2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Model of Computer Architecture for Online Social Networks Flexible Data Analysis

The case of Twitter data

Romain Giovanetti
CRISTAL Lab
University of Lille
Lille, France
romain.giovanetti@univ-lille1.fr

Luigi Lancieri
CRISTAL Lab
University of Lille
Lille, France
luigi.lancieri@univ-lille1.fr

***Abstract*—Since several years, there is an increasing interest for new services based on the analysis of data coming from online social networks. Such services can, for example, provide the e-reputation of a product or a company, detect new trends in a commercial, social or political context, etc. The huge quantity of data is an opportunity in term of representativeness but is also difficult to manage. Within Twitter, for example, it appears that the huge stream of data is, most of the time, incompatible with a flexible analysis unless to have high computer resources. The only practical solution is often to observe in a static way a limited portion of a phenomenon in a limited time slot. This paper is devoted to the study of necessary conditions to provide an equilibrium between the computer architecture complexity and the analysis flexibility.**

Keywords—flexible data analysis; online social network; twitter; computer architecture; distributed database; platform

I. INTRODUCTION

The OSNs (i.e., Online Social Networks) are becoming more and more central in our society. But, at first, these tools were seen as a curiosity by researchers who considered them as unimportant or playful. Actually, from their point of view, the gain provided by OSNs compared with existing Internet services such as email or blogs, for instance, was not clear.

Over time, the evolution of the socio technical context has changed this perception and has resulted in a situation where researches related to OSNs have, in recent years, increased dramatically, involving many disciplines. In fact, the increasing number of users and the democratization of mobile technologies make that OSNs' data are a true mirror of society. Indeed, interactions in text mode, image or video show the tastes or the concerns of individuals as well as their social, political or economic preferences. This is particularly visible in the case of Twitter, which produces an, almost immediate echo of all important events from anywhere in the world. This reactivity appears to be higher than that of the press even if it is less structured. For instance, Paul S. Earle and his colleagues show that, by using Twitter's data, about 75% of the earthquake detections occur within 2 minutes of the origin time [29] (see also [9]). This is considerably faster than seismograph detections in poorly instrumented regions of the world. Many other studies in the domain of public health, security, economy, etc. show that OSNs can be useful, not only to be rapidly informed but also in order to provide many details for the diagnosis of events, and forecast their evolution. It is also interesting to see that the use of OSNs by end users has also evolved in parallel. A 2015 report from Pew Research Center tells us that clear majorities of Twitter users (63%) and Facebook users (63%) now say each platform serves as a source for news about events and issues outside the realm of friends and family [30].

These examples explain why the potential of OSNs started to mobilize researchers and businesses, who have seek to exploit the wealth of "big data" linked to interactions between users. The challenge is as much financial as scientific. On one hand financial, because OSNs' data open the way to enhancement of existing functionalities at lower cost. This is the case of opinion polls that, in an ideal case, could be done without face-to-face interviews, providing a fastest and most accurate image of collective opinions. On the other hand scientifically, because these data pave the way to forms of observation of human behavior unthinkable up to now. All this potential is based on the analysis of data related to interpersonal acts of communication (tweets, Facebook status updates, short videos on Vine, etc.), as well as the context in which these communications are carried out (time, localization, popularity, etc.).

Beyond the difficulty inherent in the data analysis, which involves skills in social science domain, the features of the underlying computers systems also pose many problems. They include in particular the constraints of data collection, automatic processing of linguistic data, and Human–computer interaction (HCI). These constraints arise because the vast amount of information generated in real time requires a powerful and adaptable computer architecture. Josh James summarizes the situation stating that "data never sleeps", for instance, during one minute 347 222 tweets are posted, 4 166 667 users like something on Facebook, and 1 736 111 photos are published on Instagram [1]. In the particular case of Twitter, the processing of linguistic data is complicated by the low length of status (140 characters) and the high level of noise (bad spelling, etc.). The constraints of the user interface appear by the need to synthesize the information extracted from the analysis. In other words, under what form should be represented the information (charts, alerts, etc.) in order to give an accurate meaning with respect to the observer's needs? We will see that to highlight an event or a trend, it is necessary to take into account, in combination, all of these constraints.

Most researchers chose the option of collecting tweets corresponding to the topic they wish to analyze, at the time

IEEE/ACM ASONAM 2016, August 18-21, 2016, San Francisco, CA, USA
978-1-5090-2846-7/16/$31.00 © 2016 IEEE

they need them. For example, when one wants to know the public opinion on the brand iPhone, he collects and analyzes tweets containing the keyword *iphone*. In this case, the analyst must be patient until enough data have been collected. For a trend analysis, it may take several days. If, once this analysis obtained, one realizes that it is useful to compare the previous results to other smartphone brands, it will be necessary to wait again that some new data have been collected. As we can see, the problem with such approach is that it inhibits the analyst spontaneity. Indeed, he must carefully anticipate requests about data that he could need. If during the analysis process, some data miss, he could be discouraged to have to wait several more days to get them.

