Twitrends: A Real Time Trending Topics Detection System for Twitter Social Network

Cosmina Ivan
Department of Computer Science
Technical University of Cluj-Napoca
Cluj County, Romania

Andrei Moldovan
Department of Computer Science
Technical University of Cluj-Napoca
Cluj County, Romania

ABSTRACT
Big Data processing applications have become popular in the last few years. One of the main reasons is that the data generated by current systems and applications is more complex, have a higher speed and its volume increases exponentially. Another reason would be that the traditional methods for data processing and storage are obsolete and the new tools and frameworks brought a lot of advantages. Various social networks need to process big volumes of data, and users take into consideration the speed and quality of the process. We propose an initial approach for processing data from Twitter social network, in a system which allows a real-time classification of tweets based on topics and user location. With this approach we argue that in a dynamic world, were data increases exponentially and the processing needs to be very fast, the proposed system is capable to determine trending topics in real time.

Keywords
Twitter; trending topics; real-time; geolocation; Big Data

1. INTRODUCTION
Over the last few years, both the volume of data that needs processing and storage, and the variety of sources that provide the data have increased exponentially. This phenomenon is due to technological evolution and was embraced by well-known companies like Amazon, Google, Twitter, etc., under the name Big Data, due to the fact that the size and complexity of the data became difficult, even impossible to manage using traditional data processing systems. The speed at which data is created and need to be processed and stored, continuously increase in the last years. The most popular tools and frameworks for real time stream data processing in the Big-Data era will be presented. The paper presents the Big Data concept by defining it from different perspectives, the various processing models followed by an analysis of the currently industrial and research frameworks. The purpose of this paper is to present the necessary concepts regarding the different Big Data processing models, to propose and implement a system which make use of two new stream processing frameworks from Apache -Storm and Heron, in order to determine the most discussed subjects from the Twitter social network and their classification at the geolocation level. For obtaining relevant outcomes, the needs to be done in real time where the allowed latency is at the seconds level, using as input data, a real data stream from Twitter social network. In this respect, another objective was to propose and implement a classification method through which to obtain results as accurately as possible by selecting relevant information from the vast amounts of data produced by the social network.

The processing model will involve the following steps: 1) reading the data stream, 2) processing the data in real time, 3) providing real-time data output and analytics. The frameworks Apache Storm and Heron represent two open-source, scalable systems developed for deployment in clusters, used recently in real-time big-data processing and analysis. A cluster represents an independent group of servers that collaborate as a unified system in order to offer greater availability and scalability. Horizontal scalability is obtained by allocating more nodes in the cluster, so the processing is based on multiple hardware resources.

This paper is organized as follows: Section 2 encompasses the research we have done on existing stream processing conceptual frameworks and implementations, with focus on Apache Storm and Heron. Section 3 present the proposed system design and implementation, in terms of the architectural topology proposed, functional components description, the execution model, and implementation details. In Section 4, a validation of the system was done and Section 5, contains conclusions and future developments.

2. BIBLIOGRAPHIC RESEARCH
The concept of Big Data is relatively new, which became popular in the last decade, and its definition is relatively complex due to the properties that characterizes the immense volume of data which are in a continuous growth.

Despite the increased interest, a universally accepted definition for this concept is not established yet. According to MIT Technology Review (2013): "a data set, which is defined as high today, will be with a great certitude considered small in the near future" [1]. The size of the data sets is often reported to currently existing technology for processing it. In the absence of a well-established definition, representative players on the market have contributed to the Big Data phenomenon with their approach and implementation, for example Oracle says that "Big Data is derived significantly from business traditionally based on relational databases, correlated with new sources of unstructured data", Intel appreciate as " Big Data Opportunities occur in organizations that generate an average of 300 terabytes of data per week " [2].

The most popular way to characterize Big Data is based on the 3 V's: volume, variety and velocity. Volume refers to the size of the data, and has increased exponentially over the last few years and this trend continues. Velocity describes the frequency at which the data are generated and received. Variety is one of the most important characteristics, as it describes the diversity in content and representation. A forth V was added recently, namely veridicity which refers to reliability, accuracy and precision all together [3].

