Universidad Carlos III de Madrid

e-Archivo Institutional Repository

This is a postprint version of the following published document:

Mariluz Congosto, Pablo Basanta-Val, Luis Sánchez-Fernández, T-Hoarder: A framework to process Twitter data streams, In *Journal of Network and Computer Applications* (2017), vol. 83, pp. 28-39
https://doi.org/10.1016/j.jnca.2017.01.029

© 2017 Elsevier Ltd. All rights reserved.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

T-Hoarder: A framework to process Twitter data streams

Mariluz Congosto, Pablo Basanta-Val, Luis Sánchez-Fernández

With the eruption of online social networks, like Twitter and Facebook, a series of new APIs have appeared to allow access to the data that these new sources of information accumulate. One of most popular online social networks is the micro blogging site Twitter. Its APIs allow many machines to access the torrent simultaneously to Twitter data, listening to tweets and accessing other useful information such as user profiles. A number of tools have appeared for processing Twitter data with different algorithms and for different purposes. In this paper T Hoarder is described: a framework that enables tweet crawling, data filtering, and which is also able to display summarized and analytical information about the Twitter activity with respect to a certain topic or event in a web page. This information is updated on a daily basis. The tool has been validated with real use cases that allow making a series of analysis on the performance one may expect from this type of infrastructure.

1. Introduction

Recently, the generalization of micro blogging (Java et al., 2007; Honey and Herring, 2009) by a wide sector of society has contributed to transform the way in which people consume information, spread it, and interact with others. The availability of mechanisms to publish short messages, pictures, and videos, together with their use from mobile phones, allows that information to flow in real time through a network of users. This change also transforms the way data is processed and integrated in different business models which take into account the information stored in micro blogging records to determine the next set of decisions to be taken in a business infrastructure (Gudivada et al., 2015; Manyika et al., 2011; Zikopoulos and Eaton, 2011).

Twitter is the most popular micro blogging network.[2] It is char acterized by the limit in the length of its messages, called tweets (140 characters) and by the asymmetric relations between their users. At the moment, 500 million tweets are published daily, which means a huge source of social data that has caught attention from researchers. From 2008, it is also a source of study by academic researchers (Huberman et al., 2008) and it has been applied in a number of social fields such as elections (Conover et al., 2010; Gayo Avello, 2011; Barberá and Rivero, 2012), social movements (Peña lópez et al., 2014), predictions (Bollen et al., 2011a; Asur and Huberman, 2010), user's influence (Cha et al., 2010), behavior (Bollen et al., 2011b; Dodds et al., 2011), and message propagation analytics (De Domenico et al., 2013).

Currently, it is possible to accede to the complete volume of information on Twitter by means of payment through GNIP.[3] Also, researchers may access a partial set of tweets and collect the data by means of specific Twitter APIs. Since its inception, Twitter APIs can be used without any specific restrictions. This favored the creation of services that collected data like tweetbackup.com or Twapper Keeper.[4]

However this initial configuration changed a few years ago (September of 2012), with a new rule of use for the APIs that prevented users sharing tweets.[5] Currently, the rules of Twitter's API[6] only allow sharing datasets of a limited size. In this context, some tools emerged to process tweets directly from Twitter. Currently, the set of tools that can be used to access tweets includes TwapperKeeper (OBrien, 2011), Twitter Tap (Kranjc, 2014) and Twitterstream to Mongodb (Del Fresno, 2012). However, their specific constraints bring in new concerns and as a result more processing capacity is required. In addition, the specific implementation decisions hamper the type of analysis one may carry out with this type of tools.

In a reaction to produce a more open, cost effective, and flexible

[2] ⟨https://about.twitter.com/company⟩.
[3] ⟨https://gnip.com/⟩.
[4] ⟨https://twapperkeeper.wordpress.com/2011/02/22/removal-of-export-and-download-api-capabilities/⟩.
[5] ⟨https://dev.twitter.com/archive/terms/api-terms/diff-20130702⟩.
[6] ⟨https://dev.twitter.com/overview/terms/policy⟩.

approach to Twitter data stream processing, T Hoarder has been proposed. It is a minimal framework that stores data into folders, providing a lightweight approach to process tweets. Additionally, T Hoarder includes an engine that carries out data analytic processing and displays them in three axes: time, space, and relevance. T Hoarder is designed to deal with a medium to large number and size of datasets, to gather data for long periods of time (years), to be able to analyze propagations, and to identify the origin of data. Its architecture is also of interest because its underlying ideas can be used in other available engines for stream processing. It also provides solutions to common problems in the way data is processed and/or graphically displayed.

The rest of this article introduces T Hoarder infrastructure in detail. It starts by an introduction to the necessary aspects that discuss the quality of the information available in Twitter (Section 2), which establish the computational limitations related to access tweets. It continues with an analysis of the most related work initiatives (Section 3), which are compared with T Hoarder. Then, T Hoarder is properly described (in Section 4) to be later evaluated in Section 5 with real traces taken from Twitter and processed with T Hoarder. Lastly, Section 6 draws conclusion and also describes our ongoing work.

2. Tweets as source of information

Twitter was a pioneer in opening its data by means of APIs from 2006. This resulted in the emergence of multiple applications that access and process Twitter data and has made it easier for researchers to obtain datasets for their experiments. In 2009, Twitter allowed access to the stream of tweets by means of the Streaming API, a restricted version of a more complete approach called "Firehose".

2.1. Accessing Twitter APIs

The versions of their APIs have changed the form of access, the amount of information by time unit offered to the user (see Table 1). Currently, the access to tweets is allowed using the following mechanisms:

- Twitter API V0.0: from 2006 to June of 2010.
- Twitter API V1.0: from June of 2010 to June of 2013. The basic authentication, based on user/password, in REST API, disappeared and it is replaced by OAuth (Leiba, 2012).
- Twitter API V1.1: from June of 2013 to date. It integrates the search API in the Rest API, extending the authentication by OAuth to the Streaming API and modifying the speed limits. At the moment of writing this article, Twitter provides two APIs to retrieve data: the REST API and the Streaming API.
  1. REST API: it allows all type of queries to the data of Twitter in a synchronous form.
  2. Streaming API: it establishes a socket between Twitter and a server by which information is asynchronously received.

T Hoarder uses the Tweepy library (tweepy.org) that resolves low level access to the API. This library is widely used and updated to the latest API version. However, the source code is available in the Github repository. On the other hand, so far Twitter has announced new versions with almost a year in advance allowing to face the changes. In the last three years the APIs changes have been limited to the JSON structure of the tweets.

In both APIs it is necessary to consider the restrictions that Twitter imposes in its use:

1. *Speed limit*: It restricts the number of queries to the REST API for fifteen minutes. The limit depends on the type of query. In the case of the Streaming API, the tweets per second (TPS) are limited to 50.
2. *Time recover limit*: It is possible to recover tweets from previous seven days with `/search/tweets` GET method of the REST API. The last 3200 tweets of a user with method GET `/statuses/ user timeline` of REST API and no previous tweets are available with Streaming API.
3. *Tweets sample*: Only the `/statuses/ user timeline` GET method of the REST API provides a complete sample of the last 3200 messages of a user. In the `/search/tweets` method GET of the REST API and the API streaming it is provided only a reduced percentage of tweets (85 95%), even when the frequency is less than 50 tweets per second.

2.2. Secure access

In order to access an API of Twitter, it is necessary to authenticate via the OAuth protocol. In practical terms, it is required to create an application in Twitter to obtain the keys of access for that application (*Consumer Key* and *Consumer Secret*). Also, it is necessary to have a user in Twitter to generate the access key, called *Access token*, for that application. This way, Twitter knows what user accesses and from which application.