The alternative approach is to broaden the thematic field of data to collect. If we take the previous example, one will not only collect tweets containing the keyword *iphone* but also those with related words (e.g., *phone*, *smart-phone*, *mobile*, etc.). It is possible thanks to associative networks (semantic linked words). This approach has many advantages especially in terms of comfort since the probability that useful data will miss is reduced. But, from the other side, the problem is that this option introduces constraints in terms of data load, given the multiplication of tweets to collect. In such case, the computers architecture should be carefully designed in order to sustain this load, not only during the tweets reception but also toward the user interface. Indeed, due to the huge amount of data, the fact to provide to users a fast and interactive interface with comparative graphs can also become a challenge. As an example, a database architecture can be from 2 to 10 times faster than another [20].

The following generic architecture shows the aggregation of different elementary features such as collecting, storing, and analyzing tweets, and managing the user interface. Beyond the sake of combining complementary functions, this generic architecture also allows to highlight systemic features that emerge from this association. For example, the platform could have a degree of autonomy by collecting tweets that have not been explicitly requested, but have a chronological or semantic link with the initial request. This can be a community link if we consider this type of platform can be used by a group of analysts. In this case, one of the analysts can anticipate queries on topics that might interest his colleagues later. This type of functionality is only possible by collecting tweets over a long period of time and away from the traditional punctual strategy.

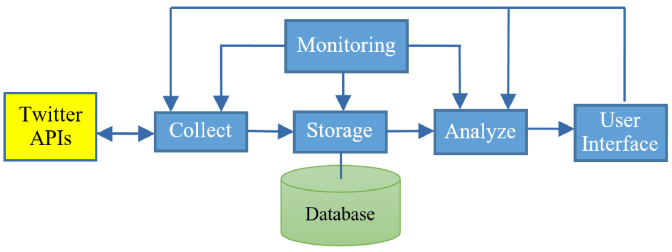

Fig. 1. A generic architecture for Twitter Data Analytics

This continuity requires the platform to be the subject of a special attention. The architecture and its performance must be over-sized. Furthermore, its operation must be monitored in order to quickly identify and fix problems that would prevent the collecting of data.

In the following, we describe the main functional blocks of this generic platform even if they are not easy to isolate in the literature. There is no unified approach in this domain because authors are most of the time concerned by the analyze of data rather than by the problems of their collecting, their storage or the performance of their user interfaces. For example, in many cases, modules developed for recovering Twitter's data have been designed to work with analysis tools (e.g., R, RapidMiner [19], etc.) or by specifically targeting the storage architecture (such as Elasticsearch Twitter River [38]). First, we focus on data collecting from Twitter. We describe the form of these data and the means proposed by the social network to recover them. We then list technical tools for tweets analysis. Next, we present a state of the art related to platforms for collecting and processing Twitter's data. The type of database used and the features of their user interface are summarized in order to highlight their strength and weakness. Later we discuss about user interfaces for data analysis. Finally, we present as a case study the platform we developed for Twitter's data analysis.

II. Collect of Twitter Data

Twitter is a free micro-blogging platform, which exists since March 21th, 2006 [2]. For now, users publish messages (tweets) of a maximal length of 140 characters, but regularly, the company questions this artificial limit. Actually, Twitter will soften the 140 characters rule soon (may 2016) [39]. This debate is not neutral because short messages determine a particular use marked by spontaneity and simplicity. The popularity of SMS is unwavering on smartphones while more advanced forms of messaging are now available. This shows us that short messages are more than a fad. The tweets can be sent to a private circle of readers but are usually open to the public. They are not editable; they can only be removed. Users can be their authors; they can retweet them (i.e. cite) or add them to their favorites. These actions have been interpreted by some authors from a social point of view. The retweet action, for instance, can be seen as an agreement, a recommended reading, an information sharing, a flattery, a snapshot of an event, a payback or a greater exposure [3].

In terms of data structure, tweets may contain different entities such as a mention of another user (e.g., @userT), a marker of metadata (such as "hashtags" like #subject25) or an URL that can lead to another OSN, a media hosting service, or a website. If an URL targets a media (such as a picture, a video or a live stream video), some extra metadata can indicate its type or its dimensions. Thus, the media is automatically displayed below the tweet as a card [4], saving users to follow its URL in a new tab of their web browser. In addition to these principal data, an appendix of additional metadata is also available. It contains among others the unique identifier of the tweet, a geotag, the language of the tweet (automatically determined by Twitter [5]), and the number of times the tweet was retweeted and favorited [6].

Many solutions are available to collect and analyze these data. Twitter offers APIs (i.e., Application Programming Interfaces) [10] since 2006. The policy of Twitter regarding them is that a minority of consumers has a full access to the tweets (Firehose) and the rest has a limited access (Public APIs). Both of them provide only publicly published tweets.

Getting a permission to access the Firehose is practically impossible and is often the subject of a private monetized contract between Twitter and a big private actor of the social analysis world. In recent months, Twitter has taken back hold on the Firehose and many companies have seen their access cut [36].