Due to the increasing volume of data that needs to be created, processed and analyzed continuously, the traditional processing and storage methods and technologies became obsolete, and new models and their counterpart frameworks as

Storm, Spark, Flink or Heron [4] are used for data streaming processing.

2.1 Batch processing using MapReduce model

Majority of batch processing systems were based on the programming paradigm known as MapReduce processing model, which was introduced by Google and first implemented in Apache Hadoop [4], a framework which integrates various batch processing technologies. MapReduce is based on dividing the processing in two major stages: Map and Reduce, each receiving as input data a key-value pair, whose type is established by the programmer and also return a key-value pair as a result. Batch processing based on the MapReduce model presume reading a set of data of the dimension of the batch. When the batch is full (or when the execution of some processing is forced by the planner) data will be submitted for processing, obtaining a "framed" model of execution, either in terms of time (at some imposed intervals), or in terms of the volume of data (imposed by the dimension of the batch). Although, Hadoop is a good framework for what it was developed, the model has some limitations as for example the possibility that the problem and data cannot be transposed in key-value pairs, or the specific delay of producing the output data determined by the dimension of the data that has to be processed and the computing power of the system.

2.2 Stream processing

As an alternative, the recent data stream processing, also known as processing in real time, involves a continuous processing of the input data. In this context, real-time can be analyzed from two points of view: the data and the final user point of view. From the data perspective, the term real time refers the data processing as soon they are received, so the results obtained after the analysis will be always current. From the perspective of the final user [5], the definition of the big-data concept correlated with real time, will be made based on the necessary time for the system to respond to an interrogation, so that the user could have immediately the response to the request. From this point of view the notion of real time can be compared to a call to a REST service or any other call of RPC type.

2.3 Lambda architecture

The Lambda architectural model [6] requires the integration of two levels: a fast processing level, based on streams and a massive processing system, using the batch model which keeps the advantages of the two methods, and the disadvantage of the latency is solved by the layer that processes the data in real time

The Lambda architecture includes three levels:

1. *The batch level* which manages the master data type and calculates in advance the batch views. A possible implementation is one using Apache Hadoop or an OLAP system (online analytical processing) like Vertica or Netezza.
2. *Speed level*: realizes an analysis in real time of a subset from the total of collected data, being used to offer immediate access to the most current results,
3. *Service level*: which represents a caching mechanism of the obtained results from the analysis made by the batch level.

Therefore, Lambda defines a big-data architecture which allows predefined and arbitrary interrogations. He includes the advantages of the MapReduce model for simultaneous processing of a big volume of data, and the latency introduced by that is resolved through de service level in real time. It can be observed that both models: the batch model and the stream processing, bring advantages and disadvantages. Batch processing allows the analysis of a big volume of data, but introduce latency in obtaining results, problem which was solved by the stream processing at the price that data is available only for a definite period of time. The hybrid lambda architecture offers a combined approach, in trying to overcome all these disadvantages, but at a price of a significant complexity.

2.4 New stream processing frameworks

The main reason that led to the development of a great number of frameworks for stream processing was mainly the inherent limitations of the batch processing, namely the high latency introduced. Real-time processing requires the processing of a continuous flow of data, so that the results obtained can be available with a minimum latency accessible to the final user. The well-known systems that offer stream processing data are Apache Storm, Heron, Apache, Splunk-Streaming, Spark and many others [8]. As we will describe in more details the first two as being of interest, the rest will be only shortly presented.

Splunk is a platform that can be used for real –time analysis for machine generated big data, and for processing structured and unstructured files. Splunk captures, indexes and correlates real-time data in a searchable repository from which it can generate graphs, reports, alerts, dashboards and visualization. Spark streaming is an extension of the base framework Spark that offers a high level API in Java, Python, R for the Big Data processing.