Within the options of the Streaming API, it is possible to obtain tweets by means of random filtering, and also to the total stream of tweets. This option is only available for resellers of data; however, since its inception, it has been available for some researchers.

Currently, it is possible to select a set of tweets by one of these three forms:

1. *Keywords*: it allows obtaining the tweets that adjust to a pattern search providing a list of words, and comma separated expressions. The expressions are words separated by spaces that are in a message, with independence of the order in which they appear, but fulfilling the condition. The commas that separate the words or expressions act like OR logic operators and in an expression the spaces that separate the words behave like AND logical connectors. For instance, the search "`#metromadrid, Madrid metro`"would provide tweets that contain `#metromadrid` OR (`metro`AND `Madrid`)
2. *By users:* it allows retrieving tweets of the users provided by means of a list with its separated identifiers by commas. Also, all the reactions to these messages are obtained including automatic or manual broadcastings (RTs), mentions, appointments, and answers.
3. *By geo localization:* In this case, application receives tweets published from one or several rectangular geographic zones, each of them delimited by a superior east coordinate and another inferior west coordinate.

Table 1
Comparison among the main methods to access Tweets.

API	Recover	Search type	Delivered Tweets
Streaming	None	Keywords, users, or locations	If speed < 50 tweets per second, it recovers 80 90% of tweets
REST GET search	1 week back	Keywords, users, or locations	Unknown, because it only delivers the most relevant tweets. It is always less information than Streaming API.
REST GET statuses	3200 Last tweets	Tweets that belong to one user	All

Table 2
Comparison among the main methods to access Tweets.

	Data Collection from Twitter	Dependency on DB software	Open Source
TwapperKeeper	Yes	Yes	Yes
Twitter-Tap	Yes	Yes	Yes
twitterstream-to-mongodb	Yes	Yes	Yes
T-Hoarder(our approach)	**Yes**	**No**	**Yes**

These three options are mutually excluding; it is not possible to look for more than one option simultaneously. When it is necessary to monitor users and keywords, the developer has to resort to solutions that process in parallel, fusing, and eliminating duplicated messages later. For instance, this case is relatively frequent in the tracking of electoral campaigns in which it is interesting to know the conversation around the candidates (users) and the mottos for the campaign (represented as a set of keywords).

3. State-of-the-art

Since its inception in 2006, the amount of data available on Twitter has grown exponentially, producing 500 million tweets daily. There is also an increasing number of tools available to process data coming from Twitter in order to obtain valuable information.

Our comparison starts by analyzing different approaches for data collection. The main approaches that can be compared with T Hoarder include TwapperKeeper (OBrien, 2011) TwitterTap (Kranjc, 2014), and TwitterStrean to MongoDB (Del Fresno, 2012). The main difference among these basic crawlers and T Hoarder is that T Hoarder is neutral, referred to the type of storage required (see Table 2). Most approaches store tweets in large data bases that can be processed later. In contrast, T Hoarder stores this information in a plain file system that can be accessed later for offline analysis.

In the state of the art (Goonetilleke et al., 2014), there are a number of platforms that compare with T Hoarder:

- `Twitter Echo`. (Bošnjak et al., 2012) . Twitter Echo is a Twitter crawler with a fault tolerant architecture, with support for reliable data storage, and a modular crawling infrastructure.
- `Twitter Zombie` (Black et al., 2012). Twitter Zombie is a meth odological approach and technical tool for Twitter data collection. It has been used in the US Republican Party Presidential debate for South Carolina elections to analyze their data.
- `Twitter Analytics` (Stavrakas and Plachouras, 2012). It pro vides the application with a 3 Layer infrastructure: a crawling, an integration modeling and a data analysis layers. Its architecture is intended for campaign analytics and opinion mining.
- `Trendminer` (Preotiuc Pietro et al., 2012). The last approach is trend miner: an open source framework for efficient text processing. It is based on the map reduce processing model and has a complex processing engine. Proposed T Hoarder shares with Twitter Miner the idea of having an infrastructure which is able to process a huge number of data in a proper way.

Comparing T Hoarder with the other related frameworks for Twitter (see Table 3), it may concluded that its main novelty is its flexible and lightweight internal architecture, which allows for efficient management of big and long lasting Twitter experiments. This storage architecture allows users to explore data offline, before it is going to be processed to produce an analytic result. None of the previous frame works have this type of support, because they were designed as simple collection infrastructures or as specific analytic tools. Previous efforts

Table 3
Comparison among the different management frameworks.

	Type of analytics	Integrated visualization	Complexity
Twitter-Echo (Bošnjak et al., 2012)	Tweets collection	No	Med
Twitter- Zombie (Black et al., 2012)	Tweets collection	Yes	Med
Twitter Analytics (Stavrakas and Plachouras, 2012)	Opinion mining	No	Med
Trendminer (Preotiuc-Pietro et al., 2012)	Tweet Analysis	No	High
T-Hoarder (our approach)	Tweets collection, location and spread	Yes	Low

are more focused on the requirements of specific analytics, while T Hoarder is more neutral on the type of analysis carried out.

4. T-Hoarder in a nutshell

This section describes the architecture of T Hoarder, departing from the high level architectonic perspective to the different aspects related to its interfaces.

4.1. Overview

T Hoarder was conceived to store tweets about certain hot topics in which the different types of propagation patterns may be analyzed in large datasets. At the moment, T Hoarder is able to show the evolution in time of a set of subjects of social interest like citizen mobilizations, political opinions on crisis, scandals, and corruption (Congosto and Aragón, 2012; Peña lópez et al., 2014; Congosto, 2014, 2015). The T Hoarder platform stores tweets by thematic lines and it automatically processes them in three axis: time, space, and relevance. The time axis allows considering the temporal evolution of a set of indicators like the most mentioned tweets and more active proportion of retweeted messages, popular users, popular *hashtags*, and the most frequent words. The space axis locates tweets geographically and the relevance shows the most wide spread messages. Also, T Hoarder has an interactive graphical interface that easies navigation in these three axes. The platform was designed with a minimalist architecture that avoids the dependency on other software packages.

T Hoarder opted for UNIX as a reference operating system and Python as a main programming language for the environment. The use of databases was discarded and T Hoarder decided to rely directly on the UNIX file system because:

1. It can run with minimum execution environments such as a Raspberry PI.[7]
2. It is able to facilitate the transfer of the data from one server to another. When the data bases are large, the backup/restore activities are problematic.
3. It has not defined (a priori) any data modeling, not knowing the type of different information that is going to be processed.
4. It enables easy integration of different datasets.
5. It does not require access to random information since the proces sing of the data is sequentially defined.

A three layer disconnected architecture was defined to avoid that the execution of a layer interferes in other elements of the architecture. The communication between these layers is always done using files.