The restricted access offers two sets of APIs: REST (i.e., Representational State Transfer) APIs and Streaming APIs. Studies that focus on entities such as hashtags, terms or keywords in tweets, tend to use the REST APIs, while studies that attempt to observe, for instance, longitudinal of movies or politics, use the public Streaming APIs. These APIs are free but require a free Twitter account. The data accessible via the REST APIs are severely limited because Twitter imposes download rate limits divided into 15 minute intervals. Similarly, Streaming APIs provide a limited access to the real-time stream of tweets that represents less than 1% of the total flow. From a technical point of view, Twitter uses OAuth to provide authorized access to its APIs. The REST APIs are based on the client-server model: a connection between Twitter and a consumer is dynamically created for each query. Conversely, Streaming APIs rely on a continuous connection between Twitter and consumers; they are designed to send large volumes of data.

All the scientific works quoted in this paper make use of the free restricted access (REST APIs and Streaming APIs).

III. Twitter Analytics Tools

To show how the initiatives related to the Twitter data analysis are many, let us quote the post of Pam Dyer where he already identified in 2013, 50 tools, mostly online, for analyzing the content of OSNs, among which 20 more specifically dedicated to Twitter [21, 22]. It's interesting to note the high volatility of these websites. Indeed, only 6 of them are still operational today (March 2016). Many of those who are closed invoke a change in the Twitter APIs as the main reason of their fate. Beyond that reason, which could be called functional, it should also be noted that the economic model of these websites is still to be defined. Actually, even in the case of Twitter, which has already a very large community of users, the viability of the business model is still sometimes debated [23]. This explains that, apart from very marginal business initiatives, the development tools and the analytics websites are primarily related to evaluation projects in academia.

In general, the collected tweets must be shaped and processed to bring out elements of knowledge buried in the, sometimes weak, signals and data to reveal trends or alerts. Some of these features may be supported by the existing data mining tools [7]. However, the form of tweets and their associated data (retweets, author's id, etc.) have specific characteristics that imply a mainly linguistic pretreatment, especially important if one wishes to make operational data analysis (automatic, scalable, etc.).

These different features are related by data structures and high level programming languages. In his book Mining the Social Web [11], M. A. Russell explains how to datamine on various OSNs using Python as programming language. In the chapter related to Twitter, he uses in his demonstrations the Python Twitter Tools module [12]. He also addresses the analysis of data in various forms but does not mention the issue of storage. Even without an external database, the language Python allows to export data into text files (such as CSV, XML or JSON). Similarly, the book Twitter Data Analytics written by S. Kumar [19] addresses the questions of the collecting, the processing and the visualization of statistical indicators of Twitter's data. He uses the language Java and associated libraries to perform these treatments.

Beyond the specific application, several developers have designed plugins to adapt the existing data mining tools to the Twitter APIs. For example, "Analytics module for Twitter" allows one to query Twitter directly into Microsoft Excel 2010. One can perform analysis like who are the most active users, which tweets correspond to a given hashtag or which tweets are rather positive or negative [24]. Some authors use the reporting features of Google Analytics to track the activity of OSNs and especially Twitter [25]. These approaches are particularly suited to marketing strategies aimed, for instance, to measure the popularity of a product, an event or a TV show.

To perform more sophisticated analysis, it is more interesting to use specialized tools in data mining and statistical computing. Most of these tools have a connector that can be interfaced with the Twitter APIs. We describe below the open-source and free tools, but these opportunities also exist for commercial products (Matlab, Mathematica, etc.).

MOA (i.e., Massive Online Analysis) is an open source tool specialized in data flow analysis and also allows developing recommendations systems [27]. MOA originated in Weka, a popular data-mining tool (classification, etc.). These two tools offer together great versatility. The MOA Twitter reader module allows in particular adapting these tools to the Twitter context. In addition to the collecting, it also offers to detect changes in real time, such as the identification of terms whose frequency changed. It also allows an analysis in real-time of feelings.

RapidMiner is a popular data analysis tools available since 2006. There are now a free version (on sourceforge.net) and a commercial version ($ 2,000). Recently, RapidMiner studio offered features for analyzing the activity on Twitter, multilingual texts, sentiment, etc. [26] (see also Knime [28]).

On July 24th, 2009, the twitterR module for R makes its first appearance. R is free software for data processing and statistical analysis, which implements the programming language S. This module is a wrapper for high-level dialogues with the Twitter APIs. It simplifies the OAuth authentication and transforms S language requests to HTTP REST requests [13]. Since February 23th, 2014, it is possible to easily record tweets and other information in a relational database management system like RSQLite [14].

The use of data analysis tools reveals their limits in terms of data storage, features and HCI, and can be first exploratory steps in the process of designing a data analysis platform that will offer more features and will better handle huge volumes of data.

IV. Twitter Data Analysis Platforms

The majority of the available scientific literature on the subject reveals that there are many technical solutions to recover data from Twitter and many publications make use of a relational database to store them. Comparing the different architectures that were discussed in the scientific literature is a complicated task because studies present their work with a variable level of clarity and specificity. Nevertheless, we ordered them in two categories according to how they store their data. On one hand, we list platforms that rely on a centralized database, on the other hand, those that make use of a distributed database.

In March 2009, K. Makice publishes the book "Twitter API: Up and Running - Learn How to Build Applications with the Twitter API". He explains how to capture tweets via the Twitter APIs using the language PHP and how to store them in a MySQL relational database [31].

