 73 times in 1 hashtag.

Word	Occurrences	Team alignment
madrid	9176	Neutral
real	7334	Real Madrid
champions	6159	Neutral
final	2182	Neutral
atletico	1173	Atlético de Madrid
hoy	983	Neutral
atleti	724	Atlético de Madrid
atlético	559	Atlético de Madrid
décima	549	Real Madrid
ganar	264	Neutral

Table 6.1: Top 10 most common words, use case 1

Hashtag	Occurrences	Team alignment
halamadrid	747	Real Madrid
finalchampions	646	Neutral
champions	404	Neutral
halamadridaporladecima	239	Real Madrid
ladecima	184	Real Madrid
realmadrid	151	Real Madrid
aporladecima	131	Real Madrid
ladecimaesnuestra	85	Real Madrid
aupaatleti	73	Atlético de Madrid
madrid	69	Neutral

Table 6.2: Top 10 most common hashtags, use case 1

- The sentiment analysis shows that 80% of tweets were positive, 17% neutral, and 3% negative. The sentiment over time is stable around 6 out of 9 before the match, oscillates from 5.8 to 7.2 during the match, and stabilizes again around 6.5 when the match was finished. See Figures 6.3 and 6.4 for more details.
- The arousal analysis shows that the intensity of tweets is constant during the whole streaming, around 4.2 out of 9. See Figure 6.5 for more details.
- The geolocation shows that the most active autonomous communities were Madrid (3710), Andalucía (3419) and Comunidad Valenciana (1186), and the least active autonomous communities were Baleares (74), La Rioja (84) and Navarra (105). As for the cities, the most activity comes from Madrid, Valencia, Barcelona and Seville. See Figure 6.6 for the

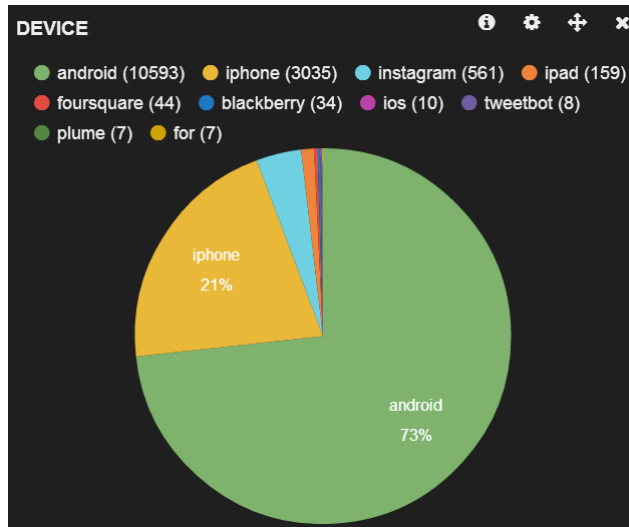


Figure 6.2: Devices used, Use Case 1

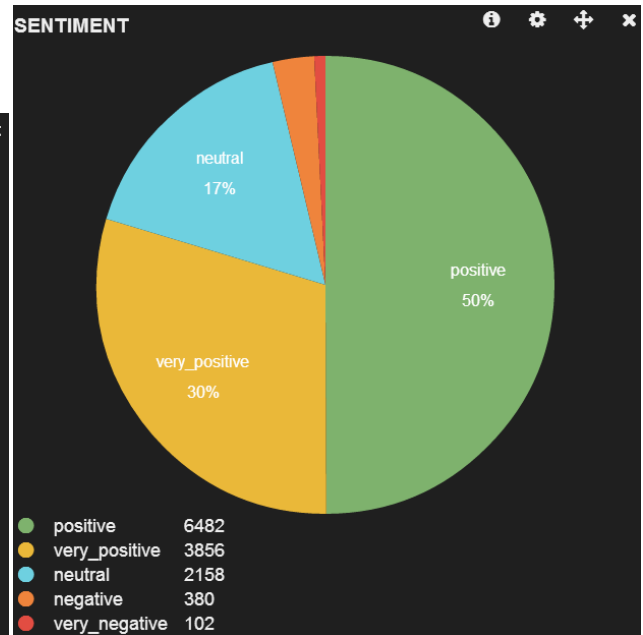


Figure 6.3: Sentiment analysis, Use Case 1

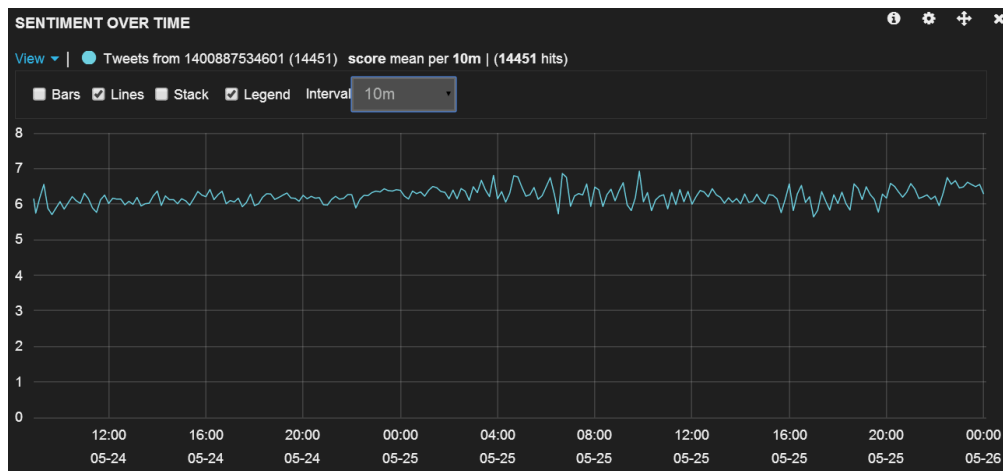


Figure 6.4: Sentiment over time, Use Case 1

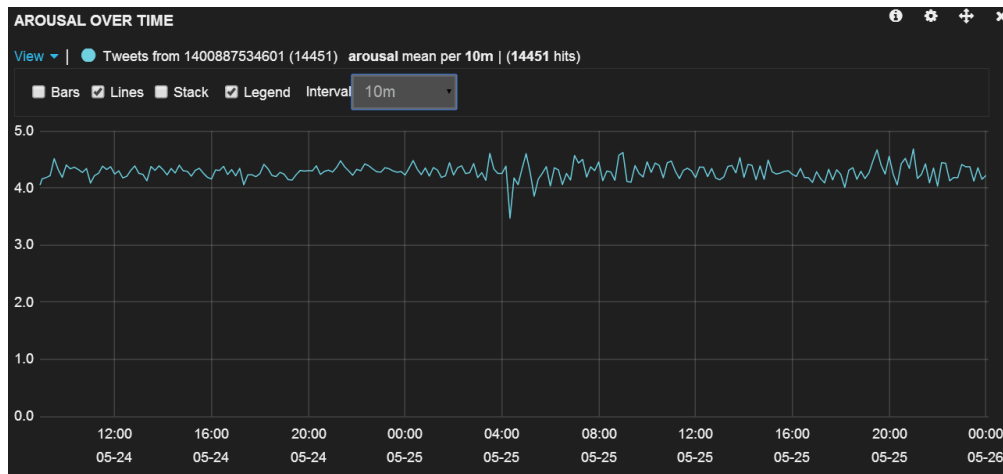


Figure 6.5: Arousal over time, Use Case 1

complete map.

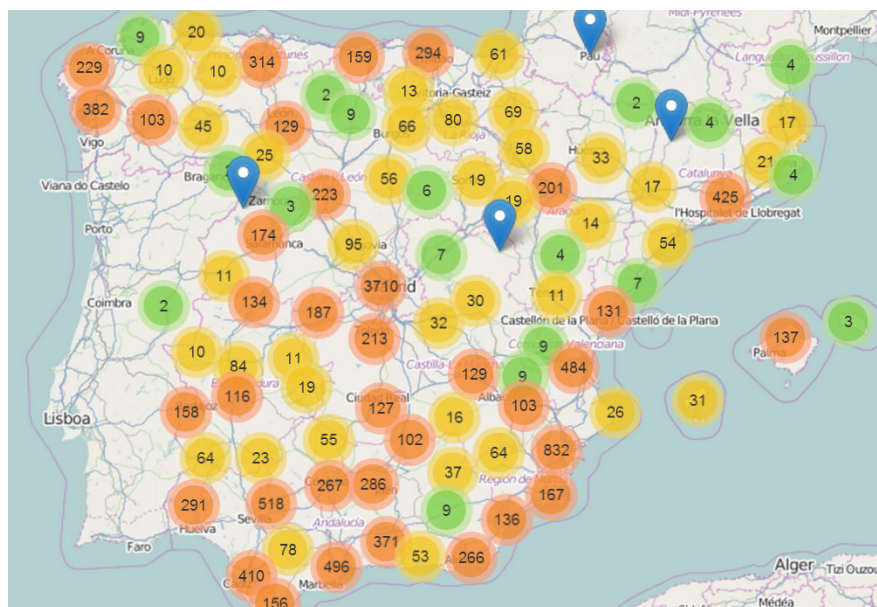


Figure 6.6: Tweet map, Use Case 1

6.1.3 Results by time

In this subsection, we are analyse how the statistics vary over the time.

- During the previous hours, the number of tweets over time is constant, with an increase around 19:15, as the starting time approaches. Most common words include "final" (final), "hoy" (today), "ver" (to see) and "noche" (night), which reveals tweets show the excitement of seeing today's match.
- During the hour prior the beginning, the number of tweets increases linearly, with a peak from 20:42 to 20:46, which shows people were especially active when the match started. Most common words include "ver" (to see), "empieza" (it starts) and "partido" (match), which shows people were tweeting about the game start.
- During the first half, the number of tweets over time decreases, as people was more focused on watching the game. There were two important things happening during the first part:
 - Diego Costa injury at minute 9', star striker from Atlético de Madrid, which is reflected with a slight increase of the tweet count with most common words "costa", "ni" (nor), "jugando" (playing). Sentiment over time goes below 6 during the minutes 9' and 10' (20:55 and 20:56), reflecting the concern of Atlético fans.
 - Atlético de Madrid goal at minute 35', from Diego Godín. A big increase of the tweet count appears going from 10-15 tweets per minute to 35-40, and the most common words include "godin", "gol" (goal), "1" and "0", which shows tweets were talking the goal. The sentiment goes from 5.6 the minute after the goal to 6.4 the next minute, and keeps constant during 5 minutes, showing the initial annoyance of Real Madrid fans and the joy of Atlético de Madrid supporters.
- During the half time, people keeps talking about the scored goal. The number of tweets decreases linearly as the second half approaches, from 35-40 per minute to 5 per minute, and the most common words include "descanso" (half time), "casillas", "gol" (goal), "1", "0" and "gana" (it wins). The sentiment keeps constant around 6.2, with an increase to

6.5 when the second half starts, and the "very positive" tweets go beyond 30% for the first time.

- During the second half, the first 15 minutes were quite active, with 20-25 tweets per minute, and people keeps talking about the Atlético de Madrid goal, with similar most common words. After the first 15 minutes, the tweet count decreases to 10-15 per minute, and Real Madrid supporters seems to be worried about Madrid losing the final, with most common words like "pierda" (it loses), "perder" (to lose), "jugando" (playing) or "mierda" (shit).
- At minute 93' (22:33), Sergio Ramos draws the match for Real Madrid, and the tweet count increases to 35-40 per minute during the last 3 minutes of the second half.
- During the over-time, the tweet count starts at 50-60 per minute and decreases linearly to 10-15 per minute, all tweets referring to Ramos goal, with words like "ramos", "grande" (huge), "sergio" and "ganar" (to win), and a sentiment over 6.5. People talks also about the over-time, with words like "prórroga" (over-time), "parte" (half) or "arbitro" (referee). There were three goals during the over-time, all of them very close to each other:
 - At minute 110' (23:08), Gareth Bale scores the second goal for Real Madrid. The joy of Real Madrid supporters is reflected in an increase from 30 tweets per minute to 80, with a sentiment over 6.3, a record of "very positive" tweets of 39%, and words like "bale", "décima" (tenth, referred to the tenth championship for Real Madrid), "millones" (millions, referred to Bale, permanently questioned by the cost he had), "rey" (king) or "mejor" (best). To a lesser extent, it also shows the frustration of Atlético de Madrid fans, with words like "puta" (bitch) or "merece" (it deserves)
 - At minute 118' (23:16), Marcelo scores the third goal for Real Madrid. Here, Real Madrid supporters celebrate it with an increase to over 120 tweets per minute, with words like "1", "3", "ganar" (to win), "europa" (Europe), or "nuestra" (ours). Atlético de Madrid supporters seem resigned to lose, with words like "enhorabuena" (congratulations), "gracias" (thank you), "cholo" (Atlético de Madrid trainer) or "equipo" (team). The "very positive" tweets arrive to 40% between Real Madrid supporters. It

is remarkable to see that Marcelo is not mentioned, the victory of Real Madrid seems to be more important at this point.

- At minute 123' (23:21), Cristian Ronaldo scores the fourth goal after a penalty. The tweets sent after this goal do not seem to differ from the previous ones: Real Madrid keeps celebrating the victory, with words like "ganado" (won), "campeones" (winners) or "mejor" (best), but no mentions about the goal or about Cristiano Ronaldo. Atlético de Madrid supporters were thanking the team for the good match and the amazing season, with words like "temporadón" (amazing season), "cholo" or "partidazo" (amazing match). The sentiment keeps over 6.5 for both sides.
- During the hour after the game, tweets start decreasing linearly from 80-90 per minute to 15-20. Real Madrid supporters celebrate the victory with a 95% of positive tweets (50% "very positive") and less than 1% of negative tweets, with words like "campeones" (winners), "décima" (tenth), "gracias" (thank you), "champions" or "europa" (Europe). On the other side, Atlético de Madrid supporters congratulate Real Madrid with words like "felicidades" (congratulations) and thank the team with words like "temporada" (season) or "gran" (nice), with an 88% of positive tweets, proving a nice fair play.
- After the match, tweets decrease to 5-10 per minute with a sentiment mean of 6.2. Tweets seem to contain almost the same words as before for both sides, with some variations like "ayer" (yesterday) that indicate people was commenting about the game.

6.1.4 Conclusions

From the previous results, we extract the following conclusions:

- Real Madrid supporters were three times more active than Atlético de Madrid supporters, there were three tweets containing "real" for each tweet containing "atlético" or "atleti". This indicates that either Real Madrid has more supporters than Atlético, or Real Madrid supporters use Twitter more than Atlético supporters.
- There were four notorious peaks in the tweet count: The beginning of the match, the first Atlético goal, the first Real Madrid goal, and huge peak from the second goal of Real

Madrid to 20 minutes after the end of the game. This indicates that people is less prone to tweet during the game unless a goal is scored. The start and the end of the match were very active zones but, curiously, not the half time. People started talking about the match 2-3 hours before starting, and stopped talking about it 2-3 hours after the end.

- Real Madrid supporters gave much importance to the first and second goals, but not to the third and fourth. Sergio Ramos (first) and Bale (second) where seen as "heroes", with a lot of tweets mentioning them, but not anymore with Marcelo (third) and Cristiano Ronaldo (fourth).
- Atlético de Madrid supporters gave a lot of credit to Godín for the scored goal, being one of the top words during the 15 minutes after, but less than the credit Real Madrid supporters gave to Sergio Ramos for his goal, with more mentions during the entire over-time, around 30 minutes.
- Atlético de Madrid supporters, even they almost won, were not angry or sad. They appreciated a lot the effort the players put on the field, and were very thankful with the team and, specially, with the trainer for the good season. This can be seen once the third goal was scored, with most common words from Atlético supporters being appreciation words.
- Atlético de Madrid supporters showed fair play, congratulating Real Madrid for the win. This can be seen again, once the third goal was scored, with most common words being "congratulations" or "good match", with no insults or bad words.
- When tweeting about this match, users barely used computers. This is probably due to people tweeting at bars, at the stadium, or during the celebration, for which handheld devices were used. Android devices were three times more used than iPhone devices, as Android seems to dominate the Spanish smartphone market.

6.2 Use case 2: Miley Cyrus concert in Barcelona

6.2.1 Scenario

As the second use case, we used the concert that the singer Miley Cyrus, very active in Twitter with over 18.2 million followers, played in Barcelona 15th June 2014, from 21:30 to 23:30. We analysed the repercussion this concert had in Miley Cyrus fans over the world, in Miley Cyrus fans talking about the gig, and in Spanish fans.

The input filters were set to take all tweets from all over the world, written in all languages, and whose text contained at least one of the following set of words:

- @MileyCyrus
- Miley Cyrus

The data gathering was initiated 15th June 2014, at 12:00, and was ended 16th June 2014, at 20:00.

6.2.2 Global results

The global results are:

- Tweet count: The total number of messages sent matching our criteria was 186712 tweets and 97937 retweets. The most active time zones were the hour after the gig (from 23:20 to 00:30) with 8123 tweets and 5127 retweets per hour, the day after the gig from 14:00 to 16:00 with 7746 tweets and 4798 retweets per hour, and from 17:30 to 19:00 with 8606 tweets and 4326 retweets per hour. See Figure 6.7 for more details.
- Device: Handheld devices were the main source. 44680 used an Android device (26%), and 43377 used an iPhone (25%). From the 49% remaining, 74708 came from computers (40%). See Figure 6.8 for more details.
- Language: The main language worldwide was English, with 96401 tweets (52%), and the second one Spanish, with 35751 (19%). The third place, Portuguese, is mostly due to Brazilian fans. See Figure 6.9 for more details.

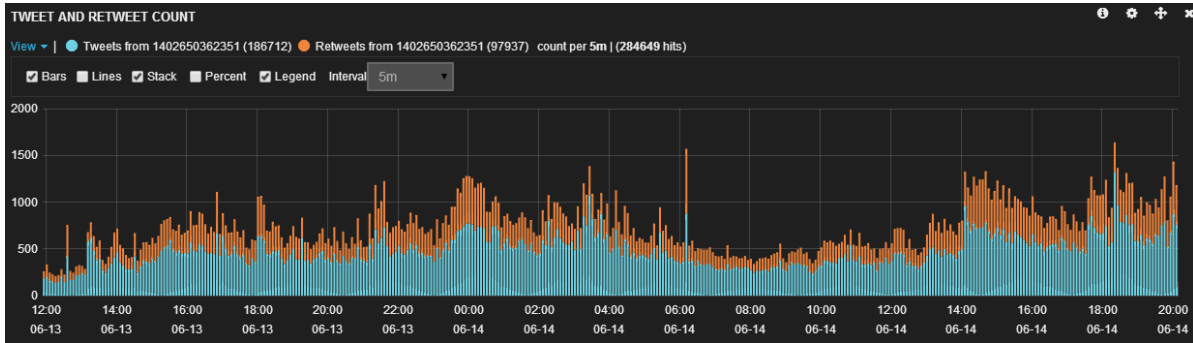


Figure 6.7: Tweet and retweet count from worldwide tweets, Use Case 2

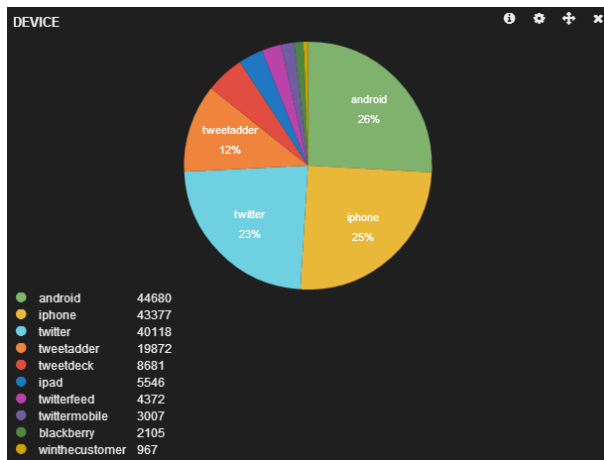


Figure 6.8: Devices used from worldwide tweets, Use Case 2

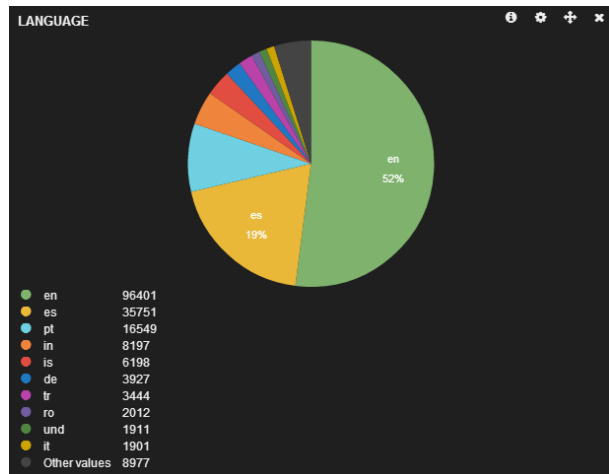


Figure 6.9: Language from worldwide tweets, Use Case 2

Word	Occurrences
miley	128817
cyrus	124657
barcelona	14133
selena	13992
gomez	12426
ikurriña	11181
fav	10689
love	9076
mileyformmva	8957
beautiful	8554

Table 6.3: Top 10 most common words from worldwide tweets, use case 2

Hashtag	Occurrences
mileyformmva	49140
bangerzincolumbia	2295
mileycyrus	1863
selenaformmva	1836
bangerztour	1291
ecuadorneedsbangerztour	1270
bangerztourmexico	1181
bangerztourbarcelona	937
miley	700
smilers	283

Table 6.4: Top 10 most common hashtags from worldwide tweets, use case 2

- Most common words: The top 10 words, from high to low, can be seen in Table 6.3.
- Most common hashtags: The top 10 hashtags, from high to low, can be seen in Table 6.4.
- Sentiment: The sentiment analysis shows that 84% of tweets were positive or very positive, 12% were neutral, and only 4% were negative or very negative. The sentiment mean over time is stable around 6.2 out of 9 before the match, but it goes to 6.5 the hour after the gig and at 12:00 the day after the gig. See Figures 6.13 and 6.10 for more details.
- Arousal: The arousal mean over time shows that the intensity of tweets is stable around 4.6, but increases to 4.9 during hour after the gig, and the day after the gig at 12:30 arrives to 5. See Figure 6.11 for more details.