[7] ⟨http://www.raspberrypi.org/⟩

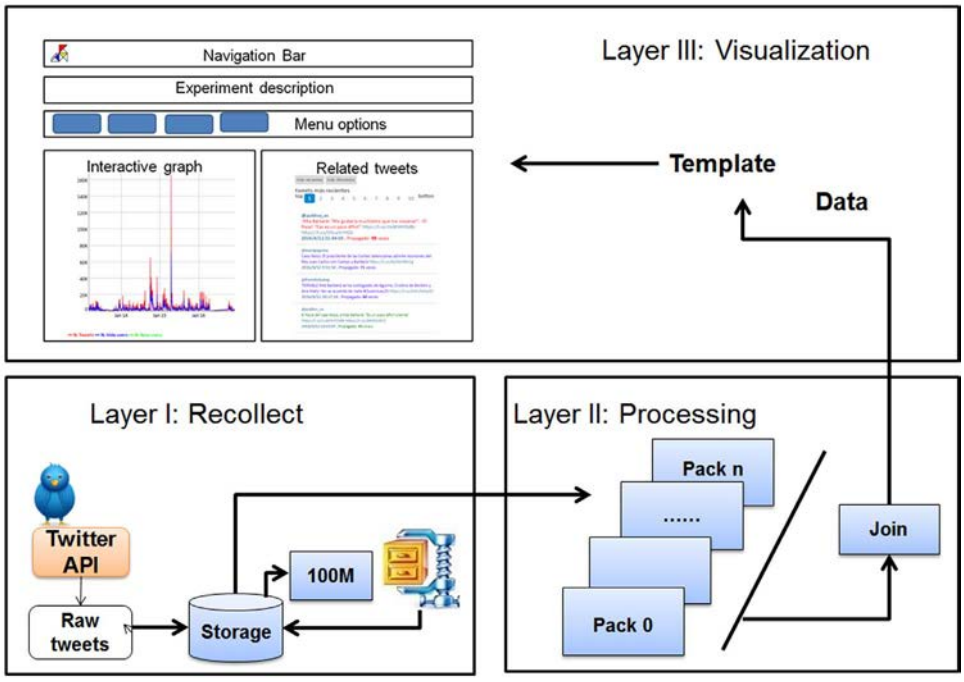

Fig. 1. Three levels of T-Hoarder architecture: data collection, processing and visualization.

The functional division in T Hoarder is the following:

1. Layer 1: Data collection and storage
2. Layer 2: Data processing
3. Layer 3: Visualization

Fig. 1shows the functional layers and how they exchange information. Layer 1 collects data from the Twitter API and stores it in compressed packets when the size reaches 100 MB. Layer 2 processes the compressed packets separately. It is only necessary to process them once. Additionally it processes the latest information not yet compressed by having a size of less than 100 MB and integrates the results of all the information processed. Layer 3 displays the results on the web.

The source code of T Hoarder is publically available in a GitHub repository (Congosto, 2015).

4.2. Layer 1: data collection and storage

Following the mechanism imposed by Twitter APIs, T Hoarder registers an application and a set of users of such application on Twitter. In order to be able to access the Twitter APIs by means of OAuth, it is necessary to create the access keys for the application and the users. These keys are generated in the component `tweet auth`, available on Congosto (2015), and stored in files so that the components of T Hoarder can use them to access to the APIs.

T Hoarder uses the Streaming API instead of the REST API for the following reasons:

1. It is the method most adapted to obtain data in real time with the only limitation of the maximum rate of 50 tweets per second. The alternative would be the use of the method GET `/search/tweets` of the REST API that force periodic requests to obtain the messages with the consequent difficulty in determining the sampling frequency.
2. T Hoarder is designed to cope with long lasting Twitter experiments. In another type of short duration experiments (days) and in which it is necessary to go back to a previous date, such as the Trending Topic (TT) it is not necessary the use of T Hoarder. It is more reasonable to obtain tweets with method GET `/search/tweets` of the REST API with which they will be possible to be recovered up to seven previous days and to obtain a complete dataset.

In the options of the Streaming API, T Hoarder preferably uses keywords and users. Geolocation is information that lacks thematic context and represents a small sample of tweets (e.g. 1.5% in Spain).

Tweets are stored with instances of the `tweet streaming`component (available on Congosto (2015)), executed in parallel. The API provides tweets in JSON format (Crockford, 2016). Then, the data are transformed to flat text in which the information for each tweet is stored in a line with tabulator separated fields. This allows to read the messages that are taken with this facility and to introduce them in a spreadsheet.

Of all the received information, the selected data that is stored and used in the analysis layer are:

1. `id tweet`: Identifier of a tweet. It is an increasing number that is sequentially assigned to each message.
2. `timestamp`: date and GMT hour of a tweet.
3. `@author`: the screen name of the author of tweet.
4. `text`: text of a tweet.
5. `app`: Application from which a tweet has been published.
6. `id author`: Author identifier. It is an increasing number assigned by Twitter to each user when they log in the system.
7. `followers author`: Number of followers at the moment of the publication.
8. `following author`: Number of users followed at the moment of a publication.
9. `statuses author`: Number of tweets previously published.
10. `location`: location declared in the profile of the user.
11. `url`: if a tweet contains a URL, it is this data. On the contrary it stores a `null` value.
12. `geolocation`: the coordinates that identify the location of the user if the tweet is geo located.
13. `names`: Name provided by the user
14. `bio`: description of the user.
15. `url media`: It contains a url whether the tweet has multimedia information; on the contrary stores to a `null` value.

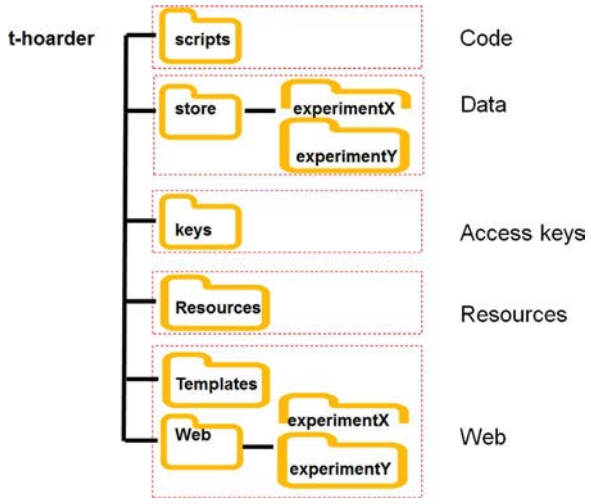

Fig. 2. T-hoarder predefined filesystem structure for code, data, access keys, resources and web.

16. `type media`: type of information multimedia (e.g. photo or video). In case of a non existing media its value is `null`.
17. `lang`: language of the tweet.

Data storage

The platform has been organized in a structure of predefined folders. Folder and file names use predefined prefixes and suffixes to easy the location of the stored information. Fig. 2 shows how code, data, access keys, and the web zone are organized. Code is located in the *scripts* directory, data is placed in the *store* directory and in each experiment's folder, the access keys for the Oauth protocol are stored in the *keys* directory and finally the web prepared data is located in the *templates* and *web* directories.

Streams of each experiment are stored in the *store* directory in packages of files. The files are generated with the following pattern: `streaming experiment x.txt name, (x: 0, n)`. The first file is numbered with zero and when it reaches the size of 100 MB is compressed and a new file with an increasing numbering is created. For text files, the compression is very efficient and the size of the data typically reduces to the third part.

The context of each experiment is stored in a folder under the *store* directory. The context includes either a file with the list of keywords, users or locations to be monitored as processed data.

4.3. Level 2: data processing

The data processing is made of independent form to the capture and storage to avoid premature tweet losses. Therefore, the elaboration of the data is made of periodic form by means of scheduled activities (running on UNIX's `cron`).

Large experiments could generate datasets that would turn out very expensive to process. Nevertheless, the method of storage of streams in packages of files delimited by size makes viable to process them by parts, and to integrate results later. This approach has the following advantages:

1. It is feasible to process compressed data directly without having to uncompress it.
2. It is possible to process the different packages from files in parallel.
3. It is not necessary to process again a package already processed; only the new data from the last iteration has to be processed at each step.

In the *resources* folder, different resources are stored to process the data, for example: tables of names, geo located localities, and dictionaries to classify tweets.

In the *web* directory, there is a directory for each experiment in which the processed data will be stored to be presented/displayed in the graphical web interface.

Processing a package

For each package, the following operations are made:

1. To filter false positives
2. To extract indicators
3. To extract relevance
4. To extract location
5. To generate the package

4.3.1. False positives filtering

Occasionally, false positives happen because the terms used in the queries contain ambiguous words or the words appear in a tweet in a different order than expected. For instance, a query "`metro Madrid`" (Madrid underground) can return false positives because Metro is a TV channel that broadcasts Real Madrid and Atlético de Madrid football matches. False positives are detected using filter patterns stored in a file called `filter.txt. This file`contains a set of words or expres sions that do not correspond to the expected context. The filter is carried out in a script called `tweets select filter.py`, available on Congosto (2015).