In 2010, R. D.W Perera, S. Anand, K. P. Subbalakshmi and, R. Chandramouli present a software architecture for developing stochastic models to characterize OSNs [37]. They focus on the time intervals between the creation of tweets and the frequency of retweets made by a user of the tweets from another user. To do so, they make use of the Search API from Twitter REST APIs, the languages Python and PHP, and a centralized MySQL database. The collecting of tweets is written in Python and uses the Twython library. Their capture script runs every 5 minutes. In order to determine the location of tweets, they employ a Yahoo web service that turns an address into GPS coordinates. Captured tweets are stored in the MySQL database by extracting their id, their timestamp and the id of their author. A PHP application reads and displays the contents of the database.

In early June 2010, M. Mathioudakis and N. Koudas are co-authors of an article that deals with a two parts application (back-end and front-end), which allows highlighting trends on Twitter when they occur [8]. The back-end part, written in Java, uses the Twitter Streaming APIs to collect data in real time and process them later. It stores the captured tweets, with all their metadata, in a module called index and sends to a bursty keywords detection module a simplified flow that contains only the text part of the tweets with their timestamp. Once the simplified flow is analyzed and trends are detected, a trend analysis module retrieves additional information about detected trends from the previously generated index module. The front-end part, called TwitterMonitor, lets final users view the results.

In 2012, M. Oussalah, F. Bhat, K. Challis, and T. Schnier describe a software architecture that collects tweets sent from a predefined geographical area and over a specified period of time using the Twitter Streaming APIs. It also performs text queries over captured data, and groups them by location [32]. Their architecture uses the Python web framework Django coupled with Apache Lucene. They are linked to a MySQL database in order to have both an efficient indexing powered by Lucene and a relational model in conjunction with the cross platform side of MySQL. The Twitter4J library is employed to collect tweets. To prevent the risk of interruption during the collecting, they use two different computers with two different operating systems: Microsoft Windows and Apple OS X, located in two different places. When the capture is complete, the databases produced by both computers are merged without redundancy by a simple algorithm. The user interface served by their architecture allows users to watch the tweets captured on an embedded Google Maps map. To quickly retrieve tweets for the map, they are stored in a geographical index. This index is based on the joint use of GeoDjango and a spatial database. This basic spatial database is a PostgreSQL database with the PostGIS spatial extension. It allows querying ranges on location points.

At the end of October 2012, A. Black, C. Mascaro, M. Gallagher, and S. P. Goggins describe their architecture, named Twitter Zombie, to capture, socially transform and analyze the twittosphere. This architecture aims to provide a consistency in the results of social sciences and to standardize the data collecting in order to be able to reproduce observations identically and afterwards [33]. It relies on the Search API of Twitter REST APIs and is written in PHP. The collected data are stored in a MySQL database. Twitter Zombie retrieves data from Twitter by running independent research jobs in continuous and on regular basis. Jobs are programmed, configured, and stored in a MySQL database. This system is launched each minute by the cron Linux scheduler. The scheme of the database has been optimized for insertions in order to prevent the storage of data to be a bottleneck when an important event occurs. This database tweak makes its size grows quickly. To build the jobs, they use the advanced search page of Twitter's website because it validates the search criteria and produces an URL they can reuse during their calls to the Twitter APIs.

In December 2013, B. Molnar and Z. Vinceller publish the results of a comparative study between five architectures designed to investigate the OSNs, and propose a new approach based on them [18]. They observed that the majority of the studied architectures uses open-source software and primary data are manipulated either by a central relational system or central a NoSQL system, however NoSQL systems store documents more quickly. They also noted that the hardware architectures rely mostly on "commercial off-the-shelf" components. They identified several problems these architectures are brought to meet. First, data recovery is often limited by the APIs and the technical reception capabilities. Second, if a real-time analysis is required, it takes a lot of resources to retrieve all the data and analyze them correctly. Third, it is difficult to make textual analysis on OSNs because there are great differences between them and it requires specific routines for each of them. Finally, the link structure between messages of OSNs differs greatly from traditional website connections and a storage issue arises from this difference. It should be solved in a different way, and performance and efficiency become central to explore those links. To address the performance issues related to time and storage, they propose to use HADOOP, a highly scalable analytics platform for processing large volumes of structured and unstructured data, and MapReduce processes as much as possible.

November 29th, 2011, T. Hoff describes the physical and logical architecture used by DataSift [15] to mutualize the expensive Firehose of Twitter. They redistribute data to developers who can't afford the cost of the Firehose and the charge of having a big dedicated hardware architecture. Indeed, at the time, he said accessing to the Firehose was worth $25 000 per day for a daily volume of 250 million of tweets. 30 peoples and 4 years of development were necessary to build a system that used 936 processors and many SSDs. The company was using C++ for critical components, PHP to provide an API to its clients, Java/Scala to communicate with HBase and launch Map/Reduce tasks, MySQL, an Hbase cluster (30 hadoop nodes, 400TB of storage) and Memcached as cache. DataSift transformed tweets before they we redistributed, they added to them informations such as their language, their feelings, the gender of their author and their Klout level (a social influence indicator). Customers were billed in real time depending on the amount of service used. This service was closed August 13, 2015 following the announcement of April 11th, 2015 about the end of the partnership between Twitter and DataSift. [16]

In 2012, D. Preot¸iuc-Pietro, S. Samangooei, T. Cohn, N. Gibbins, and M. Niranjan present the framework they developed to efficiently proceed texts resulting from data flows of OSNs [17]. This framework provides command line tools to treat tweets, already captured, and live flows. It works with modules. To deal with the huge amount of data to process, they make use of the MapReduce framework to distribute calculations and data storage on a cluster of several computers. Their idea is to chain analysis tasks in a workflow. Each task

can add new metadata to the processed tweet but can't modify it or its already existing metadata. At the time of writing their paper, they already developed 3 modules for their framework: Tokenization, which cuts the text of a tweet and identify various entities, Language Detection, which automatically detects the language of a tweet, and Stemming, which retrieves root words for easier analysis.