Apache Storm [9,10] is a distributed system, open-source of Big Data type. Storm includes a series of features like horizontal scalability, tolerance to failure, ensuring data processing and support for different languages.

Basic concepts in Storm are topology, tuple, stream, bolt and spout and their functional definition will be presented.

Topology: represents the top level abstractization which is used for describing the workflow of a Storm application. Contains as elements spouts, bolts and streams of tuples.

Tuple: an identifiable data structure which contain an ordered list of values

Stream: basic abstraction from Storm, which represents an infinite sequence of tuple, which are created and processed in parallel.

Bolt: reads the tuples of one or more streams and processes them

S*pout*: defines a source of tuples in Storm, reads an external source and sends data in the topology.

A Storm cluster is dedicated to the execution of a topology and can be compared to a Hadoop cluster. While Hadoop executes jobs of type MapReduce, Storm runs topologies the major difference being represented by the fact that the execution of a Hadoop job is finite and the topology is executed infinitely or until is eliminated. A Storm topology is illustrated in Figure 1**.**

The Storm architecture is based on master-slave pattern. The coordination of the master processes and the slave processes are made through a third-party component called ZooKeeper. The master node called Nimbus is responsible for the task

distribution in the cluster and their assignment to the processing nodes similarly to an operating system scheduler, and also used for the system monitoring.

Each node involved in processing, contains a supervisor and every supervisor has control over one or more logical workers in that node, their coordination being resolved by the Zookeeper component.

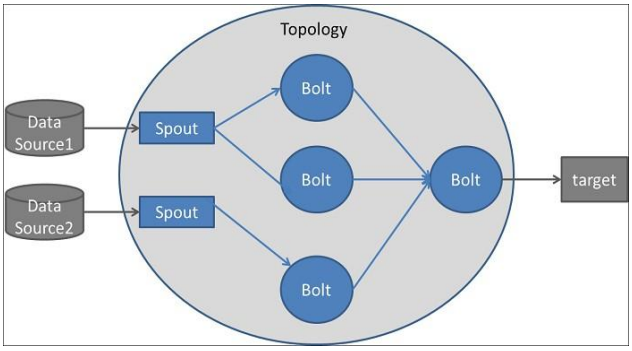

Fig 1. Topology in Storm

Apache Heron [10,11] Along with the tremendous increase in data produced especially in the social apps domain, many limitations of the Strom framework where visible, like the overuse of functionally for the Nimbus component. Form the efficiency point of view were identified the following drawbacks: suboptimal replays due to the failure of a tuple in a tree and long cycles of garbage collection determined by the topologies which consume too much RAM memory.

Due to the increase volume of processed data and also determined by the large diversity of data, improvements brought by Heron over Apache Storm, are highlighted in the architectural model where the final user submits the tasks to the planner, tasks which are executed as a job in more than a single container, one of them running the so called Topology Master (equivalent to Nimbus) and being responsible for the topology management.

Each container runs a Stream Manager which executes the routing of the data, a manager for every metrics which collects and reports different metrics and a number of processes called Heron instances, which run spouts and bolts defined by the user and specific to the application. The metadata of a topology which contain the physical plane and the execution details are kept in the Zookeeper. This new architecture was designed with more support for flexibility, versatility and dynamism.

After a set of benchmarks made by the architects from Twitter to evaluate the system, Heron proves superior performance over Storm, both in terms of execution speed and simplicity in troubleshooting and detecting the failed or lazy component from the topology.

3. DESIGN AND IMPLEMENTATION

The solution proposed is an application implemented using the Apache Storm framework and written entirely in Java. The application creates a real time analysis of all tweets send on the Twitter social network which can be used to determine the so called *trending topics*, a term associated to the most discussed subjects from the social network, at a specific time. The system initially, was implemented in the Apache Storm framework and then migrated to the new real-time processing framework proposed by Twitter, named Heron.