4.3.2. Extracting indicator

To observe the evolution of an experiment a series of indicators are computed. Such indicators are exposed later in the time axis. These indicators provide clues on the participation and publication way of the messages:

1. *Number of tweets*: amount of tweets gathered.
2. *Number of RTs*: amount of tweets spread by means of the broad casting mechanisms.
3. *Number of replies*: amount of tweets answering to another tweet.
4. *Number of mentions*: amount of tweets that contain mentions.
5. *Number of unique users*: amount of different users have tweeted.
6. *Number of new users*: amount of users who tweeted for the first time.
7. *Top hashtags*: for each one of the most mentioned hashtags, the amount of times that appears in the tweets.
8. *Top words*: for each one of the most frequent words (they do not consider ending words), the amount of times they appear in the tweets
9. *Top mentioned users*: for each user, the amount of times that appears in the tweets.
10. *Top active user*: for each one of the most active users, the amount of tweets that the user has published.

These indicators are extracted with the `tweets counter.py script`, available on Congosto (2015).

4.3.3. Dealing with relevance

In T Hoarder, relevance is measured by the diffusion of the messages. The messages spread because they catch the attention of other users who choose to give visibility. On Twitter, the propagation of messages is made by means of the retweeting (RT). The RT is a convention created in the beginnings of Twitter by users who want to share tweets with their followers. Since 2009 Twitter included a RT button that did the same but automatically, which facilitated the propagation of messages a lot. People generally spread tweets when they agree. In some sense, it could be interpreted as a positive vote for the message.

T Hoarder discarded using the data of the number of RTs that provides the Streaming API for being dynamic characteristics that vary with time. T Hoarder detects tweets diffusion comparing the similarity

of messages and considering the structure of the RT. Therefore, it detects both manual RTs and automatic RTs. The diffusion of messages is calculated per day and for all the period of tweets capture. In this way it is known the most relevant in each day and the most outstanding of the complete dataset. The data stored for the most spread tweets are:

1. Identifier of tweets
2. Date and hour of a tweet
3. Author of a tweet
4. Text of a tweet
5. RT count

This is how T Hoarder obtains the relevance.

```
for each tweet
  get hour, day
  if first tweet
  then
  last hour hour, last day day
  if tweet in RT global:
  then
  increase RT global[tweet].count
  else
  RT global[tweet].count  1
  if tweet in RT day:
  then:
  increase RT day[tweet].count
  else:
  RT day[tweet].count  1
  if hour > last hour or len(RT day) >15000
  then
  RT global  2000 tweets with more RTs
  RT day  2000 tweets with more RTs
  last hour  hour
  if day > last day
  then
  Order descending RT day by count and write file
  empty RT day
  last day  day
  Order descending RT global by count and write file
```

In T Hoarder, the most widespread tweets are obtained with the `tweets talk.py script`, available on Congosto (2015).

4.3.4. Extracting location

The location of tweets can be obtained into two ways. First, it is the declared location in the profile of the user. This data could be incomplete or it may contain the name of a fictitious location. For this reason, it is not possible to locate all the messages geographically. However, it is possible to locate a high percentage of tweets (between 70% and 60%). The second option is to provide geo localized tweets. In this case the percentage is much smaller (in Spain, 1.5%). T Hoarder has a resource in which all the municipalities of Spain are classified by state, provinces and geographical coordinates (longitude and latitude). With these data, tweets can be geo located on a map according to its declared location and also it is possible to aggregate them by province or state. Support for locations from other countries using the same approach. For the geo localization, simply the coordinates of each tweet are extracted.

Every day, tweets located by user profile and geo located tweets are stored. In both cases the records stored are:

1. Tweet identifier
2. Date and hour of the tweet
3. Author of the tweet
4. Text of the tweet
5. Tweet coordinates

The locations are obtained with the `tweets location.py script`, available on Congosto (2015).

4.3.5. Generating the state of a package

When a package is being processed, it stores in a file denominated `experiment x status.txt`, its state information, containing:

1. *Initial date*: date of the older tweet
2. *Last date*: date of the most recent tweet
3. *State*: state of the process of the package. It takes the values: semi processed or processed.
4. *Last processed tweet*: identifier of last processed tweet. *Length of the package*: packet size when it was processed
5. *Number of tweets*: amount of tweets inside the package
6. *Running time*: total time running in the system

4.3.6. Results integration

Results are calculated on a per day basis, reason why integration is something as simple as the concatenation of results. Also, it is necessary to consider a possible edge effect: as the partition of dataset in packages is made by size (100 K) it can be left a day divided in two different packages.

In the case of tops (words, hashtags, mentioned, and active users), although the top 1000 items are stored, only the top 10 are displayed. The top 10 results are stored in the interchange *web* directory, so that the web server can access them.

This is the process for join results.

```
For each data packet
   Store in memory entity counters by day (one day can be
   in two different packages)
   Store in memory top 1000 of entities (one day can be
   in two different packages)
   Store in memory global RTs
   Store in memory RTs per day (one day can be in two
   different packages)
   Store in memory locations
Generate file of entity counters by day
Generate file of top entities, reducing the top 1000 to
   10
Generate file of global RTs
Generate file of RTs per day
Generate file Locations
```

The data integration is done with the `join results.py`script, available on Congosto (2015).

4.4. Level 3: visualization

To analyze the evolution of the recovered information, web panels were designed to visualize the different types of formats. The technol ogies used for these panels are:

1. Framework bootstrap of HTML, CSS stylesheets and JavaScript. They allow giving structure, style, and interactivity with the different elements of a panel.
2. The Digraphs Library is used for the temporary graphs. It has options to favor interactivity, including facilities to make zoom lens and to invoke functions from a point of the graph to contextualize information.
3. Google Maps for the visualization of maps.

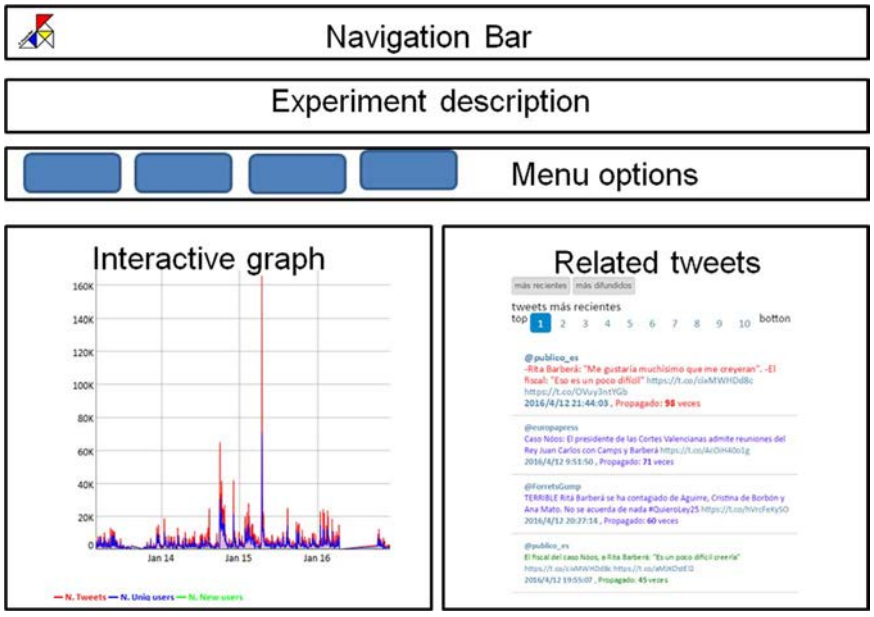

Fig. 3. Different areas on the t-hoarder web panel: navigation bar, experiment description, menu options and interactive graphs.