Figure 6.10: Sentiment analysis over time from worldwide tweets, Use Case 2

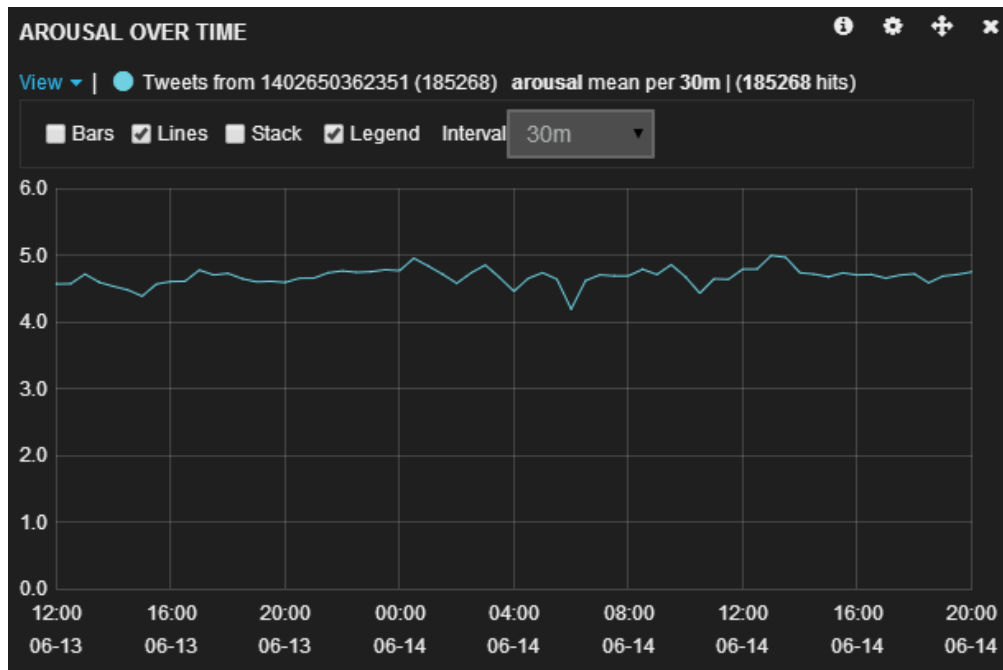


Figure 6.11: Arousal over time from worldwide tweets, Use Case 2



Figure 6.12: Tweet map from worldwide tweets, Use Case 2

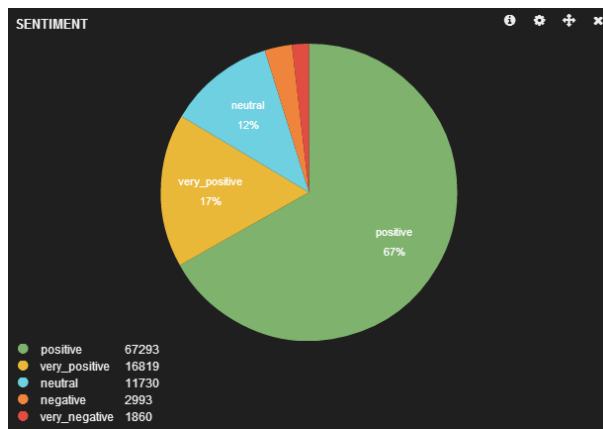


Figure 6.13: Sentiment analysis from worldwide tweets, Use Case 2

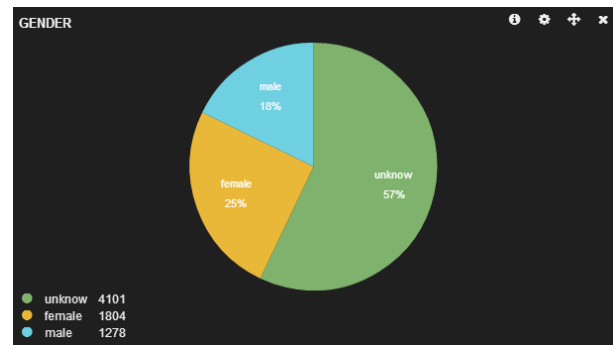


Figure 6.14: User's gender from worldwide tweets, Use Case 2

- Geoposition:** The coordinates of geopositioned tweets show that the most active countries were Spain (694), Brazil (689) and USA (497). Barcelona highlights with 341 tweets, much more than bigger cities like Rio de Janeiro (117) New York (47). Several countries have no tweets, but really curious is to see only one tweet coming from Japan, and only 6 tweets from Scandinavian countries (Norway, Finland, Sweden and Denmark combined). See Figure 6.12 fro the complete map.
- Gender:** The gender analysis shows that 25% of users were females and 18% were males. The unknown 57% is mostly due to strange user names, like "#mileyforMMVA" or "boys-miler". See Figure 6.14 for more details.

6.2.3 Concert only results

In order to analyse tweets talking about the gig, we used tweets that matched the previous criteria and also include once of the following words:

- barcelona
- españa
- spain

To do that, we used the filter "words:barcelona OR words:españa OR words:spain" in the Kibana filtering interface. The results are:

- Tweet count: The total number of messages matching the filters was 14954 tweets and 8988 retweets. The most active time zones were the hour and a half after the gig (from 23:30 to 01:00) with 1853 tweets and 1224 retweets per hour, and from 14:00 to 16:00 the day after the concert, with 1476 tweets and 1241 retweets per hour. See Figure 6.15 for more details.

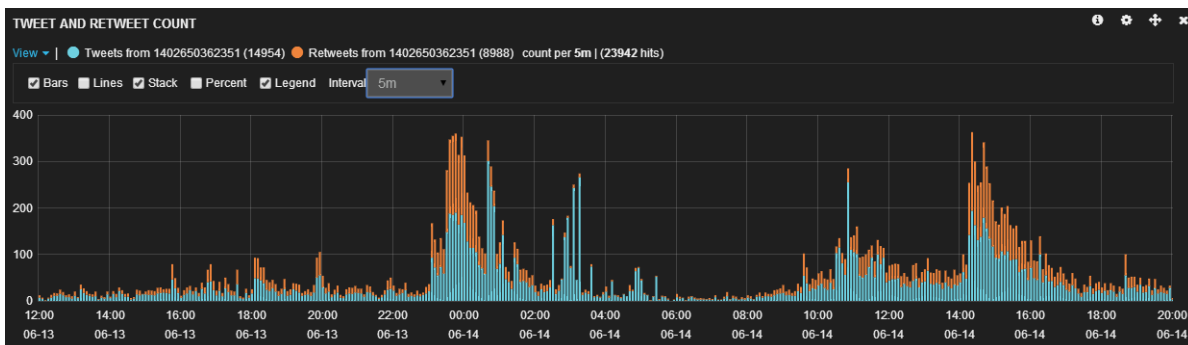


Figure 6.15: Tweet and retweet count from tweets about the gig, Use Case 2

- Device: Handheld devices were the main source. 5310 used an Android device (37%), and 2417 used an iPhone (17%). 5266 came from computers (37%). See Figure 6.16 for more details.
- Language: The main language when talking about the gig was Spanish, with 12091 tweets (81%), followed by English, with 1930 tweets (13%). See Figure 6.9 for more details.

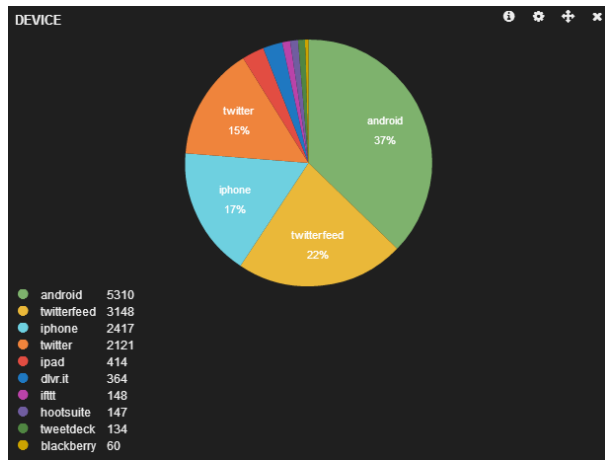


Figure 6.16: Devices used from tweets about the gig, Use Case 2

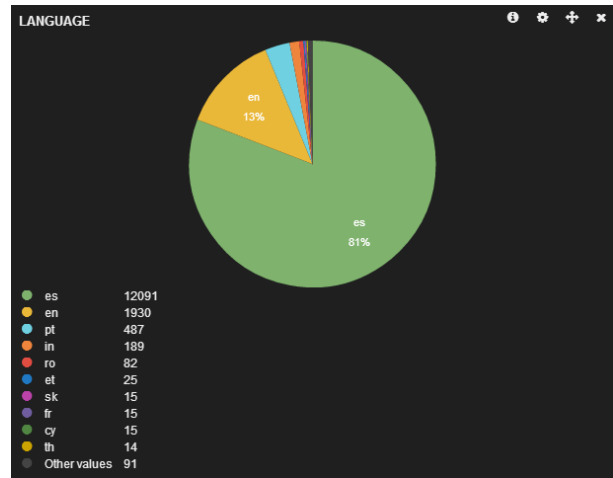


Figure 6.17: Language of tweets about the gig, Use Case 2

Word	Occurrences
barcelona	14111
miley	13388
cyrus	13279
ikurriña	7462
concierto	4324
ondea	3364
ondeando	2708
exhibe	1962
alarde	1924
musical	1454

Table 6.5: Top 10 most common words from tweets about the gig, use case 2

Hashtag	Occurrences
mileyformmva	1000
bangerztour	196
mileycyrus	174
ridicula	113
sangreuskaldun	109
actitud	109
bangerz	84
bangerztourbarcelona	80
bangerztourb	77
albert_stanlie	50

Table 6.6: Top 10 most common hashtags from tweets about the gig, use case 2

- Most common words: The top 10 hashtags, from high to low, can be seen in Table 6.5.
- Most common hashtags: The top 10 hashtags, from high to low, can be seen in Table 6.6.
- Sentiment: The sentiment analysis shows that 90% of tweets were positive or very positive, 8% were neutral, and 2% were negative or very negative. The sentiment mean over time is stable around 6.3 out of 9 before the gig, but it goes to 7 several times after the gig until 04:00 am, then stabilizes around 6.5 the day after. See Figures 6.21 and 6.18 for more details.
- Arousal: The arousal mean over time shows that the intensity of tweets is stable around 4.5, but increases to 5.1 after the concert until 08:00 am, where decreases to 4.1 for a while to stabilize again at 4.5 around 09:00. See Figure 6.19 for more details.
- Geoposition: The coordinates of the geopositioned Tweets shows that the most active country when talking about the gig was Spain itself (109), especially around the area of Barcelona (63). Apart from Spain, other countries that talked about the gig include Italy (3) or Brazil (2). See Figure 6.20 for the complete map.
- Gender: The gender analysis shows that 25% were male and 22% were female. The unknown 53% is because of strange user names, as happened before. See Figures 6.22 for more details.

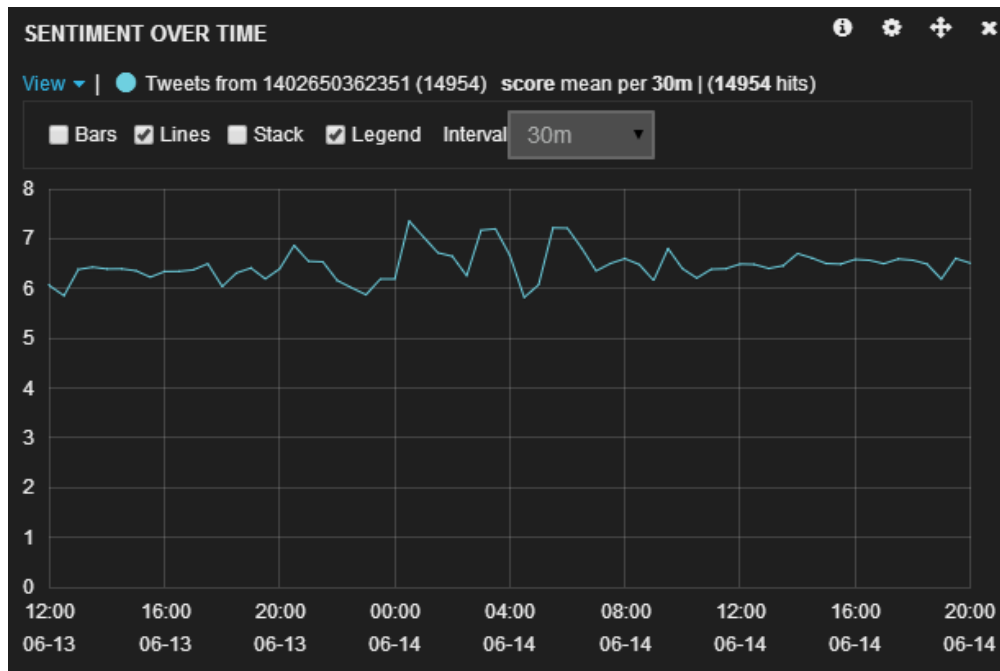


Figure 6.18: Sentiment over time from tweets about the gig, Use Case 2



Figure 6.19: Arousal over time from tweets about the gig, Use Case 2

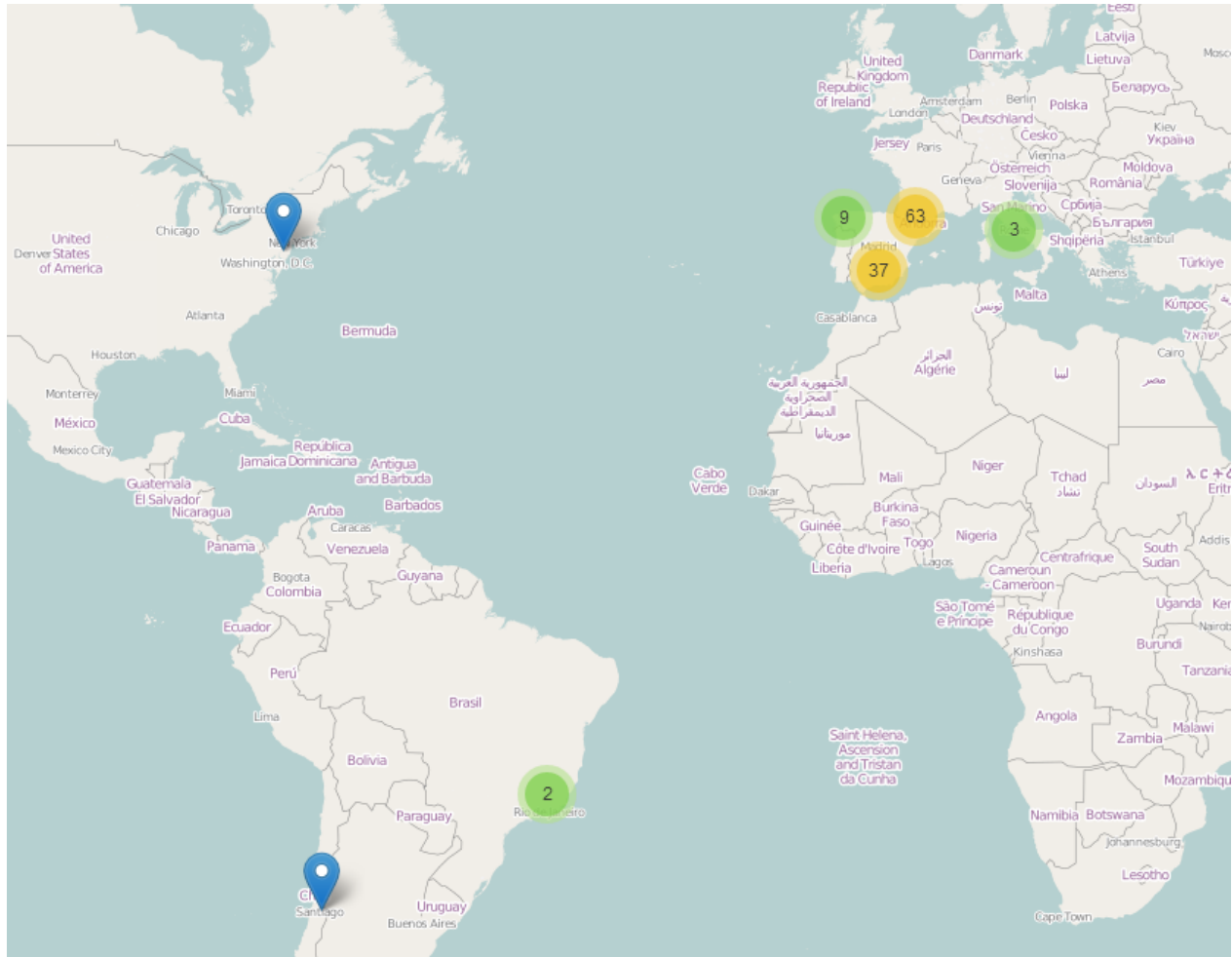


Figure 6.20: Tweet map from tweets about the gig, Use Case 2

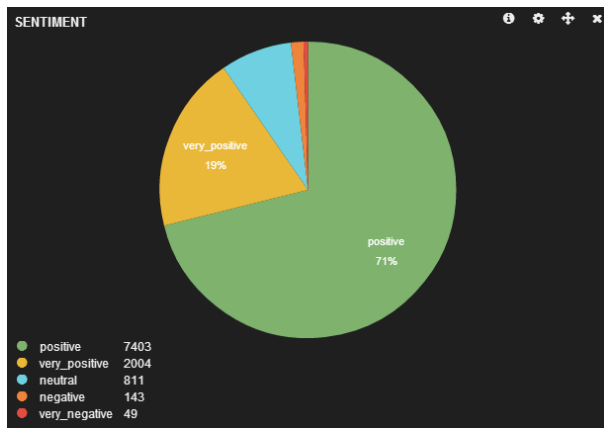


Figure 6.21: Sentiment analysis from tweets about the gig, Use Case 2

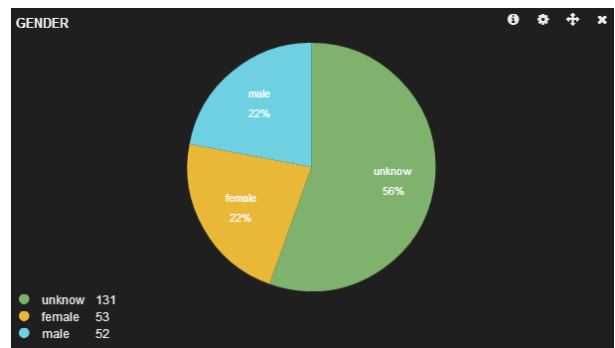


Figure 6.22: User's gender from tweets about the gig, Use Case 2

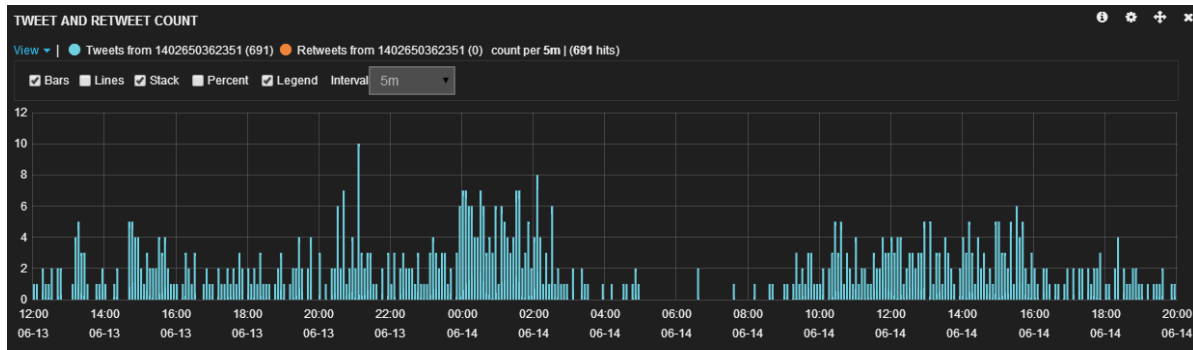


Figure 6.23: Tweet count from Spanish tweets, Use Case 2

6.2.4 Spain results

Finally, in order to see the Spanish results, we apply a filter "place.country_code:ES" using the Kibana filter interface. The results show:

- Tweet count: The total number of tweets sent from Spain was 691. The most active time zones were the hour before the gig (from 20:30 to 21:30) with 55 tweets, and the 2 and a half hours after the gig (from 00:00 to 02:30) with 55 tweets per hour. See Figure 6.23 for more details.
- Device: Handheld devices were the main source. 344 used an Android device (50%), and 157 used an iPhone (23%). From the 26% remaining, only 62 came from computers (13%). See Figure 6.24 for more details.
- Language: The main language from Spain was Spanish, with 378 tweets (55%), followed by English, with 170 tweets (25%). See Figure 6.25 for more details.
- Most common words: The top 10 hashtags, from high to low, can be seen in Table 6.7.
- Most common hashtags: The top 10 hashtags, from high to low, can be seen in Table 6.8.
- Sentiment: The sentiment analysis shows that 82% of tweets were positive or very positive, 11% were neutral, and 7% were negative or very negative. The sentiment mean over time is stable around 6.2 out of 9 before the match, but it goes to 6.5 the hour after the gig and at 12:00 the day after the gig. The sentiment over time drops to 0 twice because there were no results for that dates. See Figures 6.29 and Figures 6.26 for more details.