This section will present the proposed method for the topics classification, trending topics, having a hashtag based algorithm as main classification mechanism. For the implementation, running, testing and integration of the application, the following tools and frameworks were adopted: Eclipse Mars as IDE, Apache Maven for the integration management and the compilation Twitter4J and Bing Maps API as libraries used for the real-time analysis of tweets and their classification, JUnit for testing the framework, the log4j libraries for the logging files, and finally for running in a simulated and real cluster we used Apache Storm and Heron.

3.1 Input data format

The input data used by the application to determine the most discussed topics from the Twitter social network, represent the read tweets accessed in the application using Twitter Stream API.

The first necessary step in order to access the data is the authentication using a specific key created as a combination of: a consumer key, a consumer secret, an access token and the secret access token.

Once authenticated, the developer has access to the real public streams, which are divided in three categories: POST statuses / filter, GET statuses / sample and GET statuses. The first endpoint returns all the tweets that match to at least one filter as a parameter. It is possible to specify multiple number of filters, but at least one needs to be present. The endpoint "GET statuses / sample" returns a number of public status randomly chosen and is the chosen one for implementing the classification of the tweets. The application takes into consideration all the tweets read and does not need an initial filtering, but as the data advance in the topology, the ones that are irrelevant were removed.

The most relevant fields from the content stream, used by the system are:

Text: contains the text of a tweet posted by a user of the social network

GeoLocation: represents the coordinates of longitude and latitude of the user that posted the tweet. The coordinates are represented under the form of an object of type GeoLocation.

Place: which represents specific locations and the corresponding geo-coordinates. They can be attached to a tweet, but the fields are optional. The objects of type Place contain a series of attributes used for a more detailed description of the location. With respect to the system, the most relevant attributes are the name, for example: name="Paris", country="France" and the type of the location.

HashtagEntities: represent a list of all mentioned hashtags in the current tweet. The values of type hashtag are extracted from the text of the tweet and are represented as a character string, without the specific "#" symbol, like in the initial message

3.2 Determining the trending topics

A method for determining the most discussed subjects, also named as "trending topics" became a tough challenge with the explosion of the number of users in different social networks and of importance for various fields as online marketing, social interest surveys, product promoting. The proposed method and associated metrics are used in order to compute a set of most discussed subjects based on [12].

In order to determine the set, a basic method implies the counting of the most mentioned terms in the poster tweets in the Twitter social network. In the first step are removed all the words which are presumed to be irrelevant, after that is counted the apparitions of every word, where the word frequency is related to a series of subject categories. The major problem encountered was that when applying this method, a lot of tweets contain words relevant to more than one category are present in the selection. Moreover, the classification is made at the word level and not at the semantic level.

In TwiTrends, the proposed method for classification used to determine the trending topics over an interval of time is based on extracting and counting the hashtags from a tweet. A hashtag is represented by a word or more words concatenated, which start with the symbol "#". The hashtag symbol is used in a social relationship like Twitter to identify a message that belongs to a certain topic. So, a topic is represented through a set of hashtags associated to it. The method is known in the domain of data analysis for social network as "Trending Hashtags" method. Suppose two subjects A and B, the fact that A is more popular than B is equivalent to the fact that the number of mentions of the subject A is greater that the number of mentions of the subject B. This relationship can be described in the following formula presented in Figure 2**.**

$$popularity(A) \geq popularity(B) \Leftrightarrow no\_mentions(A) \geq no\_mentions(B)$$

Fig 2. The relationship between the number of mentions and their popularity

Applying the semantics of a hashtag on which the method of "Trending Hashtags" is based, we can deduce that the number of mentions of a subject it actually represents the number of hashtags associated to it, the hashtag identifying the topic to which it belongs. So, the identification of a subject becomes equivalent with the identification of the number of hashtag that represent that subject. Also, for a more accurate analysis we have to take into consideration a new dimension: the time. The popularity of a subject is related to a period of time, so this is defined on a bounded interval t1 – t0, denoted with Δt. The time t1 is the current time, and t0 represents a moment from the near past. As an example, we consider the term "big-data" as being a trending topic at the time moment t1, if the number of hashtags that are associated to that subject identified by a predetermined time interval of the order of hours is of certain size. Therefore, using the notation, in which A and B represent two different subjects and, in addition, #A represents a hashtag which determines the subject A, and #B is associated to the subject B, the formula can be rewritten in the following manner, presented in Figure 3 .