The T Hoarder panels are built as a puzzle, using the feature < iframe > that embeds different small pieces of HTML generated from a set of templates customized. The panel is created automatically with the shell script: `make panel.py`, available on Congosto (2015).

4.4.1. Template of the main page

It is a generic template with the structure of a web panel in which it is distinguished the description of the experiment and the access to the different time graphs or maps. This template is customized for each experiment by replacing the token @experimento with its specific name.

Fig. 3 shows how information is presented on the web. Four zones have been highlighted: navigation bar, experiment description, menu options and interactive graphs.

1. A navigation bar from which it is possible to access to the main menu of T Hoarder. This bar is common to all panels.
2. Description of the dataset and the entities that are being monitored. This is a frame with textual information.
3. Menu options to select different views from the information: users, type of tweets, more frequent terms and hashtags, more mentioned and more active users, location of messages, and help.
4. Interactive graphical representation of the information selected in the menu. There are two types of graphs: the time graphs which show the evolution of the different indicators, and the geographic ones plotted with a map.

4.4.2. Template for time graphs

This generic template for time graphs is used for several types of entities, for this reason it is instantiated for each of them replacing the tokens @experimento and @data file by their specific values. The scheme of this template is available on Congosto (2015).

The time graphs contain two parts:

1. The interactive graphic area shows the evolution of values of a selected entity. Moving the mouse through the graph the numerical values of the elements of the legend are shown. In order to make zoom lens, left button of the mouse has to be pressed and it crawls. The interface also allows eliminating the zoom lens with a double click. An example of interactive graphs is shown in Fig. 4
2. Most relevant tweets for an entity. Default, display the most recently

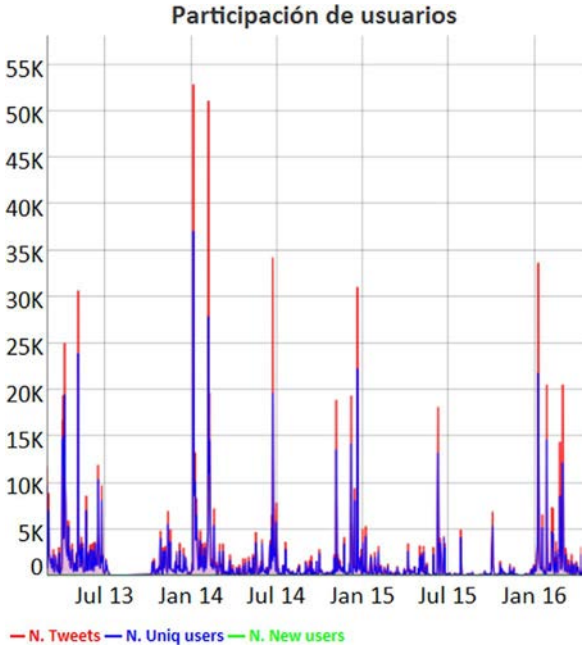

Fig. 4. Example of an interactive graph of the evolution of user participation.

spread. It is possible to show the most propagated for the experi ment pressing in the button “spread”. In order to discover the most popular tweets in a day, it is only necessary to click on the date of the graph on the left. In all cases, the messages are paged in groups of four and it is possible to show a maximum of forty. Fig. 5 shows an example of tweets related with an interactive graph.

4.4.3. Time Panel Template for the location of tweets

There is a HTML template that allows visualizing a map (with Google Maps) in which the frequency of tweets by areas is displayed by means of a heat map. This template is instantiated by replacing the token “`@data file`” by the name of the file with the data. Fig. 6 shows an example of a heat map with geographic areas with more density of tweets. This template is available on Congosto (2015).

Fig. 5. Example of tweets related with an interactive graph.

4.4.4. *Template for geo locating*

It is an HTML template that enables visualizing in a map of Google Maps the geo located tweets. This template is customized by replacing the "`@data file`" token with the name of the data file. Placing the mouse over certain location it is possible to read the messages associated to such location. Fig. 7 shows an example of a map with the geo located tweets of an experiment. This template is available on Congosto (2015).

5. Empirical evaluation

The goal of the experimental studies carried out with T Hoarder is to provide empirical evidence on the performance one may expect from this type of infrastructure. It also shows the benefits stemmed from dividing the data into different blocks, called packages in the T Hoarder jargon. To this end, the evaluation has been carried out on a single machine (Table 4) which computes the different performance of different streams of tweets. This type of infrastructure is enough to determine bands of performance for T Hoarder when using it to process medium size data sets.

5.1. *Benchmark characterization*

The four dataset analyzed (see Table 5) corresponds to Spanish and international microblogging communities. The first, labeled `Diputados`, contains the impact of tweets of the members of the Spanish parliament; it is a large data set (3 years) with a small tweeting frequency (8k tweets per day). The second dataset refers to tweets with an URL that belongs to the main newspaper in Spain: `El Pais`; it can be classified as a medium duration time (> 1 year) and average frequency among tweets (27k tweets per day). The third dataset is international, labeled Ecology, refers to tweets that containing a set of words related with sustainable environment. It has long duration (3 years) with average data rate. Lastly, the forth dataset refers to `Ebola` refereeing to a higher tweet rate (0,5 million tweets daily) in a short time (80 days).

The datasets may also be considered in terms of the packages necessary to process each dataset and the storage space required to store them (see Table 6). In terms of number of packages, `Diputados` requires 26 packages, `El Pais` takes 46, `Ecology` 100, and `Ebola` 169. The number of packets has a direct connection with the number of GB required to store the dataset information, which varies from 2.77 Gb for `Diputados` to the 16 Gb required for `Ebola`.

Each different dataset takes a different amount of time to be processed by the T Hoarder engine. `Diputados` is the smaller dataset and takes 4 h and 13 min. `El Pais` takes 4.5 h of computation time. On the other hand, `Ecology` and `Ebola` are large dataset and take more than half a day to compute the dataset.

5.2. *Package processing times*

One of the main sources of computational overhead is in the time required for processing each package. For the proposed dataset, the amount of time depends on the type of processing carried out and mainly on the amount of tweets that have to be processed.

Fig. 6. Showing the origin of tweets according to the defined location in user profiles in T-Hoarder.

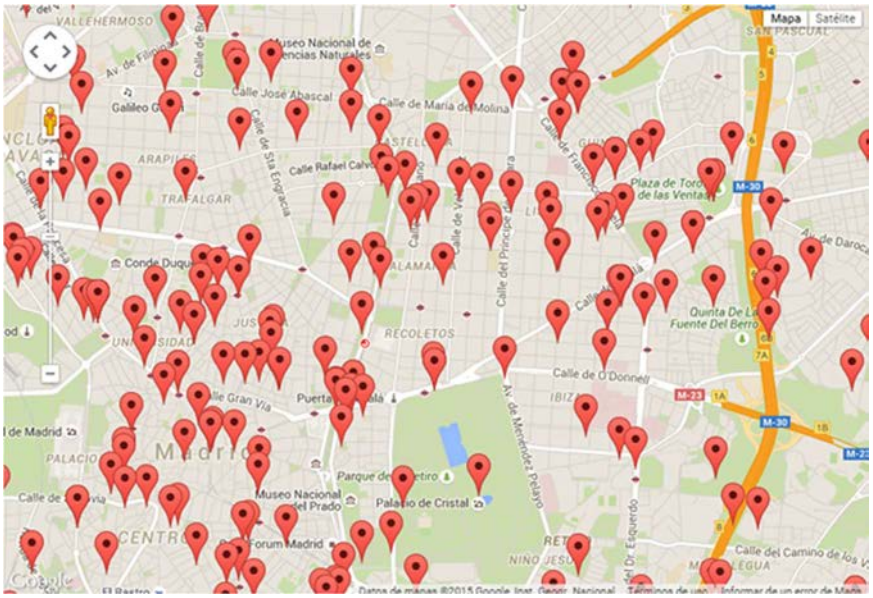

Fig. 7. Geolocated tweets on T-Hoarder.