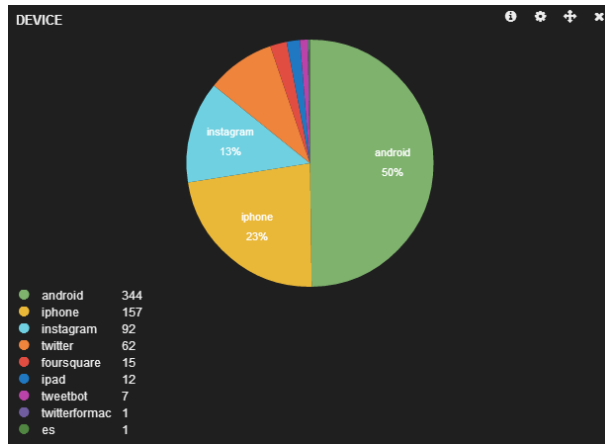


Figure 6.24: Devices used from Spanish tweets, Use Case 2

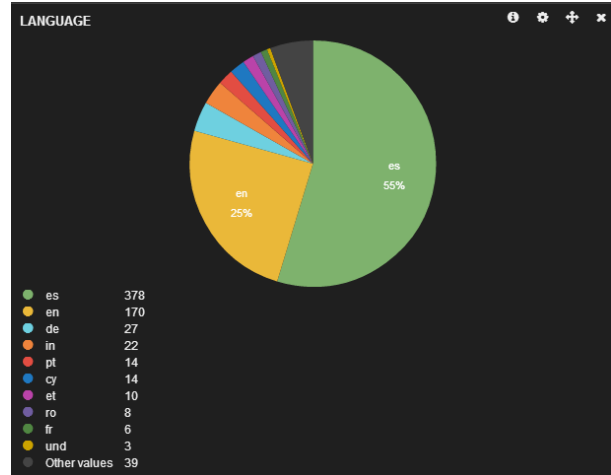


Figure 6.25: Language from Spanish tweets, Use Case 2

Word	Occurrences
miley	316
cyrus	291
barcelona	108
concierto	102
palau	76
jordi	70
sant	68
ikurriña	59
love	26
ver	22

Table 6.7: Top 10 most common words from Spanish tweets, use case 2

Hashtag	Occurrences
bangerztourbarcelona	49140
bangerztour	2295
barcelona	1863
bangerz	1836
mileycyrus	1291
mileyformmva	1270
zona40mileyenmadrid	1181
bangerzshow	937
concert	700
hechoscyrus	283

Table 6.8: Top 10 most common hashtags from Spanish tweets, use case 2

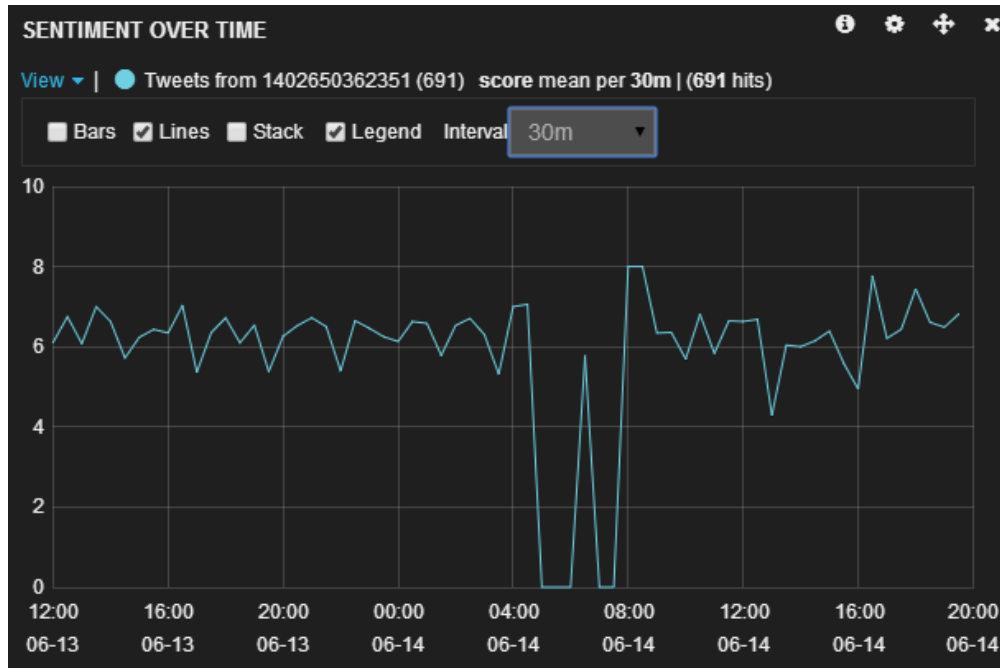


Figure 6.26: Sentiment over time from Spanish tweets, Use Case 2

- Arousal: The arousal mean over time shows that the intensity of tweets is stable around 4.6, but increases to 4.9 during hour after the gig, and the day after the gig at 12:30 arrives to 5. As before, the arousal over time drops to 0 twice because there were no results for that dates. See Figure 6.27 for more details.
- Geoposition: The geoposition of the Tweets shows that the most active autonomous was, by far, Catalonia, with 343 tweets. Other active autonomous communities were Madrid (60) and Andalucia (69), while La Rioja (0), Baleares (3) and Cantabria (3) were the least active. See Figure 6.28 for the complete map.
- Gender: The gender analysis shows that 25% were male and 22% were female. The unknown 53% is because of strange user names, as it happened before. See Figure 6.30 for more details.

6.2.5 Conclusions

From all the previous results, we can see three different events:

- The most talked topic worldwide was the nomination of Miley Cyrus for the Much Music

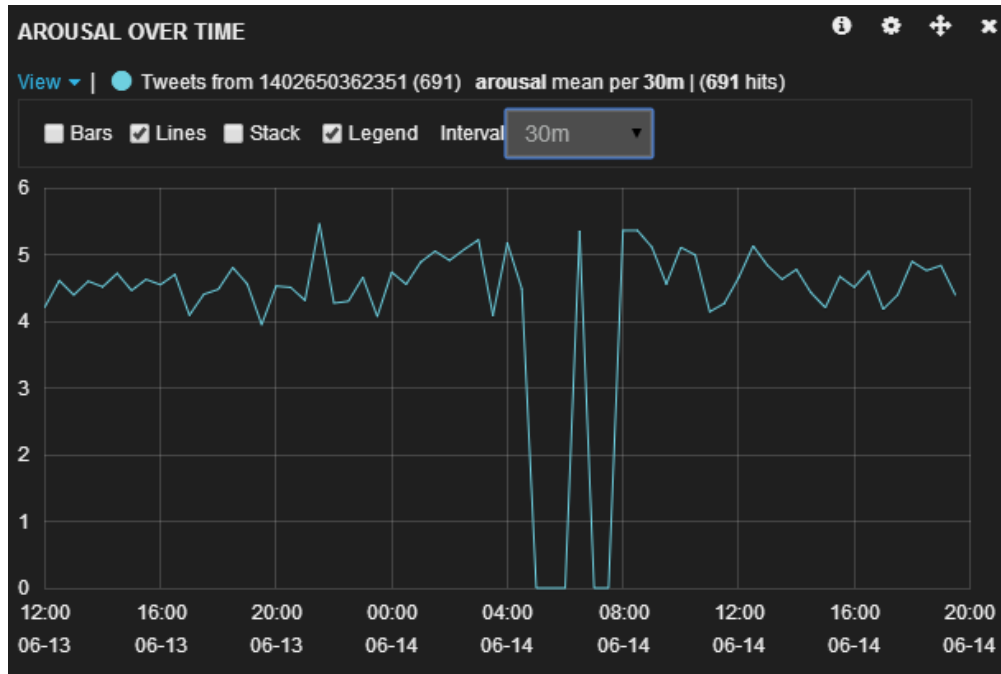


Figure 6.27: Arousal over time from Spanish tweets, Use Case 2

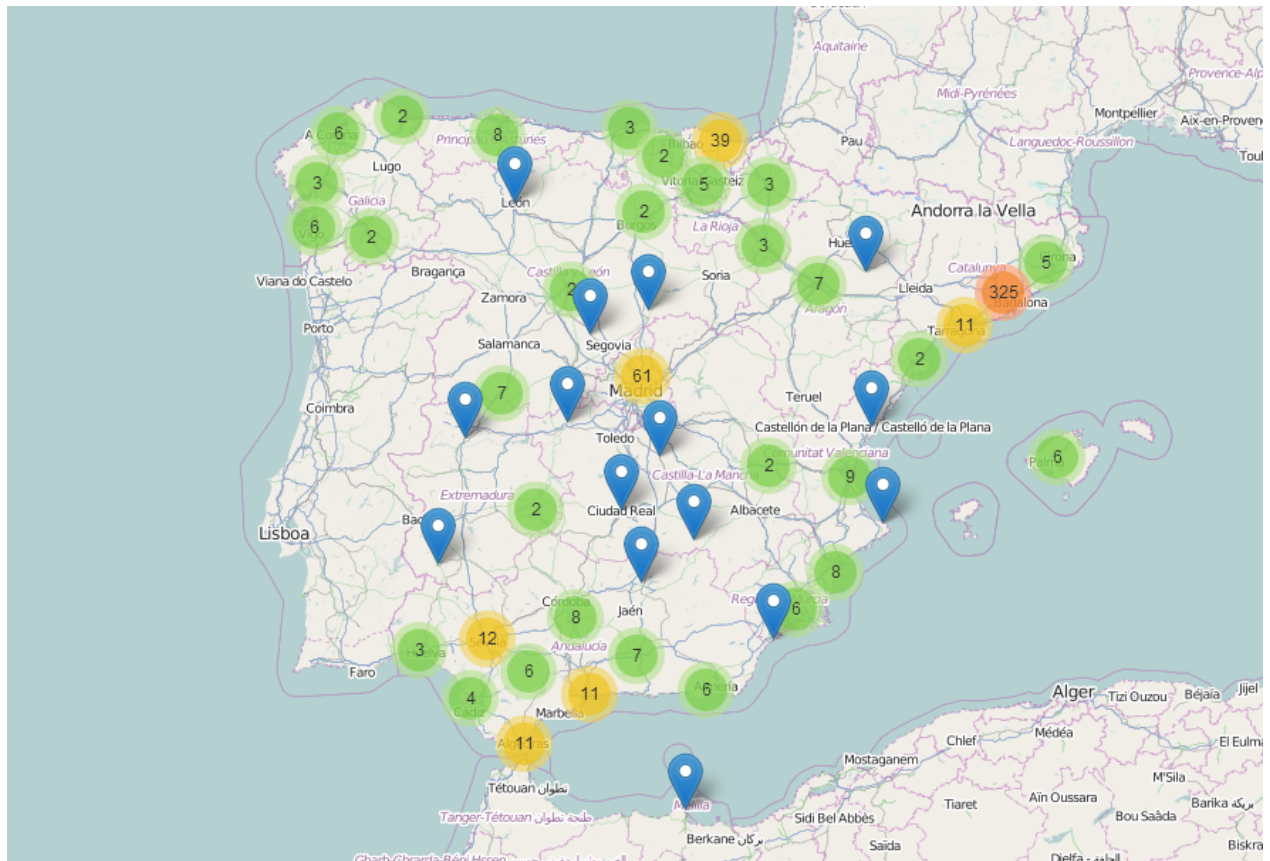


Figure 6.28: Tweet map from Spanish tweets, Use Case 2

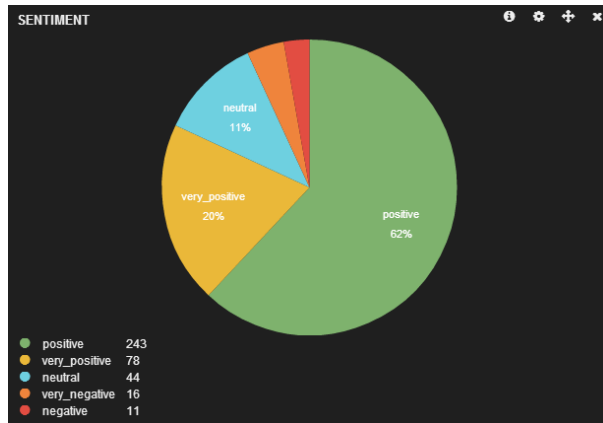


Figure 6.29: Sentiment analysis from Spanish tweets, Use Case 2

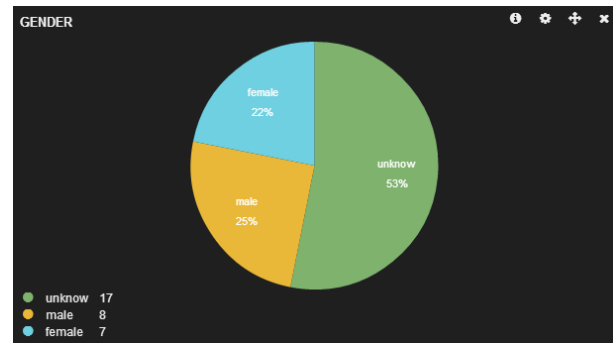


Figure 6.30: User's gender from Spanish tweets, Use Case 2

Video Awards (MMVA), an award given by the Canadian music channel "Much Music" [59]. This can be seen in the word "mileyformmva" appearing as the eighth most common word and the most common hashtag. On the other hand, neither tweets talking about the gig nor tweets from Spain mention this, which shows they were fully focused on the Barcelona concert.

- The singer Selena Gomez was very mentioned for 2 different reasons: The first one, Miley Cyrus carrying her cardboard cut-out on stage during a previous concert in Milan, in June 8 2014 [60]; the second one, because a lot of tweets voting for Miley Cyrus as the MMVA winner were also voting for Selena Gomez, which shows Miley Cyrus fans are often Selena Gomez fans also. This can be seen in the words "Serena" and "Gomez", and the hashtag "serenaformmva". Again, tweets from the gig or from Spain do not mention this topic, they only talk about the concert.
- The last event was Miley waving the Basque flag during the Barcelona concert. Worldwide, "ikurriña" had more than 11000 mentions, being the sixth more mentioned word in Twitter when talking about Miley Cyrus, but where this topic had a lot of relevance was on tweets talking about the concert, where 50% of the tweets contained words like "ondea" (waves), "ondeando" (waving), "ikurriña" (the name of the flag), "exhibe" (shows) or "alarde" (show). When focusing on Spanish tweets, this was a quite mentioned topic, "ikurriña" appearing again in the top 10, but they were more focused on the whole concert

rather than in this incident.

Apart from identify these 3 different events, we can extract some general conclusions:

- Barcelona was the most mentioned word apart from the name and surname of the artist, so the concert had a very important repercussion in Twitter. This can be seen also in the hashtag "bangerztourbarcelona", in the huge increase in the tweet count on tweets talking about the gig during the hours after it was finished, and in the coordinates map, being Spain the most active country worldwide.
- Even the concert had an important repercussion in Twitter, this was mostly from people in Spain. All most common words were in Spanish, more than 12000 tweets were written in Spanish (and only 1700 in English), and more than 95% of the geopositioned tweets talking about the concert come from Spain.
- The concert had a positive effect on tweets. When talking about the gig, the sentiment over time increases from 6.2 to 7 during the hours after the concert, and more than 90% of tweets were positive (compared to 84% worldwide and 82% in Spain).
- The gender analysis does not show reliable results. Miley Cyrus fans, being teenagers, usually introduce as their Twitter full name support messages ("I Love Miley", "Miley Cyrus Only" or "MileyForMMVA") or non-sense names ("Smiler" or "Bad Girl").
- Users tweeting from Spain seemed to use much more handheld devices than those tweeting worldwide or talking about the concert (17% against 37% when talking about the gig, or 40% worldwide). Analysing the messages those people sent, it seems to be most of them were in their way to the concert or already there, which made them use their smartphones. We can also see that Android was much more used than iPhone in Spain (50% against 23%), but it was even worldwide (26% against 25%).
- Only in tweets from Spain the concert seems to be the main topic, with words like "palau", "sant", "jordi" (Palau Sant Jordi is the stage where the concert was played) or "concerto" (gig). Tweets worldwide, even they talk about Barcelona, focus a lot more in the MMVA

awards and in the incident with Selena Gomez in Milan; and those talking about the concert talk almost exclusively about Miley Cyrus waving the Basque flag during the concert rather than the concert itself.

Chapter 7

Planning

In this chapter we detail the project planning, including the schedule and deadline for this project and how we are dividing the available time.

7.1 Methodology

When choosing the development methodology, it is important to consider the limited schedule, the changing requirements from the companies the software is intended to, and the fact that only one developer works at this project. Because of that, we use an Extreme Programming methodology [58], where the new features are constantly revised by the project manager, the project director and the client, using short development cycles and adapting the code to the changes proposed with the feedback received (see Figure 7.1).

It's also important to notice that code must be really well structured and well documented to make future implementations easy and clean, including several git branches to have the control of the code in every stage of development.

7.2 Schedule

The project duration is estimated at 4 months, starting February 17 and finishing June 17. The workday consists on 30h per week, from 9am to 3pm, from Monday to Friday, and there is a one-week holiday from April 28 to May 4. The project manager has his job at CA Labs, so he



Figure 7.1: Planning and feedback loops in extreme programming

only dedicates 25% of his time (around 7.5h per week) during the planning and analysis time. The project tutor has also his job at UPC, so he only dedicates 50% of the time (around 15h per week) during the planning and analysis time, and then 25% of his time (around 7.5h per week) to the analysis and testing part of each sprint. The project director provides his help occasionally, without a fixed schedule. See Figures 7.3 and 7.5 to see the detailed resources diagram.

7.3 Action plan

The action plan consists of 4 steps: Planning, Analysis, Development and Maintenance. See Figures 7.2 and 7.4 for the Gantt chart of the action plan.

7.3.1 Project planning

The first step is where the first definition of the project is done, using 4 phases:

- Scope: During this phase we define the problem to solve, its scope and its objectives, set the tools and methodology to use, and analyse the risks of the project to avoid failures.
- Planning: This phase includes the specification of the project steps, resources available, schedules and timetables.

- Budget: During this phase we specify all the expenses related to this project, justifying that it is economically viable for CA Technologies.
- Presentation: Then end phase is where we show the results of the planning to the company. It also includes a period where changes can be made to accommodate the feedback received.

7.3.2 Project analysis

The second step involves a detailed analysis of the project and a design based on that analysis. This step involves several meetings with CA Labs, the project tutor and project director departments. Includes the following 3 phases:

- Objectives & requirements: The first phase is intended to develop a full, detailed list with all the objectives for this project, all the requirements that it must accomplish, and the risks it must avoid.
- Use cases: Includes to develop a use case diagram that covers all the project objectives.
- Interface design: In the last phase we develop a first mock-up of the graphical interface, so we can receive feedback from the client.

7.3.3 Project development

As we mentioned before, we are using the extreme programming methodology [58], which means that the development is structured into small sprints, and each sprint contains 4 phases: analysis, implementation, testing and integration. Those sprints are:

- Initial set-up: This initial set-up includes installing the OS and the software needed in the server, and also the configuration of the working laptop so it is ready to manage the servers and also develop and test the code.
- Data Access Layer: The first sprint is used to set-up the database and develop those classes used to load and save data from them.

- User Interface Layer: During the second sprint we develop the user interface layer, based on the design and requirements we have. As we stated, the interface is as easy and usable as possible, and fully responsive.
- Business Logic Layer: The third sprint is where we develop the business layer of the system, including the master node, the crawling node and all the enrichers.
- Final integration: The last sprint is intended to integrate all the parts, improve all the parts if needed, and make the final release.

7.3.4 Project maintenance

The last step is intended to document and doing little fixes in the code.