We've seen that depending of the goals of the platform and its final users' needs, the type of database employed to store data is different. Centralized databases become bottlenecks when the number of tweets to save explodes, this case would more likely happen when using the Streaming APIs. We've also observed that platforms offer a variety of user interfaces and features. User interfaces play a significant role in the designing process of a platform. They are deeply related to the analysts' needs and the possibilities given by the OSNs' APIs.

V. User Interfaces

User interfaces are fairly standard and depend on the user's expertise level. We can therefore find search engines type Web interfaces [21] with rich refunds [37] such as graphics or word clouds. We can also find map-based interfaces [32] or already preconfigured interfaces that display results such as current trends [8]. Conversely, the analyst will launch guidelines on the command line or use environments like R [13] or Weka.

The design of the user interfaces reveals two issues. The first is to identify information to input and to return, and under what form. The second is to identify what one seeks to observe and translate it into a treatment to be applied to the inputs. Depending on the level of expertise of the analyst, this treatment will be either flexible or rigid. In general, in the case of Twitter, the number of entries is limited to the available types of metadata. The current common use is limited to enter targeted keywords or hashtags and possibly specify the duration of the capture. It is also possible to filter the results according to various criteria (e.g., retweets, geographic location, time of creation, etc. [17, 22, and 32]) but the use of these filters often requires a good level of expertise from the analyst. In addition to filters, many various, more or less conventional treatments are possible in order to extract knowledge from tweets, For instance, it is possible with a suitable language processing to identify the polarity of a tweet, the gender of its author and his age, or even his socio-economic categories. These treatments are complex and the results are sometimes very rough, but they allow understanding the tweets from new angles. These operations primarily based on language processing have yet to be discovered or improved (see the features of data mining tools).

The results to return and their forms remain a relatively open question. A simple and classic level of use is the quantitative representation. We can, for example, visualize the popularity of an event by counting the number of tweets and retweets related to a specific hashtag, either instantly or by representing its evolution over a period of time in the form of a curve. We can also carry out this measurement in comparative form, for instance, if one wishes to compare the popularity of both politicians. Things get complicated if you want to add more dimensions to the analysis because in this case, the classical representations lose of their readability. For instance, the representation of the opinion over a period classified by genders concerning the individuals who are candidates in an election is a real headache in terms of representation.

The user interfaces also provide management capabilities, monitoring, security management, etc. This is particularly true in the particular case of platforms that make use of distributed database and are often stored in cold and remote areas.

VI. Case Study

According to figure 1, we designed a partially distributed architecture based on the Twitter Streaming APIs to offer a SaaS (i.e., Software as a service) to scientists and policymakers. Our platform allows longitudinal studies of various subjects in near real time. Users can specify the words they are interested in. Our system merges all their wishes in a list of keywords to track and sends it to the Streaming APIs as parameter. Plus, we cover a large spectrum of topics thanks to associative networks. Thus, we offer a high level of flexibility. Generally, to start querying our service, users don't have to wait while data are collected because data are already collected.

We use as data storage the distributed search engine Elasticsearch. It is based on Apache Lucene and is open source. This technological choice has several advantages for collecting tweets. Firstly, it tokenizes the tweets during their indexing, allowing us to have real-time and full-text search capabilities. Thus, we can observe trends in real time through our user interface. Secondly, Elasticsearch maintains updated replicas of shards of the tweets index to prevent an eventual data loss caused by hardware failures. Thirdly, when the number of tweets sent by Twitter APIs increases sharply, it is crucial to have a system that is very quick to perform inserts in order to avoid to be disconnected [40]. Elasticsearch shows up to two times faster than MySQL for data insertion [20]. Finally, tweets sent by Twitter APIs are JSON objects that Elasticsearch can index without conversion thanks to its JSON document-oriented side. Elasticsearch also supports plug-ins. We used to collect tweets with the plug-in Twitter River, but we recently replaced it with a homemade Python service, which uses the Tweepy library, because river type plug-ins were removed in Elasticsearch 2 [34]. The interactions with Elasticsearch rely on HTTP REST requests and queries are JSON objects, so Elasticsearch is developer friendly.