Table 4
Characterizing the evaluation infrastructure.

Infrastructure	
CPU	2 Ghz (8 cores)
Memory	8 Gigabytes
OS	Ubuntu
T-Hoarder	0.0.9
Benchmarked application	
Templates	T-hoarder end-to-end tool-chain: 1. `tweets_counter.py` 2. `tweets_talk.py` 3. `tweets_location.py`
Number of tweets	10 44 million of tweets
Number of packets	26 169 packets of 100 Mbytes

Since the size of each package is fixed to 100 Mb, the number of tweets of each block depends on the stored format of the tweets. In older datasets less information was stored and as a result, it is included further data at the end of each registry in order that they were backward compatible scripts.

For this reason in datasets like `Diputados` and `Ecology` the number of tweets by package moves between 300,000 to 400,000 whereas in datasets like `El Pais` and `Ebola`, the range of tweets per package moves from 210,000 to 300,000. The variation of the number of tweets in datasets with the same format has dependency with the number of URLs, geolocation information, and size of the user's bios.

Therefore, for the given dataset, the average time for processing a tweet oscillates in 1.3 1.5 ms range. As it is shown in Figs. 8 11, the run time seems to keep some linearity with the number of tweets. In all data, the most expensive calculation is the counters and the less expensive is the location information:

- For the `Diputados` dataset (Fig. 8), the results show the different costs associated to the scripts in charge of processing tweets: `tweets counter.py`, `tweets talk.py`, and `tweets location.py`. In the case of `tweets counter.py`, the time ranges from 4 min and 22 5 min and 40 for larger packages. The `tweets location.py` script moves form 2 min and 20 3 min, for the same data set. Lastly, the `tweets talk.py` script is in the 1 min and 25 s range to the 2 min and 20 s for larger datasets.
- For the `El Pais`dataset, the results are shown in Fig. 9 The results show that the `tweets counter.py` script requires from 2 min and 40 s to 3:50, having a more reduced amount of tweet bundled in each package than the `Diputados` dataset. For the `tweets talk.py` script, the outcome moves from the 50 s to the 1 min and 10 s. Lastly, the code in charge of location takes from 1 min and 26 2 min.
- The third dataset, `Ecology`, the data shows the same performance patterns (see Fig. 10). The `tweets counter.py` script takes from 3 min to 4 min and 40 s. For the same scenario, the script in charge of calculating locations moves from 2 min to 3 min and 10 s. Lastly, the less heavy script is the `tweets talk.py` script, which moves

Table 6
Performance for each dataset in T-Hoarder.

Data set	**Packets**	**Storage space (Gb)**	**Processing time (hh:mm:ss)**
Diputatos	26	2.77	4:13:54
El Pais	49	5.15	4:33:02
Ecology	100	10.28	13:57:14
Ebola	169	16.80	17:58:13

Table 5
Types of datasets evaluated in T-Hoarder.

Data set (LANG)	**Total number of tweets**	**Total Number of days**	**From**	**to**	**Average Tweets/day**
Diputados (ES)	10,100,380	1235	11-12-29	15-05-17	8177.68
El Pais (ES)	11,703,899	433	14-05-07	15-07-14	27,052.19
Ecology (EN)	37,219,566	1130	12-06-09	15-07-14	32,940.71
Ebola (EN)	44,550,169	80	14-08-04	14-10-24	555,241.75

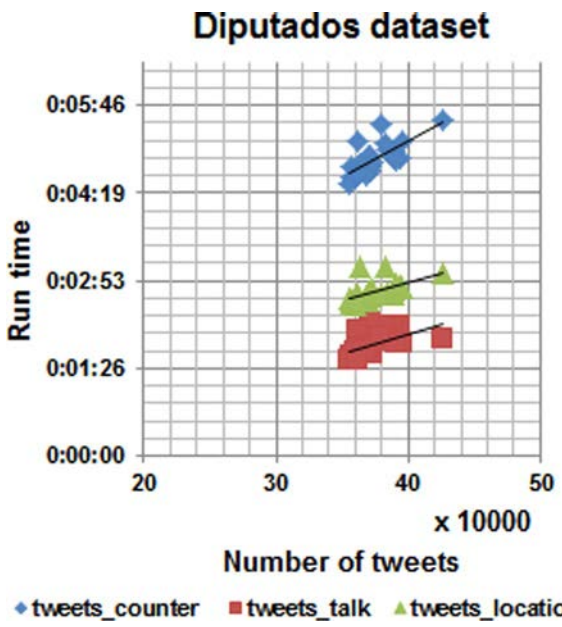

Fig. 8. Processing time per each package in `Diputados` dataset in hh:mm:ss format.

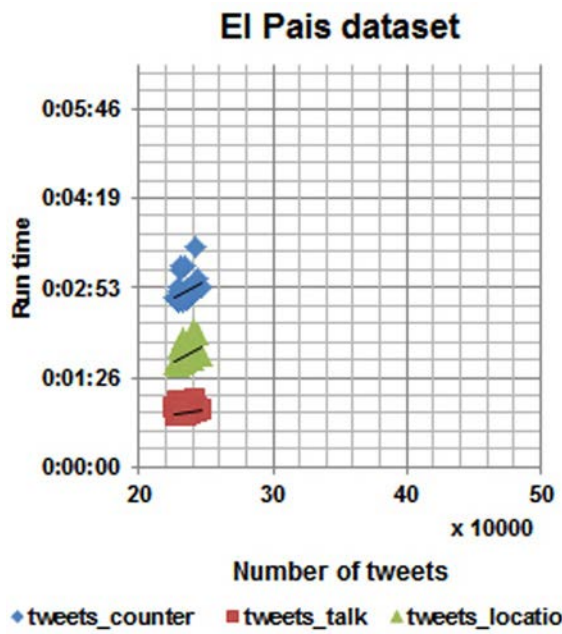

Fig. 9. Processing time per each package in `El_Pais` dataset in hh:mm:ss format.

from 1 min to almost 2 min.

- Lastly, the fourth dataset refers to `Ebola` datasets, shown in Fig. 11. Again, the same patterns shown in the previous dataset are valid for the experiment. The `tweets counter.py` script takes from 3 min to 5 min per packet. The location script moves from 2 min to 3 min and 30 s. Lastly, the `tweets talk.py` script moves from 1 min to almost 2 min.

From these empirical results we can infer:

1. Runtime to process a packet does not depend on the size of the dataset or the number of packets in which it is divided. The four cases studied have different sizes and the execution time per packet are similar
2. The execution time of a package depends on the number of tweets it contains. In Figs. 9 and 11 with fewer tweets per packet the runtimes

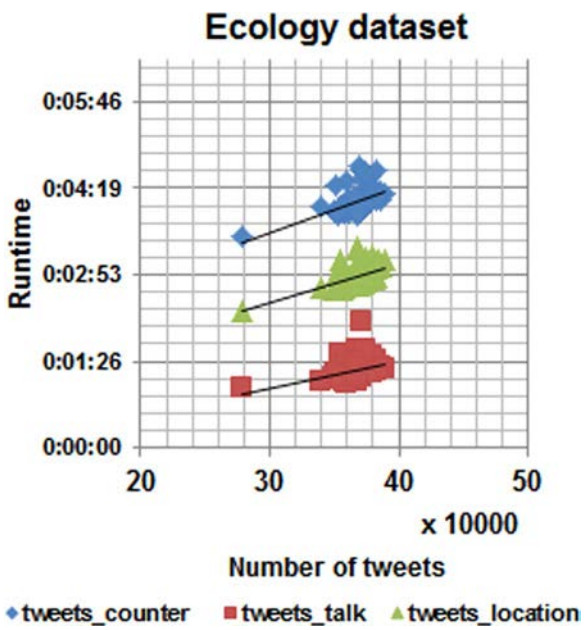

Fig. 10. Processing time per each package in `Ecology` dataset in hh:mm:ss format.