- Documentation: During this part we write down the user manual and prepare the different Kibana panels.
- Bug solving: At the same time, we fix any bug we may have missed during the sprint tests.
- Feedback: Finally, we receive feedback from CA Labs and fix little design or implementation things that were not as expected.

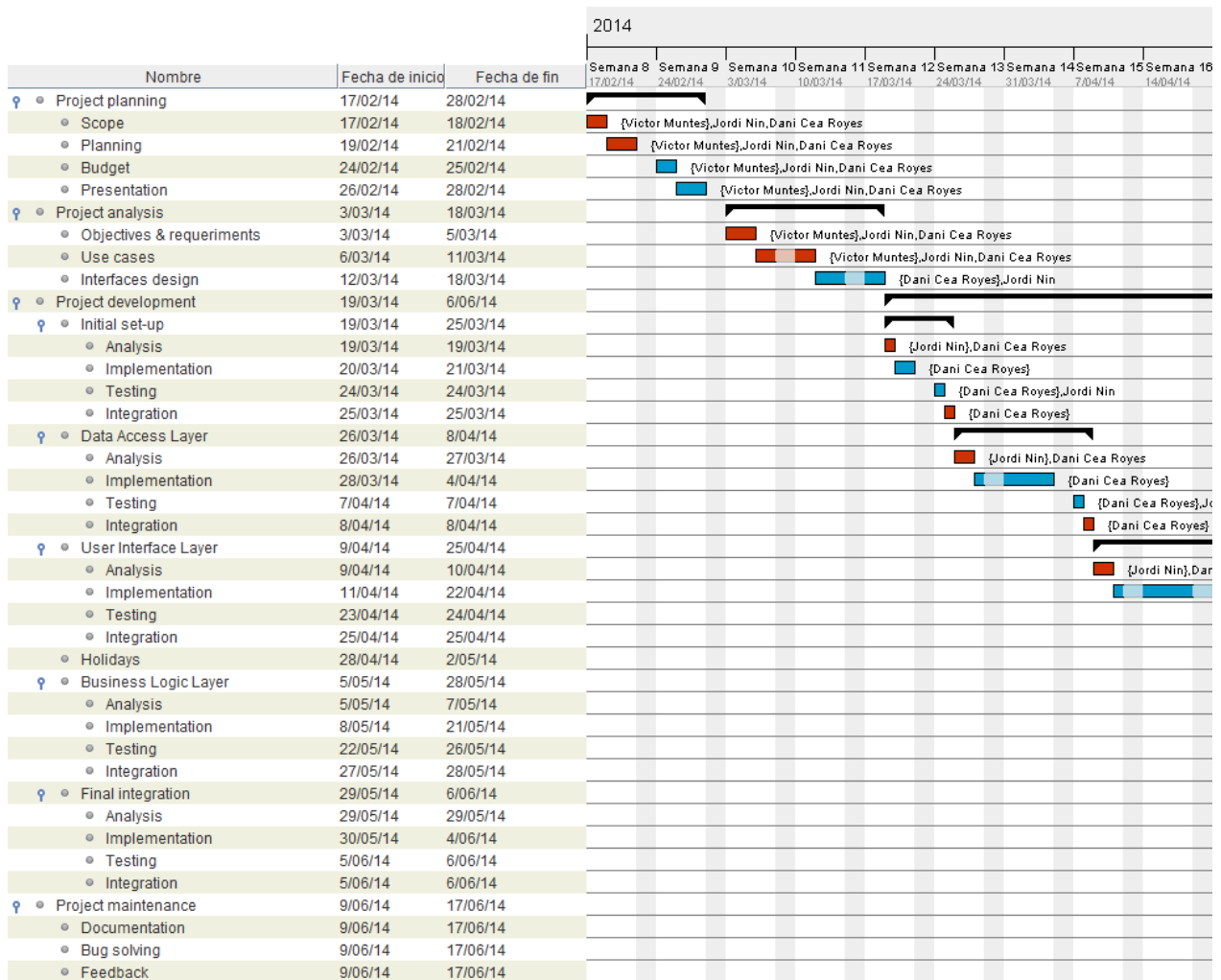


Figure 7.2: Planning Gantt chart from February 17 to April 20

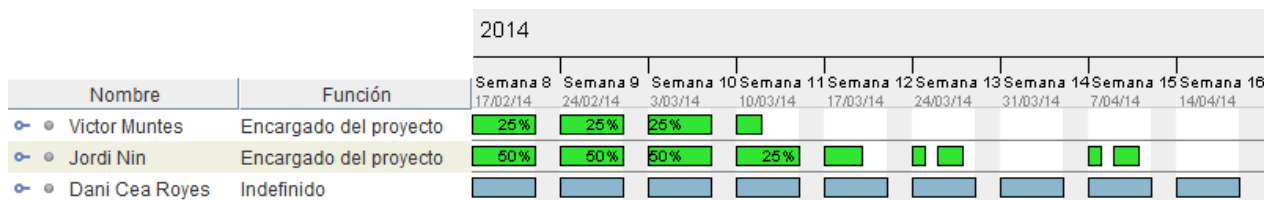


Figure 7.3: Resources diagram from February 17 to April 20

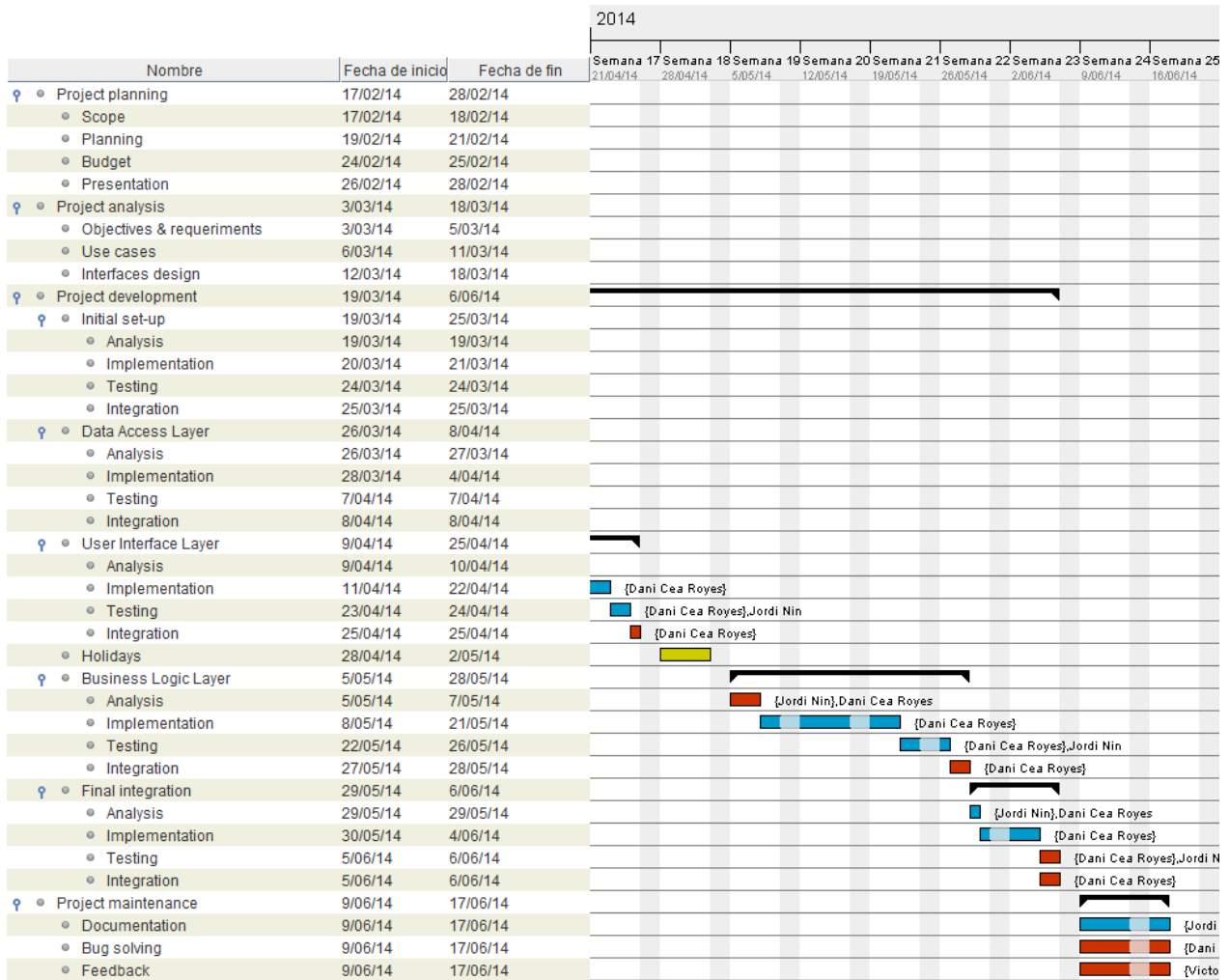


Figure 7.4: Planning Gantt chart from April 21 to June 17

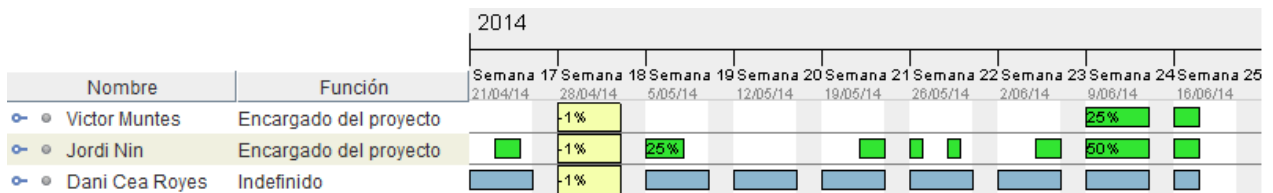


Figure 7.5: Resources diagram from April 21 to June 17

7.4 Deviations

During the project development, we had the following time deviations from the original plan:

- User Interface Layer -> -48 hours: The user interface layer was planned to be developed from April 9 to April 25, but this plan was before we discovered about Kibana [41] as the output interface, which made much easier the whole interface development, only needing from April 9 to April 15. This produced a deviation of -8 days, or -48 hours.
- Business Logic Layer -> +60 hours: The business logic layer was planned to be developed from May 5 to May 28. Because we had extra time from the user interface, we decided to add some topics the initial scope was not considering, like sentiment or gender analysis. The business logic layer, then, was started at April 16. At the end, the implementation was harder than expected due to the lack of documentation, and took two extra days to finish, May 30 instead of May 28, which produces a total deviation of +10 days, or +60 hours.
- Final Integration -> -24 hours: The final integration was planned to be developed from May 29 to June 2, but the business logic part took two extra days that made us start this June 2 instead. We realized final integration was easier than expected, and we only needed 1 day for analysis + integration, and another one for testing, so it only took until June 4 instead of June 6, with a total deviation of -4 days or -24 hours.
- Maintenance -> +12 hours: The maintenance step was planned from June 9 to June 17, but it was started June 5 thanks to the extra time we had from final the final integration. The feedback was very positive and we only needed one day (June 11), so the documentation part and bug solving covered all the time from June 5 to June 17, a total deviation of +2 days, or +12 hours.

Chapter 8

Budget

This chapter gives a deep analysis of the budget. This includes how we receive the budget, the total amount, and how we divide this budget into the different expenses. Because this is conceived as an internal product (hence, is not to be sold), we do not have any other income rather than the initial founding.

8.1 Budget Analysis

The project funding entirely depends from CA Labs and from Computer Architecture Department at Universitat Politècnica de Catalunya, which means that our budget is dependent on third parties. Despite we cannot control the amount of money spent, we have some flexibility on how to spend it.

8.2 Budget Estimation

CA Labs provides us with 6450€ to develop this project through an agreement with UPC, and the university also offers us, free of charge and only during the project develop, the infrastructure and servers needed. Even that, we estimate a total budget for the project, and then go into more detail about what exactly is included in the real budget. This estimation is divided into 4 parts.

8.2.1 Human resources

This part of the budget includes all expenses related to salaries. Because only one developer works on this project, this person is responsible for the analysis, development and testing of the software. The project manager salary is paid by CA Labs, and the project manager and project director salaries are paid by UPC, so even we estimate them, they do not affect our budget. Also, the project owner and the project manager are not present all the time, and we can not estimate the salary from the project director as he does not have a fixed schedule. All salaries include taxes.

Role	Price per hour	Hours	Total
Project manager	60.00€	42	2520€
Project tutor	35.00€	110	3850€
Developer	9.50€	574	5453€
Total			11823€

Table 8.1: Human resources budget

8.2.2 Hardware

Except for the computer used by the developer, all the hardware and infrastructure are provided either by the UPC or by CA Labs, so we estimate the rental cost of that hardware during the project. One last consideration is that deployment servers are not needed during the entire project, just once we have deployed the end product.

Product	Price	Units	Useful life	Amortization	Total
Asus n53sv	750€	1	5 years	62.5€	62.5€
Samsung Galaxy II	270€	1	3 years	37.5€	37.5€
Test Server	30€/month	5	-	-	750€
Deploy Server (UPC)	330€/month	1	-	-	660€
Deploy Server (CA Platform)	1000€/month	1	-	-	1000€
Total					2510€

Table 8.2: Hardware budget

8.2.3 Software

This part of the project includes all the software expenses of the project.

Product	Price	Units	Months	Total
Microsoft Windows 8.1 x64	0€	1	Unlimited	0€
Sublime Text 3	55€	1	Unlimited	55€
Google Chrome	Free	1	Unlimited	0€
Putty	Free	1	Unlimited	0€
WinSCP	Free	1	Unlimited	0€
Dropbox	10€/month	1	5	50€
Node.js	Free	1	Unlimited	0€
Npm	Free	1	Unlimited	0€
Git	7€/month	1	5	35€
ShareLatex	Free	1	Unlimited	0€
Ubuntu 12.04	Free	1	Unlimited	0€
ElasticSearch	Free	1	Unlimited	0€
Total				140€

Table 8.3: Software budget

8.2.4 Other expenses

This last part adds all the expenses that are not included before. All expenses include taxes.

Product	Price	Months	Total
Office rental	450€/month	5	2250€
Energy	35€/month	5	175€
Water	25€/month	5	125€
Heat & Air	30€/month	5	150€
Internet	40€/month	5	200€
Telephone	10€/month	5	50€
Total			2950€

Table 8.4: Other expenses budget

8.2.5 Total budget

The part of the budget covered by UPC and CA Labs includes:

- Human Resources: Project manager salary (2520€) + Project tutor salary (3850€)
- Hardware: Server related (deploy and test servers, 2410€)
- Software: Git repositories (35€)
- Other expenses: All of them (2950€)

Concept	Total
Human resources	11823€
Hardware	2510€
Software	140€
Other Expenses	2950€
Total (project)	17423€
Part of the budget covered	-11765€
Total (assigned budget)	5658€

Table 8.5: Total budget

The total expenses are 5658€ and the total budget is 6450€, which gives us a benefit of 792€

8.3 Budget control

The budget control is done in each one of the 8 meetings planned at the end of each spring. In each meeting, the real budget is compared to the estimated budget and adjusted to correct the mismatches, and the result is given to CA Labs and UPC so they can check it.

8.4 Deviations

The only economical deviation we had during the project development was the Samsung Galaxy II smartphone, which stopped working unexpectedly on June 7, during the final testing phase, where the smartphone was needed to test the responsiveness of the system. This had 2 consequences:

- New smartphone -> +2.30€: A new smartphone was bought: Nexus 5, with a price of 309€. Amortized to 3 years, it gives those 8 days an extra charge of 2.30€.

- Samsung Galaxy II shorter amortization time -> +4.70€: Because the final amortization of Samsung Galaxy II was 2 years 7 months instead of 3 years, the price then goes from the original 37.50€ to 42.20€ (without the last 8 days in which the smartphone is not usable anymore), an extra charge of 4.70€

This give us a total extra charge of 7€ added to the total expenses, from 5658€ to 5665€, and decreasing the benefit from 792€ to 785€.

Chapter 9

Conclusions

In this last chapter we present the final conclusions of this project and the future improvements that may be done.

9.1 General conclusions

During this project, we have observed that social media analysis tools offer very interesting services to the society, although there is still room for further improvements.

The core of this project is the data provided by social network services. In our case, the Twitter streaming API provides us a great social data source, almost irreplaceable. Despite this, it would be really interesting to add new data sources, especially Instagram and Foursquare, to complement the data that Twitter provides us.

NoSQL technologies stand out for this project. Elasticsearch and Couchbase have become crucial for the success of the system, offering a great scalability, a flexible data domain, and the possibility to handle large volumes of data easily. Without these NoSQL technologies, this project would have been more complex to achieve.

Bootstrap helps us for the usability objective, providing a responsive interface development framework. Kibana, the display tool used for this project, also takes advantage of the responsiveness from Bootstrap to offer a multi-platform display interface. It is also important to distinguish the role played by the Google Maps API, offering us an easy user interface for maps.

For the programming languages, it is important to highlight Node.js and their asynchronous

model. It provides us the possibility of having multiple threads to parallelize the code, thus improving system performance. Their open source module system allows us to reuse lots of external libraries during the development, saving a lot of time.

As a general result, we have a functional, scalable and distributed social media analyser based on real-time big data technologies, ready to extend to other data sources and able to display aggregated results. It also offers to others developers the flexibility of adding new data enrichment processes and output panels. Therefore, this project has successfully covered all the objectives defined.

9.2 Personal conclusions

Personally, developing my bachelor thesis with CA Labs allowed me to learn a lot of interesting things. It has been a pleasure to work with big data experts like Jordi and Marc, and I have learned a lot about NoSQL technologies, big data analysis, and natural language processing. At the same time, working within a company improved my team work and communication skills, and talking with foreign people from CA Technologies plus doing this report in English improved a lot my third-language skills.

Also, it was really interesting to discover the world of research, totally unknown for me before. It is much more rewarding for me to learn and use the newest available technologies to solve real-world challenges instead of using technologies dictated by companies to solve client problems. I highlight the participation in my first research paper, accepted in the 11th International Conference on Modeling Decisions for Artificial Intelligence (MDAI). It made me feel really proud to contribute to the society with my work.

9.3 Future work

The implementation of this project has left behind some topics, either for time limitations or because they were out of the scope, and also there are parts of it with an improvement margin.

The first and most important one, the quality of the enrichers could be better. Natural language analysis is an interesting field, but it is also really wide and hard. The stemmer enricher

does not always work as expected, and the sentiment analysis is not as precise as it should be, in part due to my inexperience for this topic, but also because tweets contain slang words, shortened words, irony and other things that make the sentiment guessing harder.

We could also improve the user data context. The gender guessing does not work as expected and relies on an external service, and we have no way to discover important things from users like age, nationality or purchasing power. Social context data is also an important lack, like creating social graphs with the user followers, or tracking down geolocalized users.

Lastly, another improvement for the system could be offer the possibility to distribute the enrichers. Right now, all enrichers are separated classes of the same system. The idea would be to offer those enrichers as separated modules, in order to distribute them in different modules and not saturate the crawling nodes. It is not a current problem for the system, but it would be in the future if new enrichment processes are added.

Chapter 10

Social and environmental impact

This section details the social and environmental impact our system has in the society, divided in social impact, economic impact and environmental impact.

10.1 Social impact

As we analysed in section 3.1.3, social networks have several issues for society. Each downside has been prevented in our system in the following way:

- Privacy: Currently, the implementation of the system has not included any personal information about the users like phone numbers or address, even we still have to ensure adding future sources does not provoke a violation of privacy. Also, we do not sell any private information, the application is intended for private purposes.
- Unauthorized access: We do not authenticate any user with any social network, we just extract public information those social networks provide, so this is not an issue for our system.
- Child safety: To ensure child safety in social networks, we simply do not provide any personal information, and neither provide any access to interact with any social network.
- Cyber-bullying: As we already mentioned before, we do not provide any way to interact with the social network users, so this is not an issue for this project.

- Interpersonal communication: This issue goes in the same line as before, as long as we do not provide any interactions with social networks themselves, this is not an issue for this project.

10.2 Economic impact

Our application set the bases to create social media monitoring tools fully distributed and scalable that do not require powerful hardware to provide big data analysis efficiently. The fact that hardware can be added and removed easily makes this project extremely adaptable to different configurations, providing a way to optimize machines at maximum and, consequently, saving a lot of money.

Apart from being scalable and distributed, the system is developed completely using open source tools, removing all software costs and licences and providing an easy way to customize the system if needed.

10.3 Environmental impact

Optimizing the hardware resources used by the system also has a positive effect on the environmental footprint of this project, due to avoid unused machines wasting energy. The ability crawling nodes have to gather more than one stream on the same machine provides a way to use resources completely, avoiding idle CPU time at maximum. Whenever crawling nodes are idle, the process is consuming less than 1% of the CPU time to keep itself alive.

Aside from improving the environmental impact, we followed some simple rules during the development of this project, in order to be as green as possible:

- Development laptop is turned off whenever it is not used.
- The system runs on virtual machines, sharing the resources with other systems and using the hardware more efficiently.
- The room temperature is no more than 21 ° during the winter months and no less than 24° during the summer months, and the lights are always turned off when no one is working.