We use a total of 5 computers to operate our architecture (2 x Intel Xeon 6 cores @ 3.20 GHz, 1 Intel Xeon 4 cores @ 2.66 GHz, 1 Intel Xeon 4 cores @ 2.4 GHz, and 1 Intel Core 2 Quad Q9650 4 cores @ 3.00GHz). We currently have a storage capacity of 12TB disk and 100GB RAM. Among our 5 computers, 4 are on a private local network and play the role of nodes in our Elasticsearch cluster. One of them also runs our services. The computer number 5 can be reached from Internet and serves as front-end and security gate to our data. Our architecture has the advantage of being scalable. To expand its hardware capabilities, we can easily add one or more computers to our Elasticsearch cluster. However, to do so we need to reindex all the data to create enough shards of the current tweets index in order to populate all the freshly added computers with them. But eventually, the reindex process is performed without having to stop the cluster and is transparent for users of our SaaS.

Like we started to mention them above, our platform is also composed of homemade services, they are all written in Python and use the official Elasticsearch library. One of them ensures that the recovering of tweets is not faulty. If this is the case, it tries to revive in autonomous way the collecting service and notifies administrators that something went wrong. Having a

reliable capture is necessary to obtain accurate results for longitudinal studies. Some others services are related to data analysis. They compute new data from tweets and add them to the tweets' metadata. The provided new data are for instance the gender of authors or the polarity of tweets. Finally, we also created some services to build and manage caches of buzzing words in order to speed up the buzz observatory of our SaaS.

Scientists and policymakers use our platform through a web application that consists in a server part (back-end) powered by Node.js and a client part (front-end) written mostly using AngularJS and jQuery. The server side of this app is primarily a security layer between the Elasticsearch cluster and connections from the Internet. The client side (front-end) provides several tools to end users like a comparative tool with charts, words clouds, a buzz observatory, etc. When the front-end needs to load or refresh an AngularJS directive (e.g., a chart, a list of tweets, etc.), the request is transmitted to the server side. Then, the server queries the Elasticsearch cluster and during that time, it performs requests from other users until it finds no other task to do. There is no blocking process thanks to the mass use of callback functions.

We evaluated the performances of our SaaS using the following method. We opened the comparative tool of our front-end app with Google Chrome (version 50.0.2661.102) on our local network to avoid possible lags from Internet. Each time a user enters an expression in the comparative tool with a start and an end date, the controller of the tool updates the page's directives' settings. These updates trigger many queries to retrieve all the needed data. We choose the expressions "USA", "Paris" and "You" as unique words and in combination for our evaluation. Each evaluated expression involves 10 queries and the combination fires a total of 30 queries. We made use of the Network tab of the Chrome Developer Tools to observe when all the queries started and when the latest answer arrived. The results are shown in the bellow figures 2, 3 and 4.

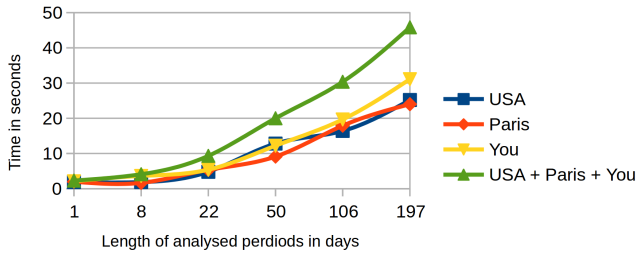

Fig. 2. Loading time of the comparative tool with different expressions

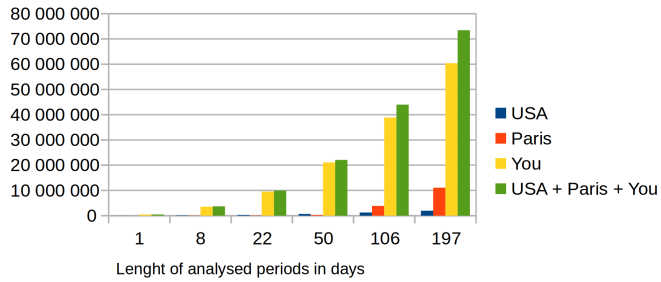

Fig. 3. Number of tweets returned by the queries

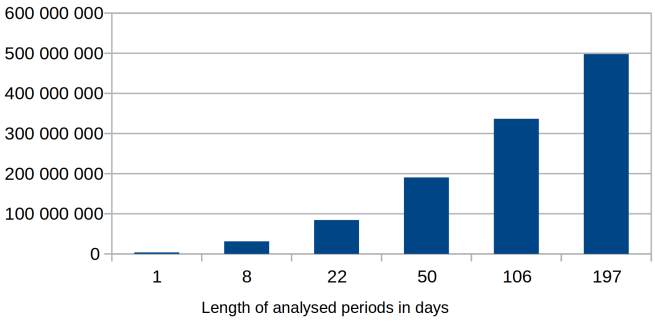

Fig. 4. Number. of tweets stored in Elasticsearch per analysed period

We could think that the loading time is deeply linked to the number of tweets returned by the different queries. However, we can see in the figure 2 that the expressions "USA" and "Paris" have similar loading times while figure 3 shows us that for the 197 days long period, "Paris" returned a number of tweets more than 4 times higher than "USA". Actually, the loading time seems to be more influenced by the total volume of tweets present in our database (figure 4) than the success rate of the queries (figure 3). Eventually, during our evaluation, we also observed the following bottleneck. The loading times may vary for our users according to the web browser they use. Indeed, the maximum number of concurrent Ajax requests is differently limited per domain (e.g. 6 requests in Google Chrome and 13 in Microsoft Internet Explorer 11 [35]), making many Ajax requests wait for empty slots to be sent by the web browser while our architecture could handle them.