$$popularity(A) \geq popularity(B) \Leftrightarrow \frac{\sum_{t0}^{t1} Count(hashtagA)}{\Delta t} \geq \frac{\sum_{t0}^{t1} Count(hashtagB)}{\Delta t}$$

Fig 3. The popularity of a subject based on the number of hashtags in a time interval

3.3 The TwiTrends topology

Implementing a real-time processing system in Apache Storm or Heron assumes the design of an architectural topology, which will be implemented and then run on a cluster. Thus, the implementation of the TwiTrends begins with the design and implementation of the topology architecture at the level of components, also it involves the definition of the relationships between topology elements named spouts and bolts through specific data streams. At each bolt we must specify the potential parallelism and the description of the mapping model of the tuples for every stream at every instance. We will describe the topology components for TwiTrends, which represents the basic architecture of the system used to determine the most discussed subject from the Twitter social network. The topology can be represented as a directed acyclic graph (DAG), consisting from a single spout node, the one that issues the tweets accessing the Twitter4J library, a set of bolts type nodes used for processing, filtering and forwarding data and the streams of tuples which link together all components.

The TwiTrends topology is a hierarchic one, composed of a top-level component from which diverge the set of elements and connections between them. Also, the architecture describes the interactions between these components, defined based on the streams of tuples through which they communicate. The main components of the TwiTrends architecture are:

TwiTrendTolology: Represents the top-level component of the topology. Contains the spout used for issuing the tweets and the eight bolts used for processing them

TweetSpout: Represents a component used for issuing the tweets in the TwiTrends topology. This was the only spout used, taking into consideration only the reading and the forwarding of a single stream

TweetFilterBolt: Reads the tweets issued by the *TweetSpout* and executes the filtering. Only tweets that contain coded messages using the standard Unicode.

ParseTweetBolt: Processes the filtered tweets issued as tuples by the component TweetFilterBolt. Taking into consideration that the tuple is filtered, at this level we have the guarantee that each tweet contains at least one hashtag

CountHashtagBolt: Takes the tweets that are parsed through the component *ParseTweetBolt* and counts each hashtag. *CountHashtagBolt* uses a dispersion table as a buffer to map each hashtag to its counter. The table is updated at each read tuple, based on those values. The component issues an output stream which contains a corresponding pair as an entry in the dispersion table based on the hashtag and the counter associated to it

IntermediateRankerBolt: Represents a generic intermediate component for determining the most N mentioned hashtags

TotalRankerBolt: Makes a total ranking of all the counted hashtags. It uses an intermediate classification model, followed by the aggregation of the results that facilitate the parallel execution.

broker, which represents a local storage space used as a database and which can be accessed in a publish/subscribe manner. *RedisBolt* makes an aggregation of the issued streams by the components *TotalRankerBolt* and the *CountLocationHashtagBolt.*

The implementation of the topology implies the translation in Java code. Every component (spout or bolt) correspond to a