- Production servers have been off until the code was ready, and once the code is moved, tests servers have been immediately turned off.

Bibliography

- [1] Il-Horn Hann, Siva Viswanathan, and Byungwan Koh. "The Facebook App Economy". Center for Digital Innovation, University of Maryland, 2011.
- [2] Radian6. <http://www.salesforcemarketingcloud.com/> (Accessed 10/06/2014)
- [3] Trackur. <http://www.trackur.com/> (Accessed 10/06/2014)
- [4] Apache Lucene webpage http://lucene.apache.org/core/2_9_4/queryparsersyntax.html (Accessed 10/06/2014)
- [5] S. Shyam Sundar. "Journal of Computer-Mediated Communication". Volume 13, Issue 1, pages 210–230. The Pennsylvania State University, October 2007
- [6] Liebeskind, Julia Porter. Social Networks, Learning, and Flexibility: Sourcing Scientific Knowledge in New Biotechnology Firms. Vol. 7, No. 4, pages 428–443. August 1996
- [7] Sonia Livingstone, David R Brake. "On the Rapid Rise of Social Networking Sites: New Findings and Policy Implications". *Children & Society* 24, pages 75-83 July 2012.
- [8] Phipps Arabie, Yoram Wind. "Marketing and Social Networks". In Stanley Wasserman and Joseph Galaskiewicz, "Advances in Social Network Analysis: Research in the Social and Behavioral Sciences". Pages 254-273 Sage Publications. Thousand Oaks, California, 1994.
- [9] "Social media, cellphone video fuel Arab protests". *The Independent (UK)*, online version. February 27, 2011. <http://www.independent.co.uk/life-style/gadgets-and-tech/social-media-cellphone-video-fuel-arab-protests-2227088.html>

- [10] Nikolas Gvosdev "The Realist Prism: Politics vs. Social Media in the Arab Uprising". World Polytics Review website March 4, 2011. <http://www.worldpoliticsreview.com/articles/8089/the-realist-prism-politics-vs-social-media-in-the-arab-uprising>
- [11] John S. Luo. "Social Networking: Now Professionally Ready" Primary Psychiatry website. February 1, 2007 <http://primarypsychiatry.com/social-networking-now-professionally-ready/>
- [12] Susan B. Barnes "A privacy paradox: Social networking in the United States" First Monday online journal. September 4, 2006 <http://firstmonday.org/ojs/index.php/fm/article/viewArticle/1394/1312>
- [13] Sarah Elkins "Facebook's New Ad Strategy" Newsweek online journal March 13, 2010 <http://www.newsweek.com/facebooks-new-ad-strategy-96477>
- [14] Tricia Phillips "ID fraud continues to rise as 80,000 hit by crooks this year" Daily Mirror, on-line version October 19, 2011 <http://www.mirror.co.uk/money/personal-finance/id-fraud-continues-to-rise-as-80000-86358>
- [15] Carrie-Ann Skinner "Unauthorised access of social networking profiles surge" PC Advisor website October 19, 2011 <http://www.pcadvisor.co.uk/news/security/3311844/unauthorised-access-of-social-networking-profile-surges/>
- [16] Attorneys General Multi-State Working Group on Social Networking "Enhancing Child Safety and Online Technologies" Harvard January 2008 <http://cyber.law.harvard.edu/pubrelease/isttf/>
- [17] Dr. Larry Rosen "Social Networking's Good and Bad Impacts on Kids" American Psychological Association August 6, 2011 <http://www.apa.org/news/press/releases/2011/08/social-kids.aspx>
- [18] Hattie Kauffman "Social Networking: An Internet Addiction?" CBS News June 24, 2008 <http://www.cbsnews.com/news/social-networking-an-internet-addiction/>

- [19] LinkedIn. <http://press.linkedin.com/about/> (Accessed 10/06/2014)
- [20] Facebook. <https://www.facebook.com/facebook/info> (Accessed 10/06/2014)
- [21] Twitter. <https://about.twitter.com/company> (Accessed 10/06/2014)
- [22] Foursquare. <https://foursquare.com/about> (Accessed 10/06/2014)
- [23] Instagram. <http://instagram.com/press/> (Accessed 10/06/2014)
- [24] Google. <http://www.google.com/about/company/> (Accessed 10/06/2014)
- [25] Sysomos. <http://www.sysomos.com/> (Accessed 10/06/2014)
- [26] Twitter API webpage. <https://dev.twitter.com/> (Accessed 10/06/2014)
- [27] Foursquare API webpage. <https://developer.foursquare.com/> (Accessed 10/06/2014)
- [28] Instagram API webpage. <http://instagram.com/developer/> (Accessed 10/06/2014)
- [29] Google Maps API webpage. <https://developers.google.com/maps/> (Accessed 10/06/2014)
- [30] Google V8 engine webpage. <https://code.google.com/p/v8/> (Accessed 10/06/2014)
- [31] Maciej Zgadzaj "Benchmarking Node.js - basic performance tests against Apache + PHP"
<http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php>
- [32] Joseph Crow "Continuous Integration for Mobile" LinkedIn engineering webpage
<http://engineering.linkedin.com/testing/continuous-integration-mobile>
- [33] Tornado website <http://www.tornadoweb.org/en/stable/> (Accessed 10/06/2014)
- [34] Spring MVC website <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html> (Accessed 10/06/2014)
- [35] Play framework website <http://www.playframework.com/> (Accessed 10/06/2014)
- [36] Wikipedia "Relational databases" page http://en.wikipedia.org/wiki/Relational_database
(Accessed 10/06/2014)

- [37] Wikipedia "NoSQL" page <http://en.wikipedia.org/wiki/NoSQL> (Accessed 10/06/2014)
- [38] Denis Nelubin and Ben Engber "Ultra-High Performance NoSQL Benchmarking" Aerospike website. <http://www.aerospike.com/wp-content/uploads/2013/01/Ultra-High-Performance-NoSQL-Benchmarking.pdf>
- [39] Couchbase documentation webpage <http://docs.couchbase.com/> (Accessed 10/06/2014)
- [40] ElasticSearch documentation webpage <http://www.ElasticSearch.org/guide/> (Accessed 10/06/2014)
- [41] Kibana webpage <http://www.ElasticSearch.org/guide/en/kibana/current/index.html> (Accessed 10/06/2014)
- [42] Marvel webpage <http://www.ElasticSearch.org/overview/marvel/> (Accessed 10/06/2014)
- [43] MongoDB documentation webpage <http://docs.mongodb.org/manual/> (Accessed 10/06/2014)
- [44] Cassandra webpage <http://cassandra.apache.org/> (Accessed 10/06/2014)
- [45] Redis documentation webpage <http://redis.io/documentation> (Accessed 10/06/2014)
- [46] Neo4j webpage <http://www.neo4j.org/> (Accessed 10/06/2014)
- [47] HBase webpage. <http://hbase.apache.org/> (Accessed 10/06/2014)
- [48] Leaflet webpage. <http://leafletjs.com/> (Accessed 10/06/2014)
- [49] jVectorMap webpage. <http://jvectormap.com/> (Accessed 10/06/2014)
- [50] OpenStreetMaps API. http://wiki.openstreetmap.org/wiki/API_v0.6 (Accessed 10/06/2014)
- [51] Amy Beth Warriner, Victor Kuperman, Marc Brysbaert. "Norms of valence, arousal, and dominance for 13,915 English lemmas" McMaster University, Canada & Ghent University, Belgium. December 2013. http://crr.ugent.be/papers/Warriner_et_al_affective_ratings.pdf

- [52] ISO 3166-2 Wikipedia webpage. http://en.wikipedia.org/wiki/ISO_3166-2 (Accessed 10/06/2014)
- [53] Elasticsearch Indices API <http://www.ElasticSearch.org/guide/en/ElasticSearch/reference/current/indices.html> (Accessed 10/06/2014)
- [54] Elasticsearch Index API http://www.ElasticSearch.org/guide/en/ElasticSearch/reference/current/docs-index_.html (Accessed 10/06/2014)
- [55] Elasticsearch Search API <http://www.ElasticSearch.org/guide/en/ElasticSearch/reference/current/search.html> (Accessed 10/06/2014)
- [56] Elasticsearch Update API <http://www.ElasticSearch.org/guide/en/ElasticSearch/reference/current/docs-update.html> (Accessed 10/06/2014)
- [57] Elasticsearch Delete API <http://www.ElasticSearch.org/guide/en/ElasticSearch/reference/current/docs-delete.html> (Accessed 10/06/2014)
- [58] Extreme programming methodology. http://en.wikipedia.org/wiki/Extreme_programming (Accessed 10/06/2014)
- [59] William Costolo "Miley Cyrus Fans Trying to Beat 'Happy' for MMVA" <http://guardianlv.com/2014/06/miley-cyrus-fans-trying-to-beat-happy-for-mmva/> June 14, 2014 (Accessed 15/06/2014)
- [60] Ethan Sacks "Miley Cyrus sings 'FU' while carrying Selena Gomez cardboard cutout on stage" <http://www.nydailynews.com/entertainment/music-arts/miley-caught-video-singing-fu-selena-cutout-article-1.1824253> June 14, 2014 (Accessed 15/06/2014)
- [61] Moment.js library website. <http://momentjs.com/> (Accessed 13/06/2014)
- [62] Elasticsearch Javascript official client website. <http://www.ElasticSearch.org/guide/en/ElasticSearch/client-api/current/index.html> (Accessed 13/06/2014)
- [63] Socket.io library webpage. <http://socket.io/> (Accessed 13/06/2014)
- [64] Geolib library webpage. <https://github.com/manuelbieh/Geolib> (Accessed 13/06/2014)

- [65] Natural library webpage. <https://github.com/NaturalNode/natural> (Accessed 13/06/2014)
- [66] Request library webpage. <https://github.com/mikeal/request> (Accessed 13/06/2014)
- [67] Underscore library webpage. <http://underscorejs.org/> (Accessed 13/06/2014)
- [68] Underscore.string library webpage. <http://epeli.github.io/underscore.string/> (Accessed 13/06/2014)
- [69] FileSystem library webpage. <http://nodejs.org/api/fs.html> (Accessed 13/06/2014)
- [70] Fast-csv library webpage. <http://c2fo.github.io/fast-csv/index.html> (Accessed 13/06/2014)
- [71] Chancejs library webpage. <http://chancejs.com/> (Accessed 13/06/2014)
- [72] Expressjs library webpage. <http://expressjs.com/> (Accessed 13/06/2014)
- [73] Mustache library webpage. <http://mustache.github.io/> (Accessed 13/06/2014)
- [74] Bootstrap library webpage. <http://getbootstrap.com/> (Accessed 13/06/2014)
- [75] Bootstrap select library webpage. <http://silviomoreto.github.io/bootstrap-select/> (Accessed 13/06/2014)
- [76] jQuery library webpage. <http://jquery.com/> (Accessed 13/06/2014)
- [77] The 11th International Conference on Modeling Decisions for Artificial Intelligence <http://www.mdai.cat/mdai2014/> (Accessed 13/06/2014)

Glossary

ACID Atomicity Consistency Isolation Durability (ACID) is a set of properties that guarantee that database transactions are processed reliably.. 39

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API.. 13, 22, 28, 29, 32, 33, 35, 41, 47–49, 51–55, 60, 63–66

BSON Binary JSON (BSON) is a computer data interchange format used mainly as a data storage and network transfer format in the MongoDB database. It is a binary form for representing simple data structures and associative arrays (called objects or documents in MongoDB). 41

CQL Casandra Query Language (CQL) is a domain-specific language used by the NoSQL database Cassandra. 42

CRUD Create, Read, Update and Delete (CRUD) are the four basic functions of persistent storage. 48, 64

DSL A domain-specific language (DSL) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains, and lacks specialized features for a particular domain first. 41, 49

GNU GNU general public license is a free, copyleft license for software and other kinds of works. 18

- HDD** A hard disk drive is a data storage device used for storing and retrieving digital information using rapidly rotating disks (platters) coated with magnetic material. 45, 50
- HDFS** Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage over Apache Hadoop systems.. 44
- HTML** HyperText Markup Language (HTML) is the standard markup language used to create web pages. 54
- HTTP** The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext. 15, 22, 33, 35, 39, 47, 53, 59
- I/O** Input/Output (I/O) is the communication between an information processing system (such as a computer) and the outside world, possibly a human or another information processing system. 39
- IP** Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internet-working, and essentially establishes the Internet. 47
- JMS** Java Message Service (JMS) is a Java Message Oriented Middleware API for sending messages between two or more clients. 38
- JSON** JavaScript Object Notation (JSON) is a lightweight data-interchange format, easy for humans to read and write, and easy for machines to parse and generate. 33, 35, 41, 48, 49
- MVC** Model View Controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. 38, 45

- NoSQL** No Structured Query Language (NoSQL) database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases. 23, 42, 45
- OS** Operative System (OS) is software that manages computer hardware resources and provides common services for computer programs. 18, 99
- RAID** Redundant Array of Independent Disks (RAID) is a data storage virtualization technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy or performance improvement. 45
- REST** Representational State Transfer (REST) is a way to create, read, update or delete information on a server using simple HTTP calls. 33, 35, 47, 55
- SAAS** Software as a Service (SAAS) is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted on the cloud by independent software vendors or application service providers. 17
- SNA** Social Network Analysis (SNA) is the analysis of social networks. 30
- SQL** Structured Query Language (SQL) is a special-purpose programming language designed for managing data held in a relational database management system. 42, 50
- SSD** A solid-state drive (SSD) is a data storage device using integrated circuit assemblies as memory to store data persistently. 45
- SVG** Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. 63
- URI** Uniform Resource Identifier (URI) is a string of characters used to identify a name of a web resource. 33
- URL** Uniform Resource Locator (URL) is a specific character string that constitutes a reference to a resource. 35, 47, 52

Appendix A

System classes

In this annex we describe all classes used in the different modules used in the system.

A.1 Data Access Layer

A.1.1 Load

Contains all methods necessary to load data from ElasticSearch indexes.

Attributes

This class has no attributes.

Methods

- loadQueries: Takes a callback function as parameter. Loads all entries from the "queries" index, responsible of storing the metadata of all queries. When finished, calls the callback function.

A.1.2 Save

Contains all methods necessary to store new ElasticSearch indexes and modify or add data to those indexes.

Attributes

This class has no attributes.

Methods

- newQuery: Takes a stream and a callback function as parameters. Starts a new query by storing the object data (stream parameters) into the index "queries" of the database. When finished, calls the callback function with the ElasticSearch response.
- newTweet: Takes a stream, a tweet and a callback function as parameters. Enrich the tweet calling all the proper enrichers, and then stores the tweet in the database index assigned to the stream. When finished, calls the callback function with the tweet as response.
- endQuery: Take a stream id and a callback function as parameters. Updates the query entry in the "queries" index for that stream, adding an end time. When finished, calls the callback function with the ElasticSearch response.
- deleteQueries: Takes an array of stream id and a callback as parameters. For each stream, deletes its index in the database and its entry in the "queries" index. When finished, calls the callback function with the ElasticSearch response.

A.1.3 External libraries

The external libraries used in the data access layer are:

- Moment: Manages time and date formatting [61].
- ElasticSearch: Official ElasticSearch client for Node.js [62].

A.2 Business Logic Layer

A.2.1 Twitter Loader

Contains all methods necessary to load data from Twitter REST API.

Attributes

This class has no attributes.

Methods

- loadLanguages: Takes a callback function as parameter. Obtains the supported language list from Twitter, calling the URL "/help/languages.json". When finished, calls the callback function with the language JSON as response.

A.2.2 Container Controller:

Class responsible to manage the different containers, including store their metadata, create them, destroy them, and assigning each container the proper authentication keys to connect to the Twitter API.

Attributes

- containers -> Array of Container: Contains an array of all containers available to the system. The information attached to each container (Name and Twitter Authentication keys) is read from the configuration file.
- busyContainers -> Array of Integer: Array with the ID of all containers that are currently busy streaming.
- freeContainers -> Array of Integer: Array with the ID of all available containers.

Methods

- get: Takes a container ID as parameter. Returns the container with that ID.
- getAll: Takes no parameters. Returns all containers.
- isAvailable: Takes no parameters. Returns true if there is at least one free container, and false otherwise.

- create: Takes a stream object as parameter. Launches a new streaming process, stores the PID of the generated process plus the stream data into the container information, and moves the selected container from "freeContainers" to "busyContainers". If there is no free container, returns an error, otherwise, returns a container ID.
- destroy: Takes a container ID as parameter. If the container is busy, destroy the process attached to that container, empties the container stream data information, and moves the container ID from "busyContainers" to "freeContainers". Otherwise, returns an error.

A.2.3 Container

Class that represents a unique, single container. Container is understood as a single process unit that runs independently from the main process and connects with the given credentials and filters to initiate and manage a single data stream from the Twitter streaming API.

Attributes

- id: The container id.
- name: The container name.
- keys: The Twitter authentication keys assigned to this container.
- process: A ChildProcess object containing the child process that is running the stream.
- stream: An object containing the information of the stream currently running in the process.

Methods

- getId: Takes no parameters. Returns the container id.
- setId: Takes a string as parameter. Sets the container id.
- getName: Takes no parameters. Returns the container name.
- setName: Takes a string as parameter. Sets the container name.

- getKeys: Takes no parameters. Returns the container keys.
- setKeys: Takes an object with keys as parameter. Sets the container keys.
- getProcess: Takes no parameters. Returns the container process.
- setProcess: Takes a ChildProcess object as parameter. Sets the container process.
- getStream: Takes no parameters. Returns the container stream.
- setStream: Takes an object with stream information as parameter. Sets the container stream.

A.2.4 Stream Controller

Class responsible to initiate and manage a stream. The stream controller has no power to stop a stream, this is done by the Container killing the process.

Attributes

- stream: Contains the Stream object managed by the controller.

Methods

- get: Takes no parameters. Return an object containing all fields of the managed stream.
- set: Takes an object with the stream filters, a container number, and a callback function. Initiates the stream with the given filters, using the credentials assigned to the given container number. Whenever a new tweet arrives, it is processed and stored, and the callback function is called with the processed tweet as parameter.

A.2.5 Stream

Attributes

- connection: Contains an object with the Twitter opened stream.
- id: The stream id.

- track: Filtered words of the stream.
- languages: Filtered languages of the stream.
- locations: Filtered locations of the stream.
- start: Start time of the stream.
- end: Programmed end time of the stream.

Methods

- getConnection: Takes no parameters. Returns the stream connection.
- setConnection: Takes a connection object as parameter. Sets the stream connection.
- destroyConnection: Takes no parameters. Stops the current opened stream connection, and sets the connection attribute to null.
- getId: Takes no parameters. Returns the stream id.
- setId: Takes a string as parameter. Sets the stream id.
- getTrack: Takes no parameters. Returns the stream track.
- setTrack: Takes a string as parameter. Sets the stream track.
- getLanguages: Takes no parameters. Returns the stream languages.
- setLanguages: Takes an object with keys as parameter. Sets the stream languages.
- getLocations: Takes no parameters. Returns the stream locations.
- setLocations: Takes a ChildProcess object as parameter. Sets the stream locations.
- getStart: Takes no parameters. Returns the stream start.
- setStart: Takes an object with stream information as parameter. Sets the stream start.
- getEnd: Takes no parameters. Returns the stream end.
- setEnd: Takes an object with stream information as parameter. Sets the stream end.

A.2.6 Device enricher

Obtains the device from an HTML Twitter source code. A more detailed explanation can be found in section 4.2.2.1.

Attributes

This class has no attributes.

Methods

- add: Takes the field "source" of a tweet as parameter. Parses that field to obtain the proper device used to send the tweet. Returns the parsed device and, in case it is not found, the source field itself.

A.2.7 Geo enricher

Gives extended geolocation functions, correcting geolocation errors and adding word plus geolocation filtering. A more detailed explanation can be found in section 4.2.2.2.

Attributes

This class has no attributes.

Methods

- add: Takes the tweet coordinates, the tweet message, an array of bounding boxes, and an array of words as parameters. Returns true if the tweet coordinates belong to at least one of the bounding boxes from the array, and if the tweet message contains at least one of the words from the filter (only if the word filter was set). Otherwise, it returns false.

A.2.8 Spain enricher

Classify a Spanish coordinate by autonomous community. A more detailed explanation can be found in section 4.2.2.2.

Attributes

This class has no attributes.

Methods

- add: Takes the tweet coordinates as parameter. Returns the ISO code of the autonomous community which the coordinate belongs to, or undefined if the tweet does not belong to Spain.

A.2.9 Gender enricher

Obtains the gender from a name. A more detailed explanation can be found in section 4.2.2.4.

Attributes

This class has no attributes.

Methods

- add: Takes a username and a callback function as parameters. Returns the gender, analysed from the input username, and the filtered name, without accents and strange characters.

A.2.10 Stopwords enricher

Filters defined stopwords from a text. A more detailed explanation can be found in section 4.2.2.5.

Attributes

This class has no attributes.

Methods

- add: Takes an array of string and a language string as parameters. Returns an array of strings with the stopwords filtered if the language is supported, and undefined otherwise.