Among the tools provided by the front-end is another tool called exportation tool that give more flexibility to our users. Indeed, it allows reusing our collected data. They can download the original tweets enriched with the extra metadata added by our different analysis services, such as the gender of authors or the polarity of the tweets. It supports JSON and CSV formats and has a control panel to adjust the selection of tweets to export.

VII. Conclusion

This article presented the challenges and some possible solutions for the realization of a platform for collecting and analyzing tweets. Let us note first that such architecture is closely linked to the organization of Twitter. Indeed, a simple change in the Twitter APIs imposes a change in the collecting process of the platform, otherwise the whole system will stop working or the results will be corrupted or incomplete.

Regardless of this, the biggest difficulty is related to the power (treatment, storage) necessary to support the Twitter Streaming APIs, which send millions of tweets to their consumers each day. We saw that some software architectures, such as relational databases, are less appropriate than NoSQL ones during the data recovering except for short time punctual analysis. Plus, distributed databases perform better and prevent a data loss thanks to their replicated nature.

User interfaces play a primary role in the process of designing a OSNs' data analysis platform because they are deeply related to the possibilities offered by the OSN's APIs and their limits, the needs of its future users and administrators, and the possible fields of data-mining.

We developed a platform that offers a SaaS. It uses associative networks to cover a large spectrum of topics

because we wanted to anticipate the needs of our users. We recover data using Streaming APIs for the reason that they allow recovering millions of tweets each day. Furthermore, we chose the distributed search engine Elasticsearch to store tweets. It provides a distributed database system suited to support sudden tweets reception rises while tokenizing tweets during their indexing, making queries in near real time possible. All these choices were motivated by the will of giving always more flexibility to analysts.

However, even with our over-sized platform and the technical choices we made, we've seen that the question of performance is still valid because we are accumulating day by day tweets and over a longer and longer period. Indeed, the more we have tweets, the more the performances are low. This raises several questions. Should captured data have an expiration date and thus reduce the flexibility of the analyst? Could a peer-to-peer architecture have a better cost/power ratio than already existing platforms?

ACKNOWLEDGMENT

This work was realized thanks to funding from the FUI Camille 3DS project. We wish to thank the partners and funders of the project.

REFERENCES

[1] J. James, 'Data never sleeps 3,0', 2015. [Online]. Available: https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/. [Accessed: 4- Mar- 2016]

[2] J. Dorsey, 'just setting up my twttr', 2006. [Online]. Available: https://twitter.com/jack/status/20. [Accessed: 4- Mar- 2016]

[3] B. Kiprin, 'The Meanings of a Favorite and Retweet', 2014. [Online]. Available: http://borislavkiprin.com/2014/01/27/meanings-favorite-retweet/. [Accessed: 9- Mar- 2016]

[4] Twitter, 'Twitter Cards', 2012. [Online]. Available: https://dev.twitter.com/cards/overview. [Accessed: 10- Mar- 2016]

[5] A. R ಠ_ಠ mann-Kurrik, 'Introducing new metadata for Tweets', 2013. [Online]. Available: https://blog.twitter.com/2013/introducing-new-metadata-for-tweets. [Accessed: 9- Mar- 2016]

[6] R. Krikorian, 'Map of a Twitter Status Object', 2010. [Online]. Available: http://online.wsj.com/public/resources/documents/TweetMetadata.pdf. [Accessed: 4- Mar- 2016]

[7] V. Gupta, G. S. Lehal, "A Survey of Text Mining Techniques and Applications," in Journal of Emerging Technologies in Web Intelligence, Vol. 1, No. 1, August 2009

[8] T. Sakaki, M. Okazaki, Y. Matsuo, "Earthquake shakes Twitter users: real-time event detection by social sensors," In Proceedings of the 19th international conference on World Wide Web (WWW '10). ACM, New York, NY, USA, 851-860. 2010

[9] M. Mathioudakis, N. Koudas, "TwitterMonitor: trend detection over the twitter stream," In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 1155-1158. 2010

[10] Twitter, 'Twitter API', 2006. [Online]. Available: https://web.archive.org/web/20061109100343/http://twitter.com/help/api. [Accessed: 9- Mar- 2016]

[11] M. A. Russell, Mining the Social Web, 2nd Edition. O'Reilly Media. October 2013

[12] M. Verdone, 'Python Twitter Tools (command-line client and IRC bot)', 2009. [Online]. Available: http://mike.verdone.ca/twitter/. [Accessed: 16- Mar- 2016]

[13] J. Gentry, "Twitter client for R". March 18, 2014.