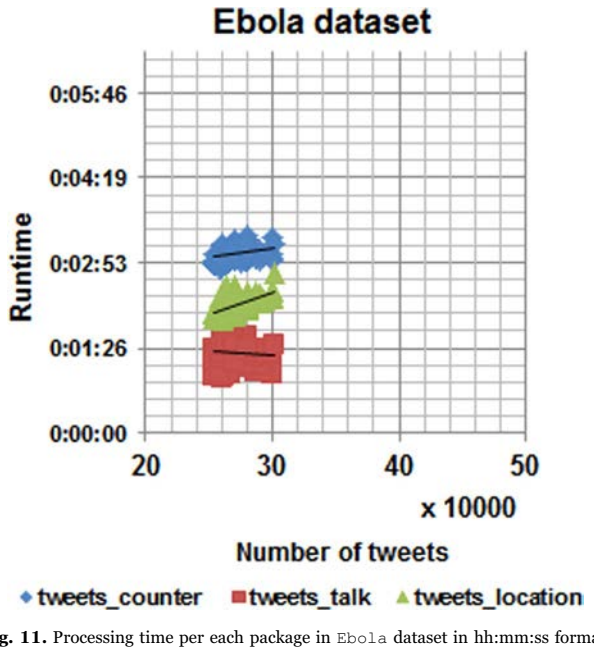

Fig. 11. Processing time per each package in `Ebola` dataset in hh:mm:ss format.

are less than four minutes whereas Figs. 8 and 10 with more tweets per packet exceed this time.

3. The processing system is scalable because, regardless of the size of the dataset, the tweets are processed in small packages in which the runtimes are known: less than six minutes for `tweets counter`, less than three minutes for `tweets location` and less than 2 min for `tweets talk`.

5.3. Joining performance

The division of data comes with extra overheads since the data has to be merged after being partially processed in each package. The integration time for packages depends on the duration of the dataset. This is because the information is processed per days and therefore it increases the number of iterations of the algorithm.

For the datasets described in previous sections, it has been evaluated this overhead. The results (in Fig. 12) show how these times

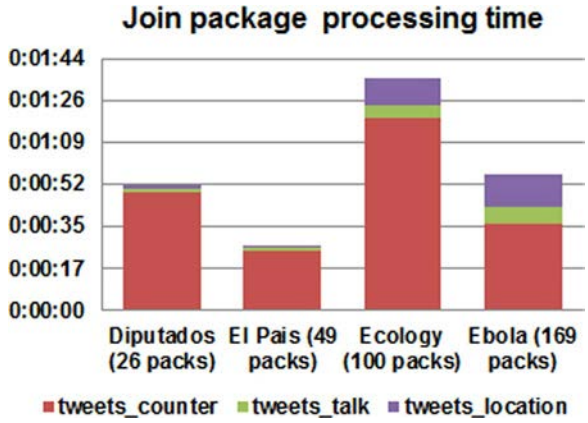

Fig. 12. Processing for joining data in hh:mm:ss format.

are shorter, in comparison with the total time required to process a package of data. In those cases, where the interval of the data is larger, like the case of `Diputados` and `Ecology` datasets, the joining process needs more time in spite of having fewer packages. However, this time is much more reduced than the total time required to process the package. Specifically, the overhead produced by the join of packages with respect to total runtime of packages is a 0.33% in `Diputados` (long period), a 0.16% in `El Pais` (medium duration), a 0.19% in Ecology (long period) and 0.09% in Ebola (low duration).

5.4. Benefits stemmed from splitting packages

Our last experiment analyzes the benefits in splitting tweets in several packages, comparing the performance of this strategy against another equivalent strategy which consists in having a single package.

A monolithic dataset has been evaluated. It consists of 10 packages which contain a large amount of tweets. This dataset is much more reduced than the number of packages introduced in previous sections; our previous experiments refer to 26 169 packets. It has been calculated the execution time for different packets separately, and the global dataset. On these data, it has been looked for the correlation between the number of tweets processed and the three contributors to the overhead: `tweets counter.py`, `tweets talk.py`, and `tweets location.py`.

The results of the evaluation have graphically displayed in Fig. 13. Among the three scripts, `tweets talk.py` is the most affected by the size of the packet and `tweets counter.py` the script that has less influence from the amount of data processed. The three run times for scripts moves from 0:05:31 to 0:00:59. The average time of execution of a package is 0:07: 28. Because once processed a package it is not necessary to process it again, it can measure the improvement in the execution time of the model based on packages to work with a complete dataset.

With a large package the computation time increases with the size of the package. In our experiments, the processing of the dataset is processed in 1:04:43, more than 8x times the executing of the individual packages (see Fig. 14 to establish the empirical evidence). This type of result seems to justify the benefits by exploiting the data into different files in T Hoarder. This numbers are much more relevant when the number of tweets that have to processed increases, but it is also significant with 10 packages.

6. Conclusions and future work

The analysis of micro blogs requires specific tools that help perform global analytics applied to this popular environment. This work contributes with our particular experience in designing and evaluating a tool to perform analytics on micro blogs. With the aim on Twitter, the

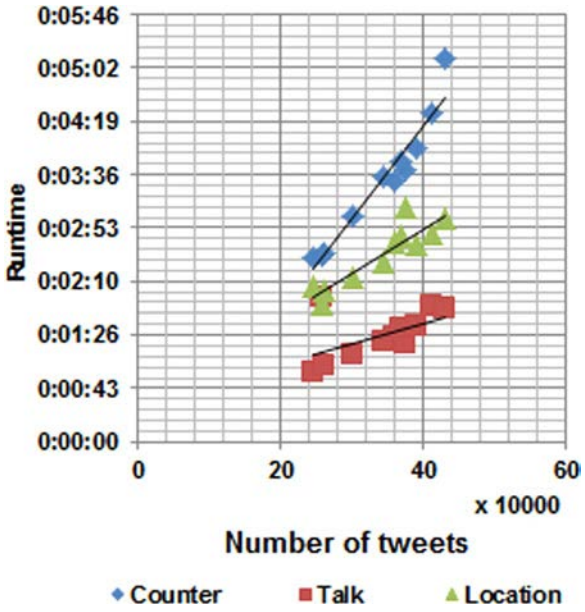

Fig. 13. Correlation among the number of tweets and time taken to processed them. Time measured in hh:mm:ss.

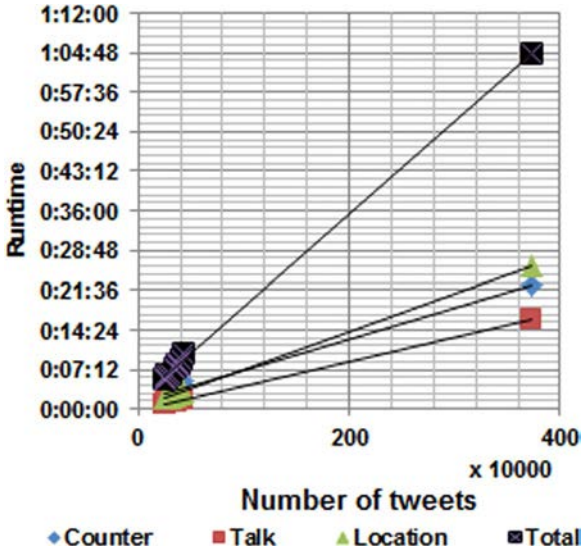

Fig. 14. Correlation among the number of tweets and time taken to process them, with a single packet strategy. Time measured in hh:mm:ss.

article has described a cost effective framework called T Hoarder. The main advantage provided by this framework is the possibility of using an integrated approach to store, process, and to display preprocessed data as an output directly on a navigator. The empirical evaluation carried out with several datasets, revealed the performance delivered by the proposed approach which is able to process millions of tweets in seconds.