A.2.11 Stem enricher

Applies a stemming process to a text. A more detailed explanation can be found in section 4.2.2.6.

Attributes

This class has no attributes.

Methods

- add: Takes an array of string and a language string as parameters. Returns the Lancaster stemming of the words if the language is Spanish or English, and undefined otherwise.

A.2.12 Sentiment enricher

Analyses the sentiment of a text. A more detailed explanation can be found in section 4.2.2.7.

Attributes

This class has no attributes.

Methods

- add: Takes an array of string and a language string as parameters. Returns an object with the sentiment analysis if the language is Spanish or English, and undefined otherwise. The return object has, as fields, the sentiment score, the arousal, an array of string with the words used to determinate the sentiment, a count field with the amount of words found, and a sentiment string based on the previous score.

A.2.13 External libraries

The external libraries used in the business logic layer are:

- Socket.io: Library for real-time bidirectional event-based communication [63].
- Moment: Manages time and date formatting [61].
- Geolib: Library to provide basic geospatial operations like distance calculation or conversion of decimal coordinates to sexagesimal [64].
- Natural: Language analysis library, including tokenizing, stemming, classification, phonetics, tf-idf, WordNet and string similarity [65].
- Request: Simplified HTTP client [66].
- Underscore: Library that provides useful functional programming helpers without extending any built-in objects [67].
- Underscore.string: Library for comfortable manipulation with strings [68].
- Fs: Native Nodejs library to manipulate files [69].
- Fast-csv: CSV parser and writer library [70].
- Chance: Random string and number generator library [71].
- Express: Flexible web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications [72].

A.3 Display Interface Layer

A.3.1 DateManager

Manages the date filter in the query interface.

Attributes

- date: Contains the current end date of the filter
- enabled: Contains a boolean with the status of the date filter, depending if the user has set an end time or not.

Methods

- getDate: Takes no parameters. Return the current end date if enabled is set to true, and undefined otherwise.
- getDateString: Takes no parameters. Return the current end date in a string format if enabled is set to true, and undefined otherwise.
- setDate: Takes a date as parameter. Sets the current end date.
- enable: Takes no parameters. Set enabled to true.
- disable: Takes no parameters. Set enabled to false.

A.3.2 LanguageManager

Manages the language filter in the query interface.

Attributes

- info: Object containing server response to the language loading.
- selectedLanguages: Array of strings with the ISO codes of the current selected languages.

Methods

- getLanguages: Takes no parameters. Return a string containing ISO codes of all currently selected languages.
- getFullLanguages: Takes no parameters. Return a string containing the English name of all currently selected languages.

- getLanguageName: Takes an ISO code as parameter. Returns the English name of the language.
- getLanguageCode: Takes an English language name as parameter. Returns the ISO code of that language.
- addLanguage: Takes an English language name as parameter. Adds the ISO code of the language to the selectedLanguages array.
- removeLanguage: Takes an English language name as parameter. Removes the ISO code of the language from the selectedLanguages array.

A.3.3 MapManager

Manages all maps during the interface, including the filter map in the query interface and the popup maps in host and history interfaces.

Attributes

- map: A google.maps.map class from Google Maps API that contains the current map controls.
- areas: An array of google.drawingManager.boundingBox from Google Maps API, each containing a bounding box and controls from Google Maps API bounding boxes, which includes colour, opacity, stroke weight and colour, if the area is clickable, editable and draggable, and the zIndex.
- drawingManager: a google.drawingManager class from Google Maps API that contains the drawing tool controls for the query interface.

Methods

- getBoundingBox: Takes an area ID as parameter. Returns the bounding box of the specified area.

- getBoundingBoxes: Takes no parameters. Returns the bounding box of all currently drawn areas.
- drawRectangle: Takes a bounding box and a hexadecimal colour as parameter. Draws the bounding box with the specified colour as the area colour.
- drawRectangles: Takes an array of bounding boxes and a hexadecimal colour as parameters. Draws all bounding boxes in the map using the colour parameter as the area colour.
- centerMap: Takes no parameters. Center the map, moving and zooming, to fit all areas in the view.
- removeArea: Takes an area ID as parameter. Remove the area with the parameter ID from the map.
- removeAreas: Takes no parameters. Remove all areas drawn in the map.

A.3.4 External libraries

The external libraries used in the display interface layer are:

- Socket.io: Library for real-time bidirectional event-based communication [63].
- Moment: Manages time and date formatting [61].
- Mustache: Logic-less HTML templates5 [73].
- Underscore: Library that provides useful functional programming helpers without extending any built-in objects [67].
- Google Maps API: Offers maps and location-based tools [29].
- Bootstrap: Mobile first front-end framework for faster and easier web development with responsiveness [74].
- Bootstrap-select: Custom select box for bootstrap using button dropdown [75].
- Jquery: Feature-rich JavaScript library extending the language with new functions [76].

Appendix B

Dashboard

The following annex contains captures from the user interface.

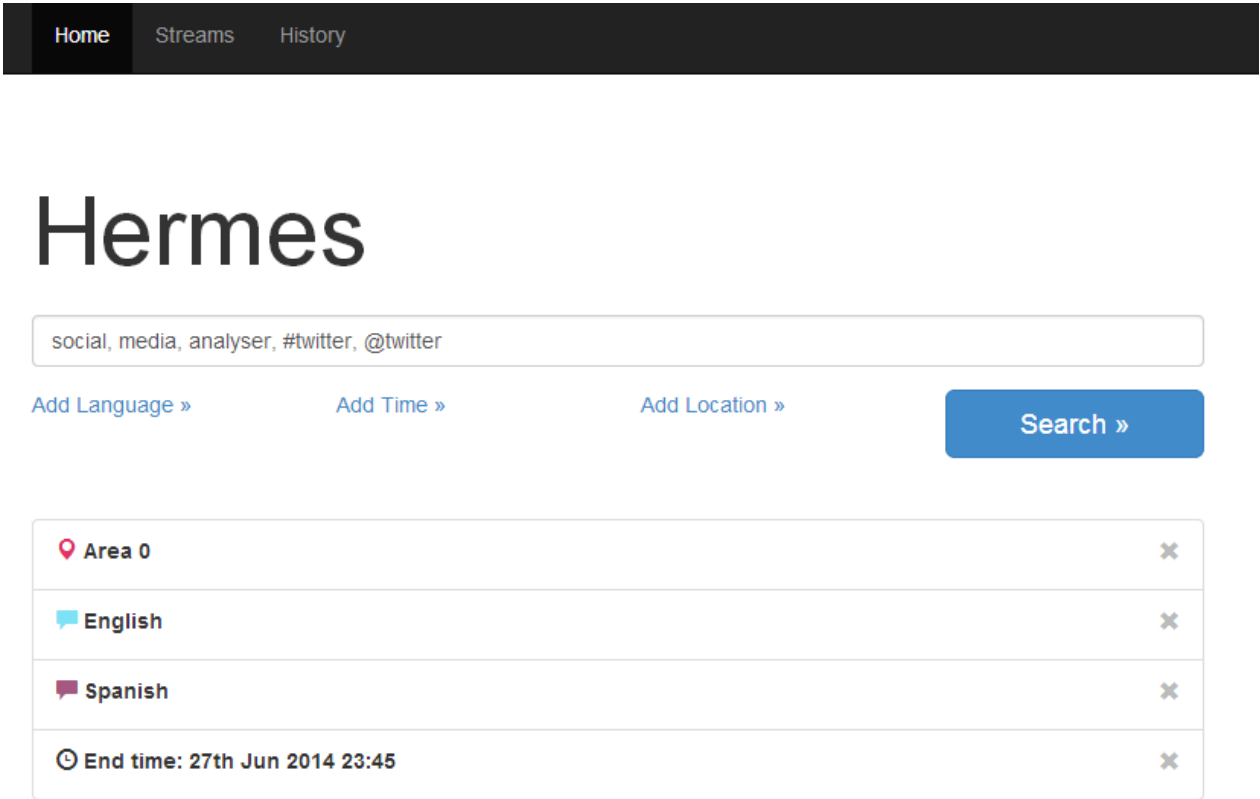


Figure B.1: Query interface

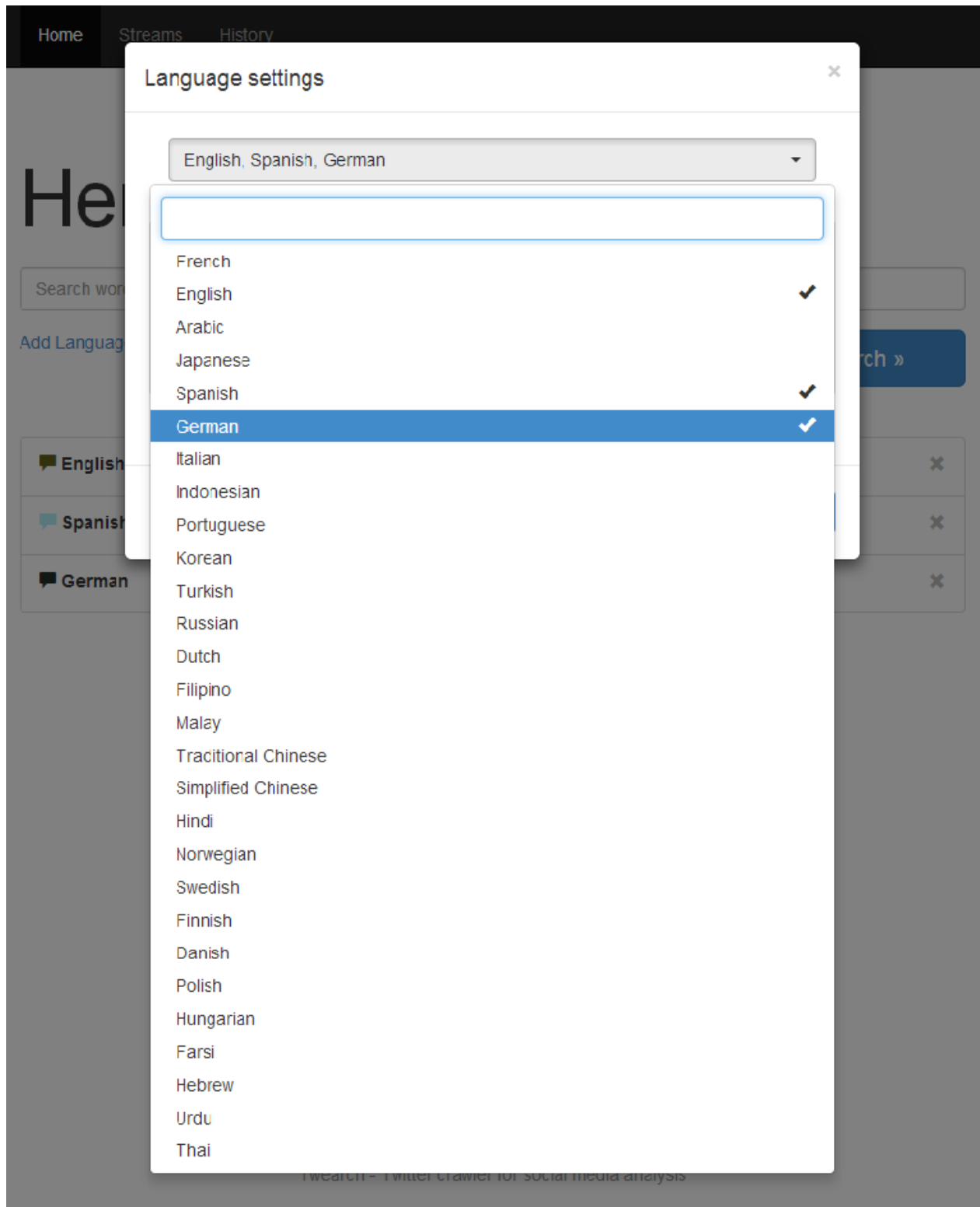


Figure B.2: Query interface, language view

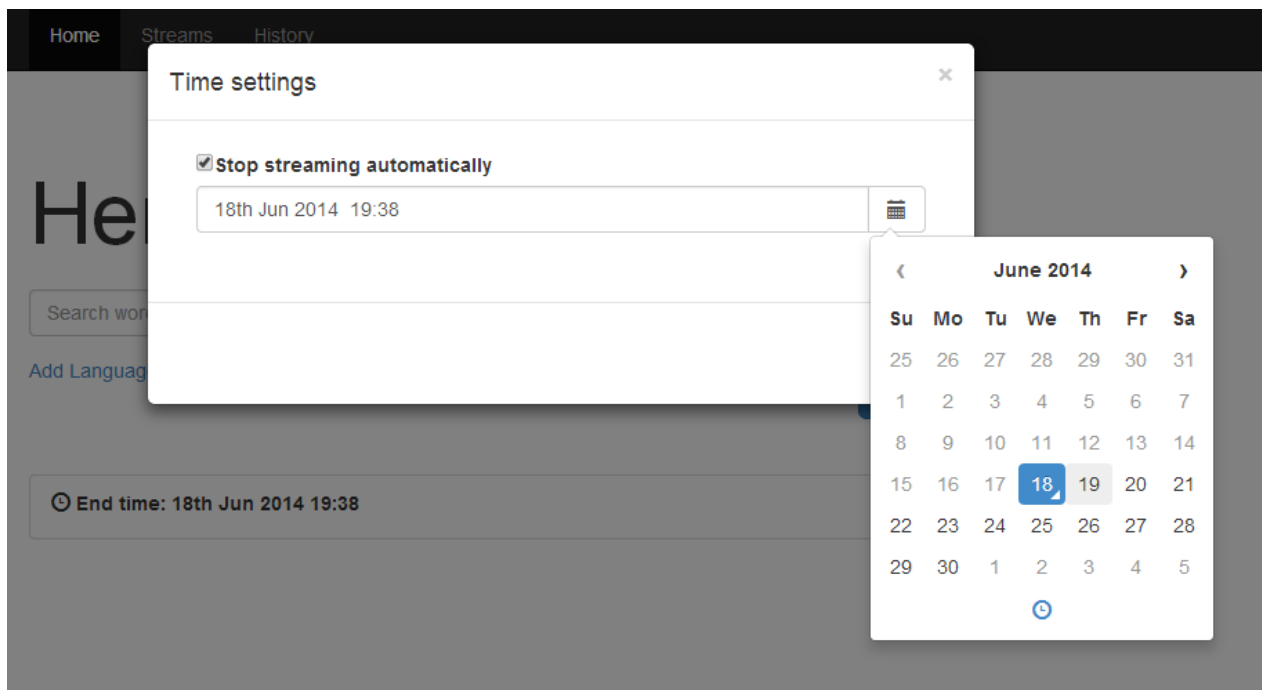


Figure B.3: Query interface, time view

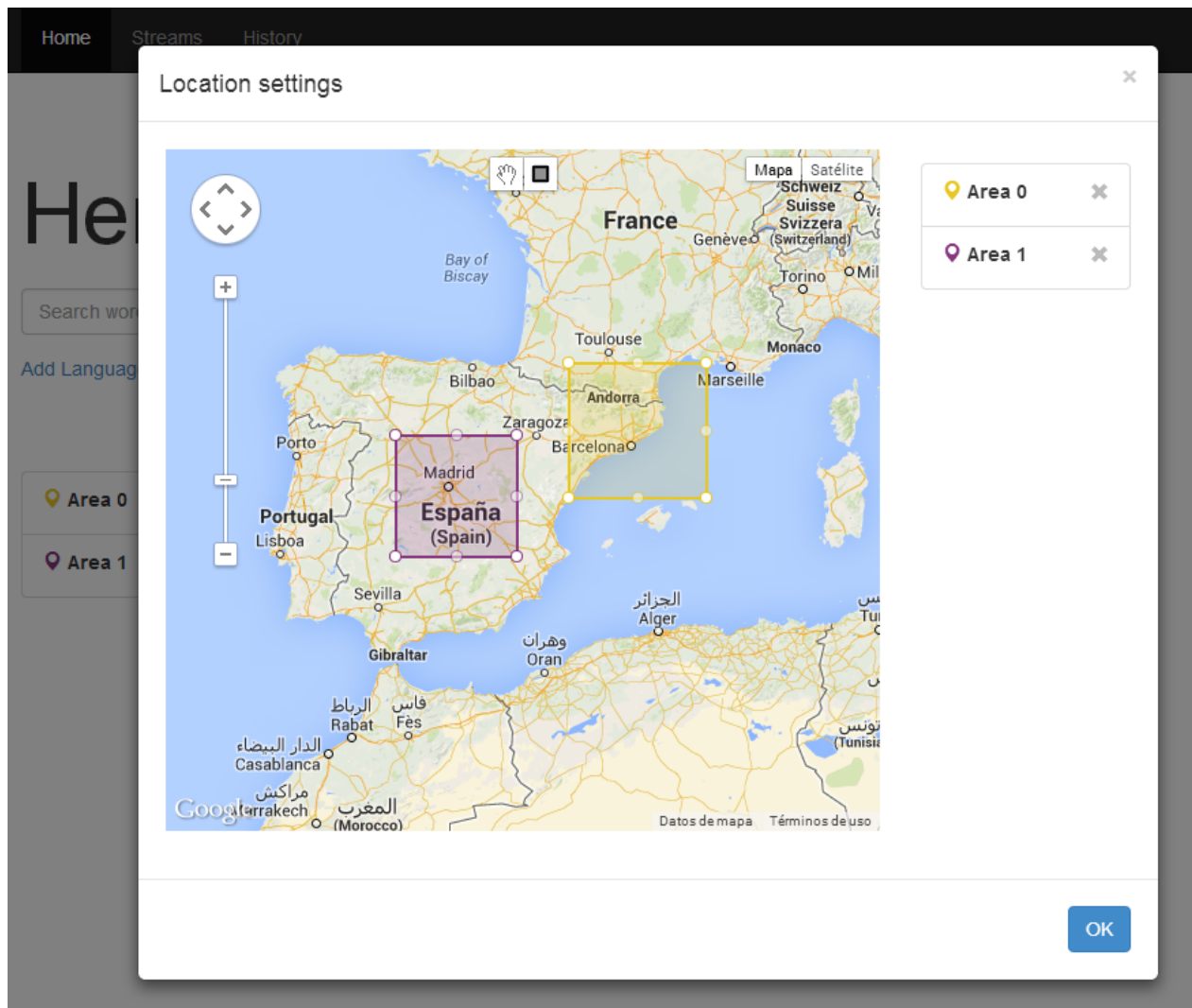
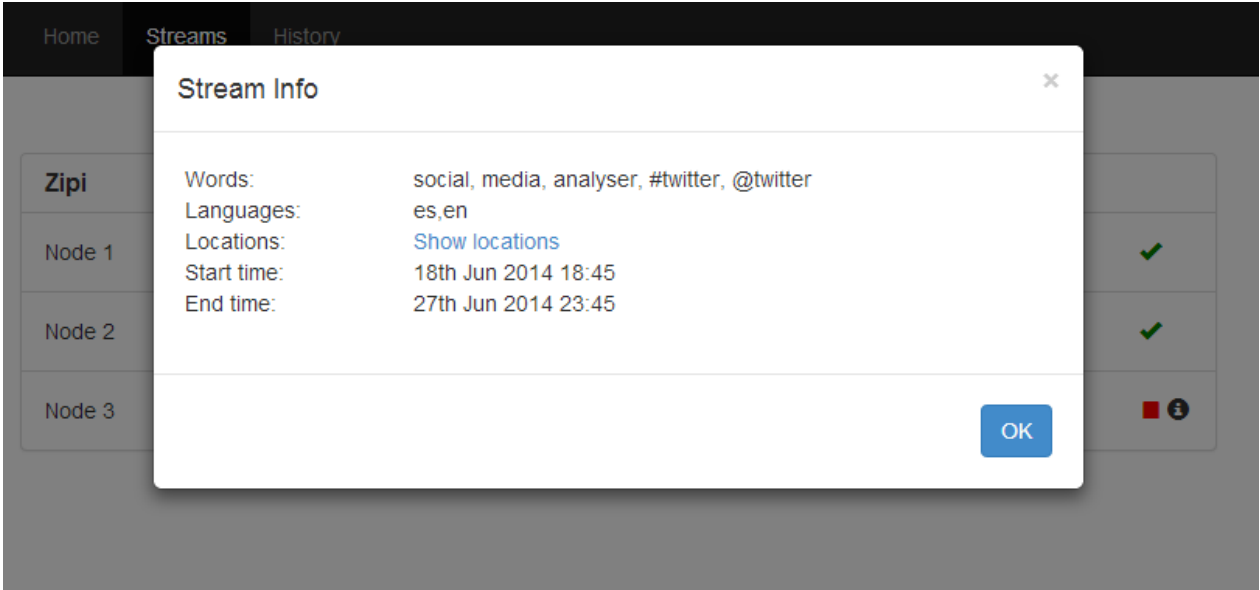


Figure B.4: Query interface, location view



Zipi	
Node 1	✓
Node 2	✓
Node 3	■ ⓘ

Figure B.5: Host interface



Stream Info [X]

Words: social, media, analyser, #twitter, @twitter
Languages: es,en
Locations: [Show locations](#)
Start time: 18th Jun 2014 18:45
End time: 27th Jun 2014 23:45