[14] J. Gentry, 'twitteR now supports database persistence', 2014. [Online]. Available: http://geoffjentry.blogspot.fr/2014/02/twitter-now-supports-database.html. [Accessed: 5- Mar- 2016]

[15] T. Hoff, 'DataSift Architecture: Realtime Datamining At 120,000 Tweets Per Second', 2011. [Online]. Available: http://highscalability.com/blog/2011/11/29/datasift-architecture-realtime-datamining-at-120000-tweets-p.html. [Accessed: 15- Mar- 2016]

[16] N. Halstead, 'Twitter Ends its Partnership with DataSift – Firehose Access Expires on August 13, 2015', 2015. [Online]. Available: http://blog.datasift.com/2015/04/11/twitter-ends-its-partnership-with-datasift-firehose-access-expires-on-august-13-2015/. [Accessed: 15- Mar- 2016]

[17] D. Preotiuc-Pietro, S. Samangooei, T. Cohn, N. Gibbins, M. Niranjan, "Trendminer: An Architecture for Real Time Analysis of Social Media Text". 2012

[18] B.Molnár, Z. Vincellér, "[19] Comparative study of Architecture for Twitter Analysis and a proposal for an improved approach". 2013

[19] S. Kumar, F. Morstatter, H. Liu, Twitter Data Analytics, Book Springer. 2013.

[20] D. Gouyette, 'For my research, MySQL or Elasticsearch?', 'Pour ma recherche, MySQL ou Elasticsearch ?', 2012. [Online]. Available: http://www.cestpasdur.com/Elasticsearch/2012/04/01/Elasticsearch-vs-mysql-recherche.html. [Accessed: 4- Mar- 2016]

[21] P. Dyer, '50 Top Tools for Social Media Monitoring, Analytics, and Management', 2013. [Online]. Available: http://pamorama.net/2013/05/12/50-top-tools-for-social-media-monitoring-social-media-analytics-social-media-management-2013/. [Accessed: 9- Mar- 2016]

[22] P. Dyer, '20 Top Twitter Monitoring and Analytics Tools', 2010. [Online]. Available: http://pamorama.net/2010/04/26/20-top-twitter-monitoring-and-analytics-tools/. [Accessed: 9- Mar- 2016]

[23] L. Maan, Y. Abutaleb, 'Twitter disappoints investors as user growth hits wall', 2016. [Online]. Available: http://www.reuters.com/article/us-twitter-selloff-idUSKCN0VK1LJ. [Accessed: 10- Mar- 2016]

[24] Microsoft, 'Analytics for Twitter', 2011. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=26213. [Accessed: 7- Mar- 2016]

[25] Google Developers, 'Social Interactions - Web Tracking (ga.js)', 2015. [Online]. Available: https://developers.google.com/analytics/devguides/collection/gajs/gaTrackingSocial. [Accessed: 8- Mar- 2016]

[26] Rapidminer, 'Using the Twitter Connector'. [Online]. Available: http://docs.rapidminer.com/studio/how-to/cloud-connectivity/twitter.html. [Accessed: 4- Mar- 2016]

[27] MOA, 'MOA (MASSIVE ONLINE ANALYSIS)', 2016. [Online]. Available: http://moa.cms.waikato.ac.nz/. [Accessed: 7- Mar- 2016]

[28] Cathy Pearl, 'Using KNIME to Find Out What Your Users Are Thinking', 2015. [Online]. Available: https://www.knime.org/blog/using-knime-to-find-out-what-your-users-are-thinking. [Accessed: 9- Mar- 2016]

[29] P. Earle, D. C. Bowden, M. R. Guy, "Twitter earthquake detection: Earthquake monitoring in a social world," Annals of Geophysics, 2011.

[30] Amy Mitchell et al., "he Evolving Role of News on Twitter and Facebook", Pew Research Center, 2015.

[31] K. Makice, Twitter API: Up and Running. O'Reilly Media. 2009

[32] M. Oussalah, F. Bhat, K. Challis, T. Schnier, A software architecture for Twitter collection, search and geolocation services. Know.-Based Syst. 37 (January 2013), 105-120.

[33] A. Black, C. Mascaro, M. Gallagher, S. P. Goggins, Twitter Zombie: Architecture for Capturing, Socially Transforming and Analyzing the Twittersphere.

[34] Elastic, "Rivers were deprecated in Elasticsearch 1.5 and removed in Elasticsearch 2.0.", 2016. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/rivers/current/index.html. [Accessed: 25- Mar- 2016]

[35] Browserscope, 2016. [Online] Available: http://www.browserscope.org/?category=network&v=top. [Accessed 15-May-2016

[36] M. Bryant, "Twitter to cut off firehose resellers as it brings data access fully in-house", 2015. [Online]. Available: http://thenextweb.com/dd/2015/04/11/twitter-cuts-off-firehose-resellers-as-it-brings-data-access-fully-in-house/. [Accessed 25-May-2016]

[37] R. D. W. Perera, S. Anand, K. P. Subbalakshmi and R. Chandramouli, "Twitter analytics: Architecture, tools and analysis," MILITARY

COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010, San Jose, CA, 2010, pp. 2186-2191

[38] "Twitter River Plugin for Elasticsearch", 2015. [Online]. Available: https://github.com/elastic/elasticsearch-river-twitter. [Accessed 25-May-2016]

[39] T. Sherman, "Coming soon: express even more in 140 characters", 2016. [Online]. Available: https://blog.twitter.com/express-even-more-in-140-characters. [Accessed 25-May-2016]

[40] Twitter, "Connecting to a streaming endpoint", 2016. [Online]. Available: https://dev.twitter.com/streaming/overview/connecting. [Accessed-26-May-2016]