Our future work in relationship with T Hoarder includes two main research lines. First, the authors are analyzing the advantages stemmed from the use of different techniques of parallel and distributed computing as architectonic blocks useful to reduce the total computation time of our current engine. The second line refers to the development of an alternative approach by using common off the shelf big data engines (based on Storm (Apache Storm, 2014; Marz and Warren, 2015; Basanta Val et al., 2015; Basanta Val et al., 2016), efficient map reduce strategies (Anjos et al., 2015; Lee et al., 2013), and Hadoop (Zikopoulos and Eaton, 2011) to run stream analytics.

Acknowledgements

This work been partially supported by HERMES SMARTDRIVER (TIN2013 46801 C4 2 R) and AUDACity (TIN2016 77158 C4 1 R).

References

Anjos, J.C., Carrera, I., Kolberg, W., Tibola, A.L., Arantes, L.B., Geyer, C.R., 2015. MRA++: scheduling and data placement on MapReduce for heterogeneous environments. Futur. Gener. Comput. Syst. 42, 22 35.

Apache Storm, 2014. Distributed and Fault-tolerant Real-time Computation. Available: ⟨https://storm.incubator.apache.org⟩.

Asur, S., Huberman, B.A., 2010 Predicting the Future With Social Media, Computing.

Barberá, P., Rivero, G., 2012. Desigualdad en la discusión política en Twitter. Congr. ALICE.

Basanta-Val, P., Fernández-García, N., Wellings, A.J., Audsley, N.C., 2015. Improving the predictability of distributed stream processors, future generation computer systems. SciencieDiret 52, 22 36.

Basanta-Val, P., Audsley, N.C., Wellings, A., Gray, I., Fernandez-Garcia, N., 2016. Architecting time-critical big-data systems. In: IEEE Transactions on Big Data, vol. PP, no.99, pp. 1 1. ⟨http://dx.doi.org/10.1109/TBDATA.2016.2622719⟩.

Black, A., Mascaro, C., Gallagher, M., Goggins, S., 2012. Twitter zombie : architecture for capturing , socially transforming and analyzing the Twittersphere. In: Proceedings of the 17th ACM Int. Conf. Support. Gr. Work, pp. 229 238.

Bollen, J., Mao, H., Zeng, X., 2011 Twitter mood predicts the stock market. Computer (Long. Beach. Calif). pp. 1 8.

Bollen, J., Gonçalves, B., Ruan, G., Mao, H., 2011. Happiness Is Assortative In Online Social Networks. pp. 1 17.

Bošnjak, M., Oliveira, E., Martins, J., Mendes-Rodrigues, E., Sarmento, L., 2012. TwitterEcho a distributed focused crawler to support open research with Twitter data. In: Proceedings of the WWW 2012, 21st Int. Conf. Companion World Wide Web, pp. 1233 1239.

Cha, M., Haddadi, H., Benevenuto, F., Gummadi, P.K., 2010. Measuring user influence in twitter: The million follower fallacy. Icwsm, 10(10-17), 30

Congosto, M., 2015. T-Hoarder source-code. Available: ⟨https://github.com/congosto/t-hoarder⟩.

Congosto, M., 2015. Elecciones Europeas 2014: Viralidad de los mensajes en Twitter. Rev. Redes 26, 23 52.

Congosto, M., Aragón, P., 2012. Análisis De Las Elecciones 20N, ALICE2012.

Congosto, M., 2014. Twitter como fuente para conocer la opinión pública. In: Las nuevas tecnologías audiovisuales frente a los procesos tradicionales de comunicación, C. A. de C./64 CAC, Ed. pp. 117 142.

Conover, M.D., Ratkiewicz, J., Francisco, M., Gonçalves, B., Flammini, A., Menczer, F., 2010. Political polarization on Twitter. Networks, 89 96.

Crockford, D., 2006. The application/json media type for javascript object notation. JSON.

De Domenico, M., Lima, a., Mougel, P., Musolesi, M., 2013. The anatomy of a scientific rumor. Sci. Rep. 3, 2980.

Del Fresno, G., 2012. Twitterstream-to-mongodb. Available: ⟨https://github.com/gdelfresno/twitterstream-to-mongodb⟩.

Dodds, P.S., Harris, K.D., Kloumann, I.M., Bliss, C.A., Danforth, C.M., 2011. Temporal patterns of happiness and information in a global social network: Hedonometrics and Twitter. PloS one, 6(12), e26752

Gayo-Avello, D., 2011. Don't turn social media into another 'Literary Digest'. Commun. ACM 54 (10), 121.

Goonetilleke, O., Sellis, T., Zhang, X., Sathe, S., Goonetilleke, O., Sellis, T., Zhang, X., Sathe, S., 2014. Twitter analytics: a big data management perspective twitter analytics : data management perspective. ACM SIGKDD Explor. Newsl., vol. 16, no. 1, pp. 11 20.

Gudivada, V.N., Baeza-Yates, R.A., Raghavan, V.V., 2015. Big Data: Promises and Problems. IEEE Computer, 48(3), pp. 20-23

Honey, C., Herring, S.C., 2009. Beyond microblogging: conversation and collaboration via Twitter. In: Proceedings of the 42nd Hawaii International Conference. System Sciences 2009. HICSS'09. pp. 1-10. IEEE, pp. 1 10.

Huberman, B.A., Romero, D.M., Wu, F., 2008. Social networks that matter: Twitter under the microscope. Computing, 1 9.

Java, A., Song, X., Finin, T., Tseng, B., 2007, August. Why we twitter: understanding microblogging usage and communities. In Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis pp. 56-65. ACM

Kranjc, J., 2014. Twitter-Tap. Available: ⟨https://github.com/janezkranjc/twitter-tap⟩.

Lee, D., Kim, J.S., Maeng, S., 2014. Large-scale incremental processing with MapReduce. Futur. Gener. Comput. Syst. 36, 66 79.

Leiba, B., 2012. Oauth web authorization protocol. IEEE Internet Comput. 1, 74 77.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H., 2011. Big data: The next frontier for innovation, competition, and productivity

Marz, N., Warren, J., 2015. Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co..

OBrien, J., 2011. TwapperKeeper. Available: ⟨https://github.com/540co/yourTwapperKeeper⟩.

Peña-lópez, I., Congosto, M., Aragón, P., 2014. Journal of Spanish cultural studies Spanish Indignados and the evolution of the 15M movement on Twitter: towards networked para-institutions. J. Spanish Cult. Stud., 37 41.

Preotiuc-Pietro, D., Samangooei, S., Cohn, T., Gibbins, N., Niranjan, M., 2012, June. Trendminer: An architecture for real time analysis of social media text. In Proceedings of the workshop on real-time analysis and mining of social streams

Stavrakas, Y., Plachouras, V., 2012. A platform for supporting data analytics on twitter: challenges and objectives. In: CEUR Workshop Proceedings, vol. 895, no. Ict 270239, pp. 1 6.

Zikopoulos, P., Eaton, C., 2011. Understanding Big Data: Analytics For Enterprise Class Hadoop And Streaming Data. McGraw-Hill Osborne Media.