[OK]

Figure B.6: Host interface, slot info view

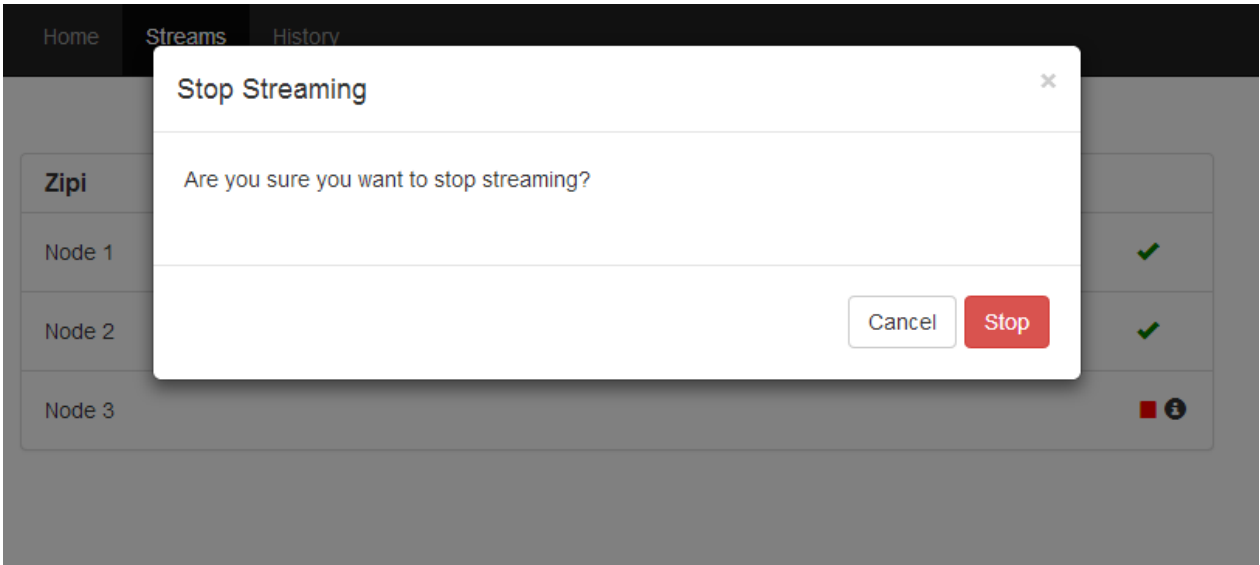


Figure B.7: Host interface, stop stream modal

ID	Track	Languages	Locations	Start Date	End Date
1398275136937		All languages	Show locations	23rd Apr 2014 19:45	23rd Apr 2014 23:38
1398275256265		All languages	Show locations	23rd Apr 2014 19:47	23rd Apr 2014 23:38
1400051846709	@awscloud	All languages	All locations	14th May 2014 09:17	16th May 2014 11:13
1400231658920		All languages	Show locations	16th May 2014 11:14	Currently running
1400321765757		All languages	Show locations	17th May 2014 12:16	Currently running
1400887534601	champions, liga campeones, atletico madrid, real madrid	es	Show locations	24th May 2014 01:25	26th May 2014 21:41
1400887735432	PP, partido popular, PSOE, izquierda unida, IU, elecciones, votar, voto	es	Show locations	24th May 2014 01:28	26th May 2014 21:41
1400932293163		es	Show locations	24th May 2014 13:51	Currently running
1401700230842		All languages	Show locations	2nd Jun 2014 11:10	Currently running
1401716624714		All languages	Show locations	2nd Jun 2014 15:43	Currently running

Search ...

[Display info »](#)
[Delete queries](#)

[← Previous](#)
[1](#)
[2](#)
[Next →](#)

Figure B.8: History interface

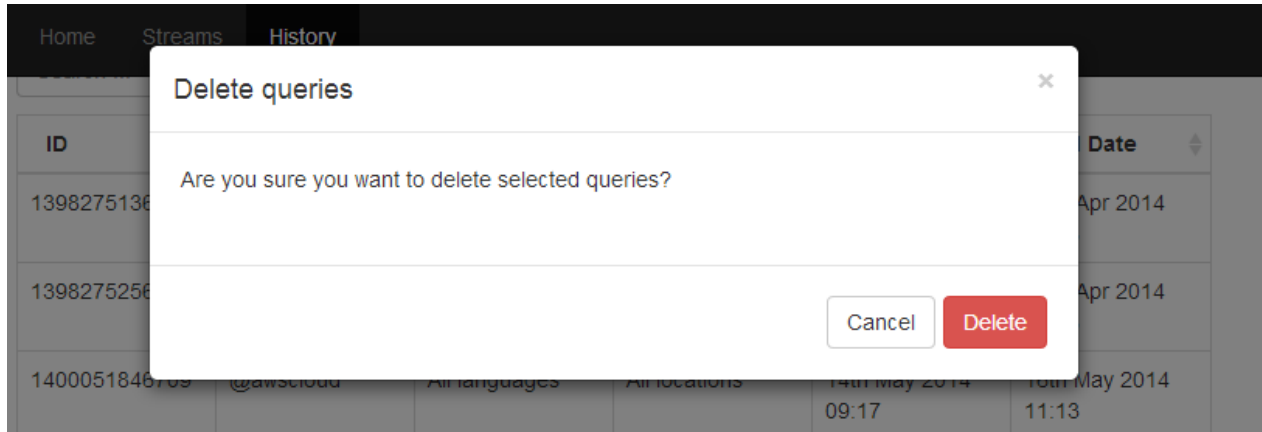


Figure B.9: History interface, delete query modal



Figure B.10: Results interface, part 1

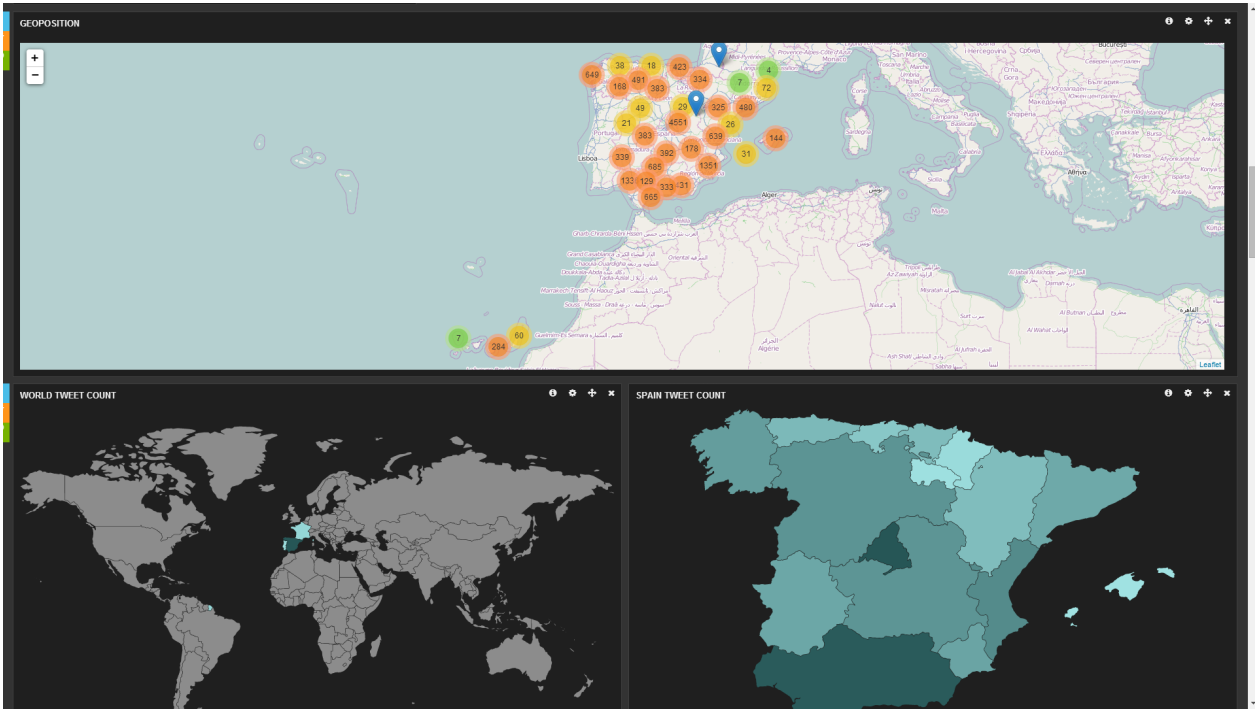


Figure B.11: Results interface, part 2

Appendix C

Paper

In this annex can be read the paper about this project that was accepted in the 11th International Conference on Modeling Decisions for Artificial Intelligence (MDAI) celebrated from 29th to 31st October 2014 in Tokyo, Japan [77].

Towards the Cloudification of the Social Networks Analytics

Daniel Cea, Jordi Nin, Rubén Tous, Jordi Torres, and Eduard Ayguadé

Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (BarcelonaTech)
Barcelona, Catalonia, Spain
dcea,nin,rtous,torres,eduard@ac.upc.edu

Abstract. In the last years, with the increase of the available data from social networks and the rise of big data technologies, social data has emerged as one of the most profitable market for companies to increase their benefits. Besides, social computation scientists see such data as a vast ocean of information to study modern human societies. Nowadays, enterprises and researchers are developing their own mining tools in house, or they are outsourcing their social media mining needs to specialised companies with its consequent economical cost. In this paper, we present the first cloud computing service to facilitate the deployment of social media analytics applications to allow data practitioners to use social mining tools as a service. The main advantage of this service is the possibility to run different queries at the same time and combine their results in real time. Additionally, we also introduce *twearch*, a prototype to develop twitter mining algorithms as services in the cloud.

Keywords: Social Mining, Green Computing, Cloud Computing, Big Data Analytics, Twitter Mining, Stream Processing

1 Introduction

A growing proportion of human activities, such as social interactions, job relationships, entertainment, collaborative working, shopping, and in general, gathering information, are now mediated by social networks and web services. Such digitally mediated human behaviours can easily be recorded and analysed, fuelling the emergence of (1) computational social science, (2) new services such as tuned search engines or social recommender systems, and (3) targeted online marketing. Due to this, public and private sector actors expect to use big data to aggregate all of this data, extract information (and knowledge) from it, and identify value to citizens, clients and consumers [15, 4, 14].

According to one research study [9] from the University of Maryland's Robert H. Smith School of Business, this growing allows Facebook, twitter and other social media sites to create between 182,000 and 235,000 jobs in US and has added between \$12.19 billion and \$15.71 billion in wages and salaries. A similar study funded by Facebook finds that in Europe, Facebook added a similar number of jobs (approximately 232,000). All these business opportunities have been

only possible thanks to the possibility to mine social media insights through development APIs.

While many success stories proliferate, in the private sector, social media analytics have found a *killer application* on the Market Research arena. Market research analyses information about customers and target markets to study the market size, market need and competition. The analysis of social media data provides an unprecedented opportunity to understand how customers behave and why, becoming a key component of business strategy. Platforms for social media analytics are proliferating rapidly nowadays (Twitonomy [20], SumAll [17], TwitSprout [21], etc.), with a recent trend towards specialising on market research and brand strength analysis (Brandchats [1], Brandwatch [2], etc.). However, most of these platforms are private initiatives and the ones that are freely available present important hardware restrictions and, therefore, limitations to perform complex queries.

In order to overcome these aforementioned limitations, the contributions of this paper are: an elastic cloud computing service to facilitate the deployment of social media analytics applications together a graphic framework to automatically display some mining results. The proposed Platform as a Service (PaaS) provides the bottom subsystems of the solution stack required by companies (underlying source API access, storage and retrieval) and provisions the necessary hosting capabilities in a scalable and elastic manner without duplicating computer resources. With our architecture, one client can query different social networks with different queries at the same time and display, in real time, the aggregated mining results.

The rest of this paper is organised as follows. Firstly, in Section 2 a brief overview of the related work is introduced. Secondly, a complete description of the proposed architecture and software stack is depicted in Section 3. Later, in Section 4 a real example for the twitter social network is shown. Finally, the paper finishes with some conclusions and future work.

2 Related Work

The major part of social mining platforms covers the entire lifecycle of data analysis, from data gathering to reporting and visualisation. In order to do so, they spend a lot of effort on *reinventing-the-wheel* at the initial stages (data gathering, storage and querying) shortening their resources for the analysis and visualisation stages [11], in which reside their competitive advantage.

For instance in [22], authors propose one architecture to extract and cluster all the tweets of a city. However, if two cities must be monitored, the architecture must be completely duplicate, posing serious scalability problems. Other platform is Datasift [5], where users pay for executing queries over a large set of data sources, but without any option to execute part of the analysis in-house to save money.

In [8], authors describes SONDY, a tool for analysis of trends and dynamics in online social data using twitter. SONDY is written in java, therefore, it is

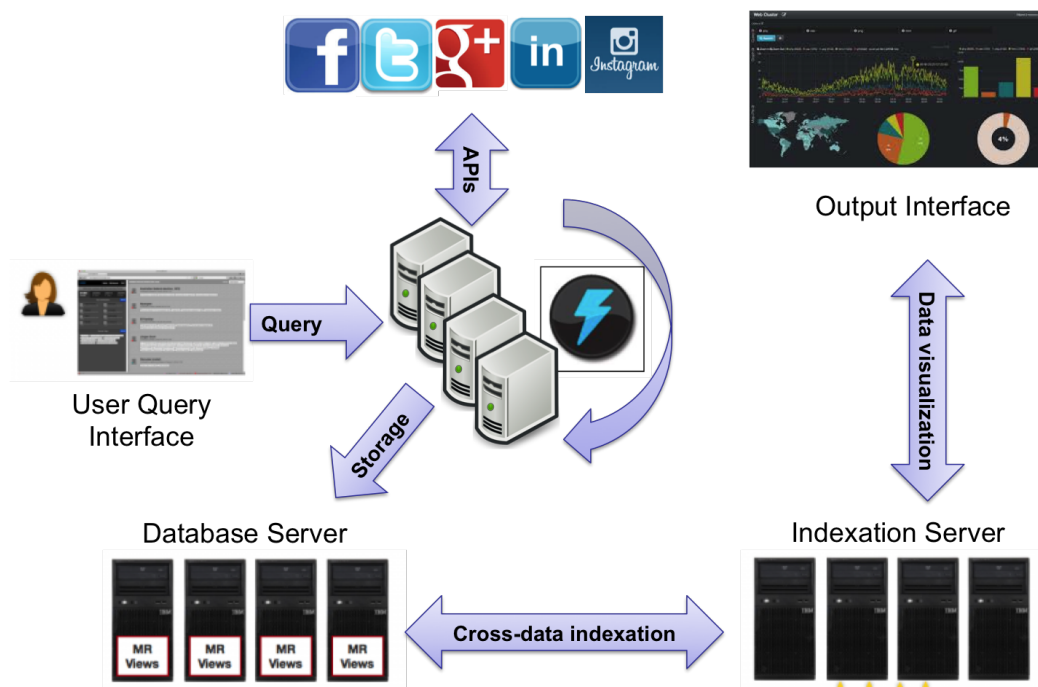


Fig. 1. General Architecture

difficult to make it scale. Besides, it does not allow users to aggregate data coming from several social networks. Finally, SocialSTROM [23] is a cloud-based hub which facilitates the acquisition, storage and analysis of live data from social media feeds (Twitter, Facebook, RSS sources and blogs), as SONDY, it is a java application and it also present scalability limitations.

3 Framework Definition

In this section we detail the main components of our framework, their goals, the selected software and how they interact.

The proposed social mining architecture is composed by 5 independent but interacting elements, as shown in Figure 1, each of them described below. The architecture receives as input the query parameters to be "analysed". Other parameters are optional, such as the possible data post-processing techniques, data enrichment methods, data sources crossings, etc. The architecture outputs some graphical statistics, in parallel data is stored into a NoSQL database for further analysis if needed.

The main components are:

- **User Query Interface:** the user query interface consist on a responsive web application where the user can set the query and also obtain some

feedback about the execution, such as total number of retrieved elements, query parameters, execution time, server usage, etc. Apart from that, user is also able to recover the queries executed in the past for further analysing.

- **Processing Cluster:** the cluster consists on several distributed nodes that are in charge of retrieving the public information, as well as, to post-process it if needed. For example, in the running example depicted in Section 4, it is responsible to connect to the Twitter Streaming API, manage the persistent HTTP connection, and filter out the results.
- **Database Server:** the database server is composed of several nodes where data is distributed along different nodes, offering a flexible and scalable data model.
- **Indexation Server:** indexation server creates a set of data indexes to increase the performance of the database server. It maintains a reverse index for each retrieved word. It automatically updates such indexes when a new data element arrives.
- **Output Interface:** for the output interface, where the results of the query are displayed, we use the graphic framework Kibana, which offers a responsive and friendly display solution for our analytics.

3.1 Software Stack

In this section we introduce all the software components, from the virtualisation platform to the data visualisation tools that we have used to develop our architecture.

First of all, to easily create and destroy virtual machines we execute Open Nebula [13] in the cluster of the Computer Architecture Department of the Technical University of Catalonia ¹. Open Nebula is an open-source project delivering a simple and flexible solution to build and manage enterprise clouds and virtualized data centers. Combining existing virtualization technologies with features for multi-tenancy, automatic provision and elasticity, open Nebula aims to provide a *open, flexible, extensible, and comprehensive* management layer to automate and orchestrate the operation of enterprise clouds. We have used Open Nebula to deploy the required virtual machines for our architecture. Virtualisation makes our system elastic with regards the amount of data captured in any moment.

For the Database server, we use Couchbase [3] as the distributed data repository. Couchbase is an open-source, distributed, NoSQL document-driven database optimised for interactive applications serving many concurrent users; creating, storing, retrieving, aggregating, manipulating and presenting the data. Couchbase borns from to the fusion of Membase and CouchOne projects in January 2012. The current release offers features including *JSON document store, indexing and querying, incremental MapReduce and cross datacenter replication*.

For the Indexation server, the natural decision is to use Elasticsearch [7], the native indexation software for Couchbase. Elasticsearch is an open-source,

¹ <http://www.ac.upc.edu/serveis-tic/altas-prestaciones>

distributed, real-time search and analytic engine built specifically to run on NoSQL document-driven databases. Documents are stored as JSON, and all the fields are automatically indexed and usable in a single query. Elasticsearch principal features include: scalability, high availability, multi-tenancy, full text search, conflict management between different versions, and a restful API using JSON over HTTP. Elasticsearch permits us to create a large amount of queries over a set of different data streams stored in Couchbase.

Finally, for the output interface, we use Kibana [10] because it is based on javascript and bootstrap and it is fully compatible with any browser. Kibana is an open-source, scalable, real-time visualisation tool natively integrated with Elasticsearch. Its main goal is to display the data stored with Elasticsearch in an elegant graphical manner. Kibana key features include time-based comparisons, easy creation of graphical data representations (plots, charts and maps), flexible, editable and responsive web interface, and a powerful search syntax. In our system, we have adapted our mining methods to display their results in this visualisation framework.

4 Twearch: A running example

In this section, we describe a proof of concept application to show the feasibility of our architecture. To do that, we have implemented a twitter listener and some basic queries on the top of Kibana. Twearch offers a simple query interface for twitter able to filter in real time the twitter data stream, by means of any combination of keywords, locations, language, etc. Besides, Twearch also offers to data miners, an output interface to create graphics using javascript.

4.1 Twitter Connection

Twitter, is an online social network born in March 2006 [6] that enables users to send and read “tweets”, which are text messages limited to 140 characters, also allows data programmers to access in real time to perform any kind of text-mining technique, such as clustering, TF-IDF, etc. However, it is impossible for a single computer to capture and process in real time the complete twitter information flow. For example, in 2012, Twitter had 500 million users registered posting over 350 million Tweets per day [12, 16, 18, 19].

Twitter offers two different APIs for developers:

- **REST API:** used to retrieve past *tweets* based in different filters. There are different resources depending on the data to be retrieved: Accounts, friendships, geolocations, statuses, users, etc. Depending on the resource, the number of queries per account is limited from 15 to 450 per *rate limit window* (by March 2014, 15 minutes long). Each query response contains 100 *tweets*.
- **Streaming API:** used to retrieve *tweets* in real time. Not all the *tweets* are sent, but only a 5%, so it’s mostly used for analysis purposes. There are 3

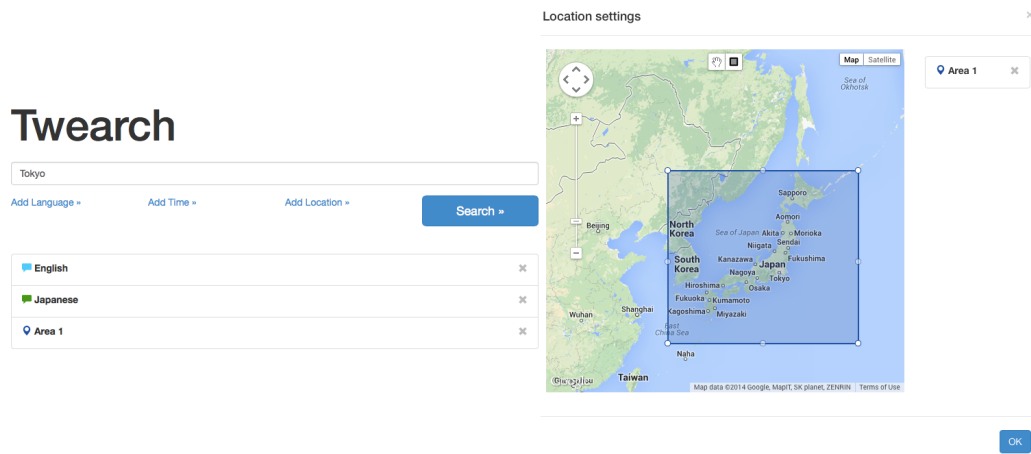


Fig. 2. Input interface query

different resources, depending on the target: Public streams, User streams, and Site streams. The stream API is limited to 1 data stream per account.

Since our framework is designed to manage stream data, we connected Twearch to the Twitter streaming API.

4.2 Query System

As a running example for this paper, we want to retrieve all tweets containing the word "Tokyo" during a week. The input interface, as it is shown in Figure 2, asks Twitter to filter all incoming tweets and sends us only those containing the string "Tokyo". Apart from keywords, using the twearch interface, one user can filter the twitter stream using hashtags, mentions, languages and spatial coordinates.

Note that, similar input interfaces can be created for others social networks, such as foursquares, instagram o facebook. Doing this, it is easy to cross information coming from different social networks to enrich data analytics without too much effort and allowing the data miner to recycle the listeners for future analytics.

The input interface sends the query to one node of the processing cluster. Once the query is received in one of the nodes, it opens a permanent HTTP connection with the Twitter Streaming API, and the data stream will begin to flow back containing the requested tweets. Then, data stream will be processed, enriched, stored into the Couchbase database and indexed by Elasticsearch server.

When the output interface (Kibana) is opened, each panel will send a predefined query through the Elasticsearch API REST, using HTTP GET requests, to ask for the concrete fields and filters needed for that panel, and then populate the data graphically using different types of panel. Here, it is important to

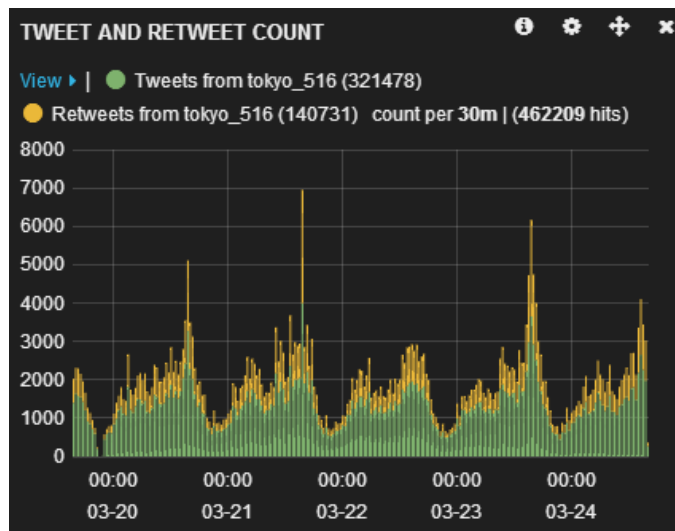


Fig. 3. Number of tweets and retweets per 30 minutes

highlight that more than one query (even from more than one social network) can be sent to Kibana, therefore query results can be aggregated and recycled each time a new query is executed.

As an example, we have created the Tweets/Retweets panel (Figure 3), where there are 2 different queries inside the request: A count of all tweets from the stream, and another count from all tweets from the stream whose 'retweeted' field is true.

We have also implemented other two panels (Figures 4.(a) and(b)) both related to the Tweet text: one counting the top 10 hashtags (a), and another for user mentions (b). This information is obtained after processing the text field of each tweet stored, and separating the strings depending whether they are hashtags or user mentions. From these two panels we can easily extract some knowledge, such as the high social impact of two teenagers groups; *egirls* and *nuest* from Japan and Korea respectively.

Finally, in order to exploit a different type of the information stored in a tweet, we have also the geo-positioned in a world map all the geo-located tweets. To do that, we use the Open Street Maps library, as it is shown in Figure 5.

4.3 Query Refining

Additionally, it is possible to refine the results and focus them on a concrete time period using the Kibana output interface. Data miners can filter the documents by their timestamp and study in detail concrete time periods. For instance, Figure 3 depicts some tweets and re-tweets peaks from 22:00 to 23:00 during some days of March. If we focus on these tweets, as it is shown in Figure 6.(a), it is possible to observe that the mentioned users changes (see Figure 6.(b)).

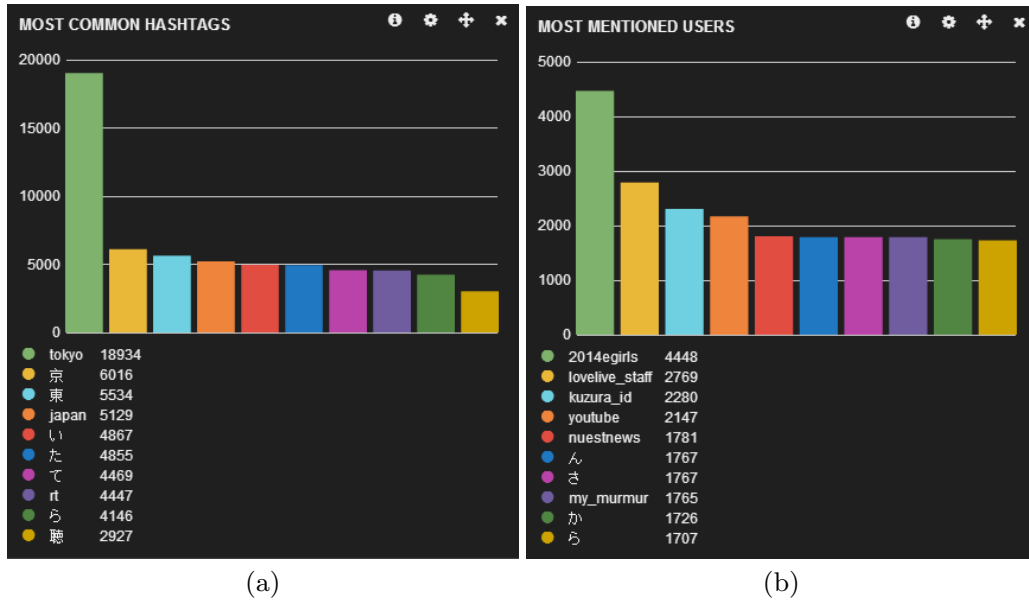


Fig. 4. Counting query of the most common words, hashtags and mentioned users among tweets

Concretely, the most mentioned user is lovelive instead of 2014egirls. Lovelive is a very popular Japanese anime serial, and the first chapters of its second season were broadcasted during these concretes time periods. So, we were able to detect new trends easily using twearch.

Finally, observing Figure 5, it is possible to see that most of the tweets are located in Tokyo, zooming at the center of the city, we discover that a big cluster of tweets is located at the Shibuya station, one of the most visited parts of the city. Therefore, using this map it is possible to automatically detect huge twitter users concentrations for a given query.

4.4 Platform Performance

Apart from the social information extracted from Twitter, another topic of interest is to analyse the performance of the system taking measurements of the main hardware components during the information retrieval process.

To achieve that, the components of the architecture send reports, every 5 seconds, about the amount of CPU and Memory being consumed, Specifically:

- **Processing node:** It is the node of the processing cluster responsible to host the input query interface and display the mining results. Besides, it is also responsible to send the streaming job start signal to the proper crawling node.
- **Crawling node:** Node responsible to connect to Twitter, receive the streaming data, refine the query and store the stream in the database.

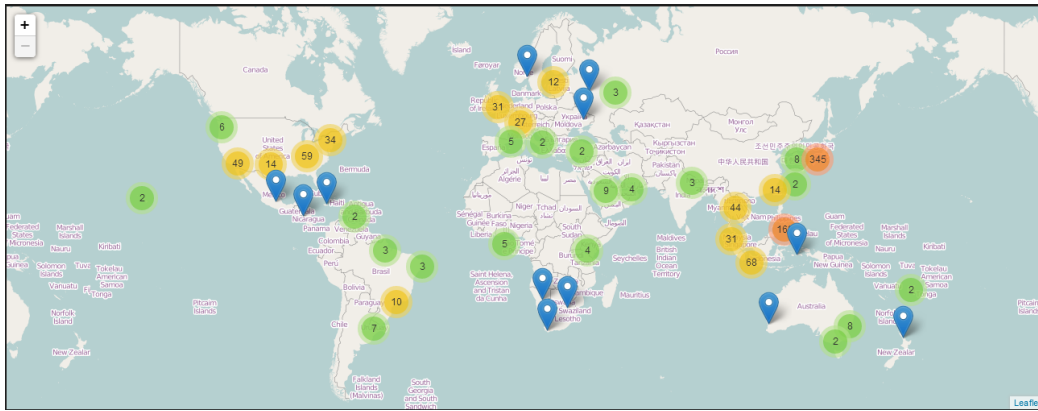


Fig. 5. Geopositioned Tweets in OpenStreetMaps

- **CouchBase + ElasticSearch:** Node hosting the Couchbase database and the ElasticSearch indexer.

Those results are stored in CouchBase and displayed in a Kibana histogram, which shows results of the average consumption every 10 minutes during the part of the streaming process (in our case, the last 2 days).

As we illustrate in Figure 8, on the one hand, the amount of CPU used by the Processing and Crawling nodes is almost negligible, with a mean value less than 2% of the CPU with some punctual peaks that are always less than 5%. On the other hand, the Couchbase and ElasticSearch are CPU-consuming processes that average a 15% of CPU consumption with peaks over the 25%

About the memory, as we can observe in Figure 9, all processes keep a constant amount of consumed memory: Couchbase and Elasticsearch use around 550MB of RAM memory, the crawling node is around 100MB, and finally the processing node around 90MB.

The CPU performance results show that it is possible to consolidate several queries into a single virtual machine, reducing the required number of virtual machines needed to perform complex queries where lot of information has to be retrieved. For the data management system (database and indexes) the real bottleneck is the RAM memory not the CPU. To overcome this drawback, more than one virtual machines (or physical servers) can be deployed to exploit the scalability of Couchbase and Elasticsearch.

5 Conclusions

In this paper we have described all the components of an elastic and scalable framework for social mining in a cloud infrastructure, in our case Open Nebula. We have described the database management system, the query language and the visualisation tool. Finally as a proof of concept, we have described how to collect

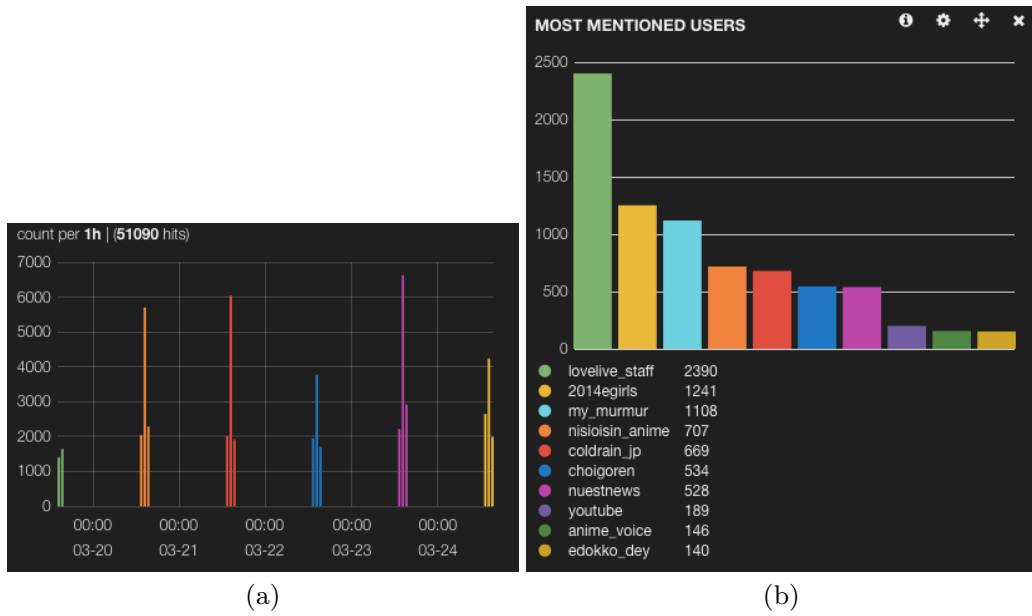


Fig. 6. Lovelive new season advertising.

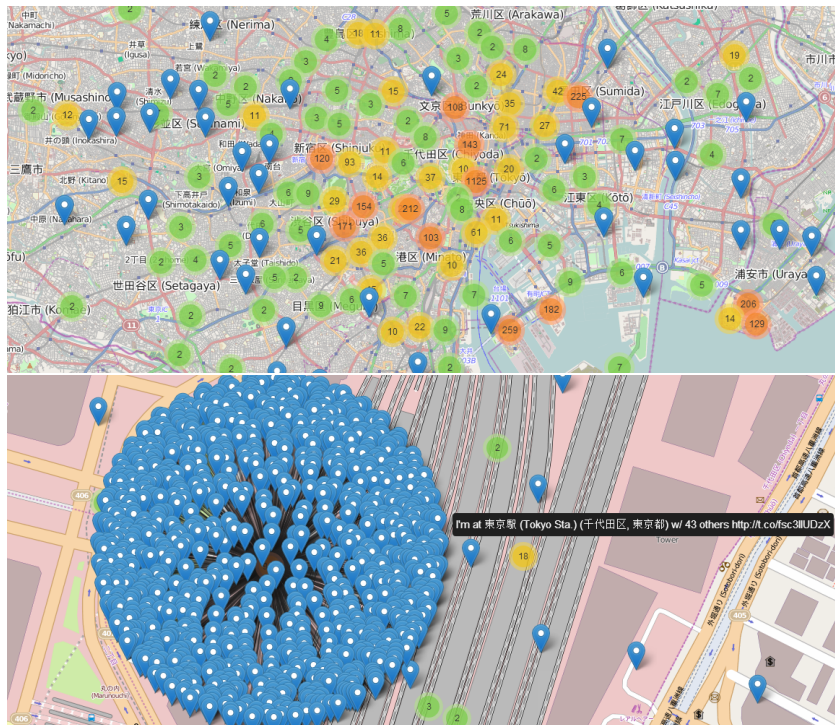


Fig. 7. Shibuya station tweet cluster.

data from twitter and we have displayed some data analytics and performance metrics for a given query using the proposed query system.

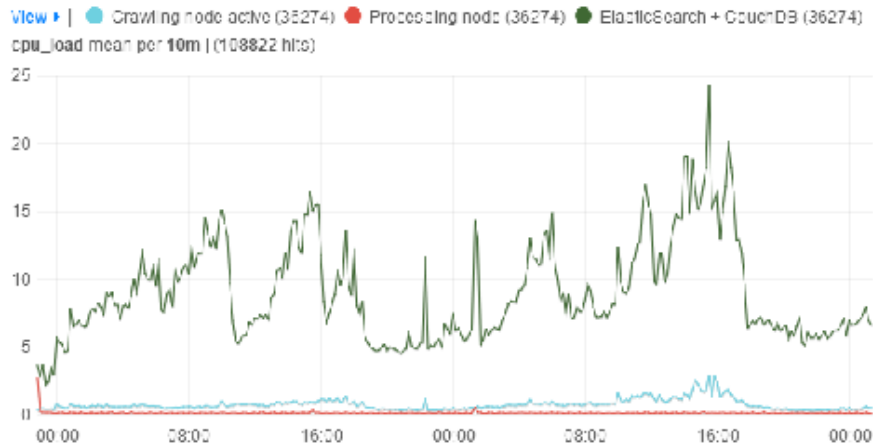


Fig. 8. CPU consumption, average and maximum, of all the system components during a 2 day streaming.

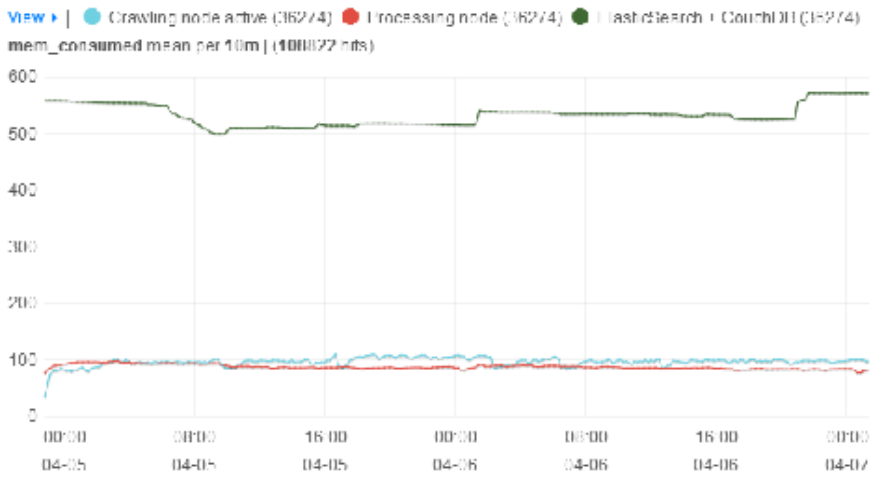


Fig. 9. Memory consumption, average and maximum, of all the system components during a 2 day streaming.

In the near future, we plan to add more functionalities to our platform as for instance, natural language processing methods, automatic data enrichment by means of data crossing. Finally, we would like to define a decision support system to help designers of appliances to optimise resource allocation in a semi-supervised way.

Acknowledgments. This work is partially supported by the Ministry of Science and Technology of Spain under contract TIN2012-34557, by the BSC-CNS Severo Ochoa program (SEV-2011-00067). Besides, authors would like to thank CA

Technologies for funding this research through a collaboration agreement with Universitat Politècnica de Catalunya

References

1. Brandchats. <http://www.brandchats.com> (Accessed March 20, 2014).
2. Brandwatch. <http://www.brandwatch.com> (Accessed March 20, 2014).
3. Martin C. Brown. *Getting Started with Couchbase Server - Extreme Scalability at Your Fingertips*. O'Reilly, 2012.
4. Junghoon Chae, Dennis Thom, Yun Jang, SungYe Kim, Thomas Ertl, and David S Ebert. Public behavior response analysis in disaster events utilizing visual analytics of microblog data. *Computers & Graphics*, 38:51–60, 2014.
5. Datasift. <http://datasift.com> (Accessed March 20, 2014).
6. J. Dorsey. just setting up my twttr. *Twitter. Oldest Tweet ever from Twitter founder.*, 2006.
7. Elasticsearch official website. <http://www.elasticsearch.org/> (Accessed March 20, 2014).
8. C. Adrien Guille, H. Hacid, and D. Zighed. Sindy: An open source platform for social dynamics mining and analysis. In *ACM SIGMOD*, 2013.
9. Il-Horn Hann, Siva Viswanathan, and Byungwan Koh. *The Facebook App Economy*. Center for Digital Innovation, University of Maryland, 2011.
10. Kibana website. <http://www.elasticsearch.org/overview/kibana/> (Accessed March 20, 2014).
11. Michael Lieberman. Visualizing big data: Social network analysis. In *Digital Research Conference*, 2014.
12. I. Lunden. Twitter passed 500m users in june 2012, 140m of them in us; jakarta biggest tweeting“ city. *techcrunch*, 2012.
13. Opennebula key features and functionality. <http://www.opennebula.org> (Accessed March 20, 2014).
14. Taylor Shelton, Ate Poorthuis, Mark Graham, and Matthew Zook. Mapping the data shadows of hurricane sandy: Uncovering the sociospatial dimensions of 'big data'. *Shelton, T., Poorthuis, A., Graham, M., and Zook, M*, 2014.
15. Mark A Stoové and Alisa E Pedrana. Making the most of a brave new world: Opportunities and considerations for using twitter as a public health monitoring tool. *Preventive Medicine*, 2014.
16. D. Strachan. Twitter: How to set up your account. *The Daily Telegraph*, 2009.
17. Sumall. <http://sumall.com> (Accessed March 20, 2014).
18. Twitter Search Team. The engineering behind twitter’s new search experience. Technical report, Twitter Engineering Blog, 2011.
19. Twitter Search Team. Twitter turns six. Technical report, Twitter Engineering Blog, 2012.
20. Twitonomy. <http://www.twitonomy.com> (Accessed March 20, 2014).
21. Twitsprout. <http://twitsprout.com> (Accessed March 20, 2014).
22. D. Villatoro, J. Serna, V. Rodríguez, and M. Torrent-Moreno. The tweetbeat of the city: Microblogging used for discovering behavioural patterns during the mwc2012. In *Citizen in Sensor Networks*, volume 7685 of *Lecture Notes on Artificial Intelligence*, pages 43–56. Springer-Verlag, 2012.
23. R. Wood, I. Zheludev, and P. Treleaven. Mining social data with ucl’s socialstorm platform. Technical report, University College of London (UCL), 2011.