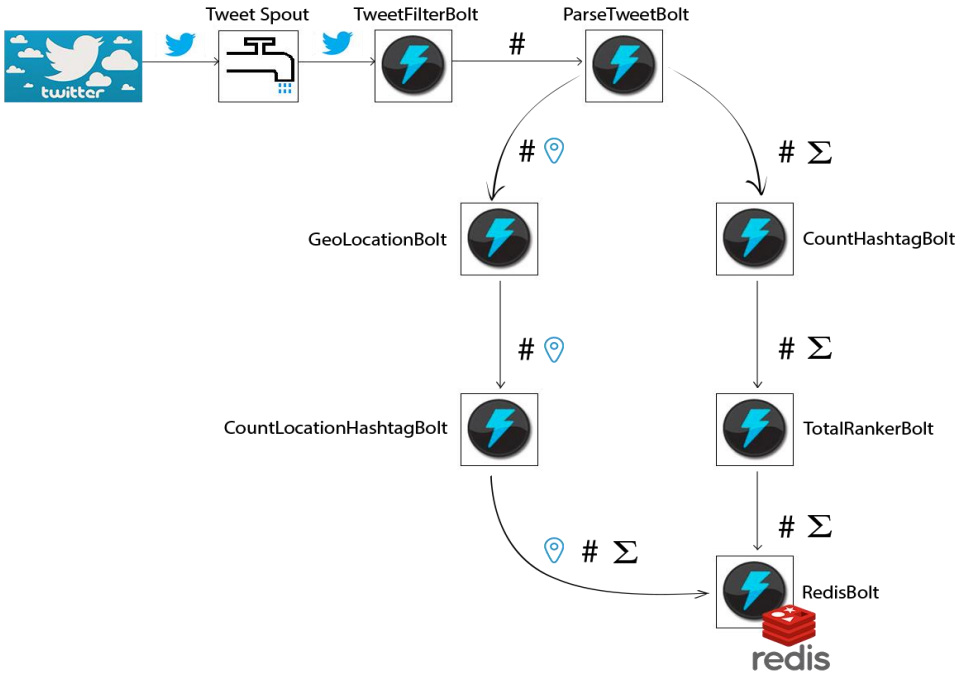

Fig 4. The TwiTrends topology

GeoLocationBolt: It takes the hashtag issued by the *ParseTweetBolt*, with the location of the tweet. Taking into consideration that the location is encoded through coordinates of longitude and latitude, the transformation to a concrete location is necessary.

CountLocationHashtagBolt: Presents a functionality similar to the component *CountHashtagBolt* but introduces a second dimension with regards to the counting. The counting is not reported to the text hashtag, but to the geolocation from which it was issued. In this case, it is not possible the extraction of the location of the tweet represented by the input tuple, the bolt uses a variable of type sentinel named "UNKNOW LOCATION" as a key in the dispersion table, this being issued by the responsible bolt for the calculation of the location. The structure of the buffer is represented through a key-value relationship, in which the key is composed from two fields: the location and the text of the hashtag

RedisBolt: Is the final bolt of the TwiTrend topology and represents a component that groups the processing results at the global level. The component saves generically the most N discussed topics identified based on the hashtag and processed in the topology. Also, this bolt contains the counting of the hashtags at the geolocation level, obtained through the stream issued by the *CountLocationHashtagBol*t. Being the final component, *RedisBolt* needs to allow access to the data stored external to the application. To facilitate this functionality, the bolt will publish all the data contained in a Redis message

class, and the level of parallelism, the streams through which the components communicate and the type of grouping are specified at class level which implements the topology. As a basic rule, bolt which needs a greater processing time needs to be processed at a greater level of parallelism, in order to maintain a flow of continuous data. The components that only process and forward data in the topology use a grouping of type shuffle, or random and they don't save any intermediate data. To reduce the regrouping time of the issued stream, usage of grouping at the hashtag level or the location, avoid the presence of the same keys in the maps of different instances. So, two hashtags or identical geolocation contained in different tuples will be processed by the same bolt instance. All the bolts in the topology are in the package *com.twitrends.bolt.* Sub-package *com.twitrends.bolt.apache* contain the classes taken from the open-source project "storm starter". The spout used is in the package *com.twitrends.spout.* TwiTrends topology accesses a set of interfaces from the package *com.twitrends.util*, to obtain the constants used in the topology, the identifiers of the components it instantiates, name of the fields of each tuple and the necessary values to login in the Twitter Stream API.

In the proposed topology, there exist processing nodes that cannot be executed in parallel. For example, *TotalRankerBolt* which is represented through a single instance, it issues only one stream which represents the result of the aggregation of the obtained tuples from the *IntermediateRanklisherBolt*,

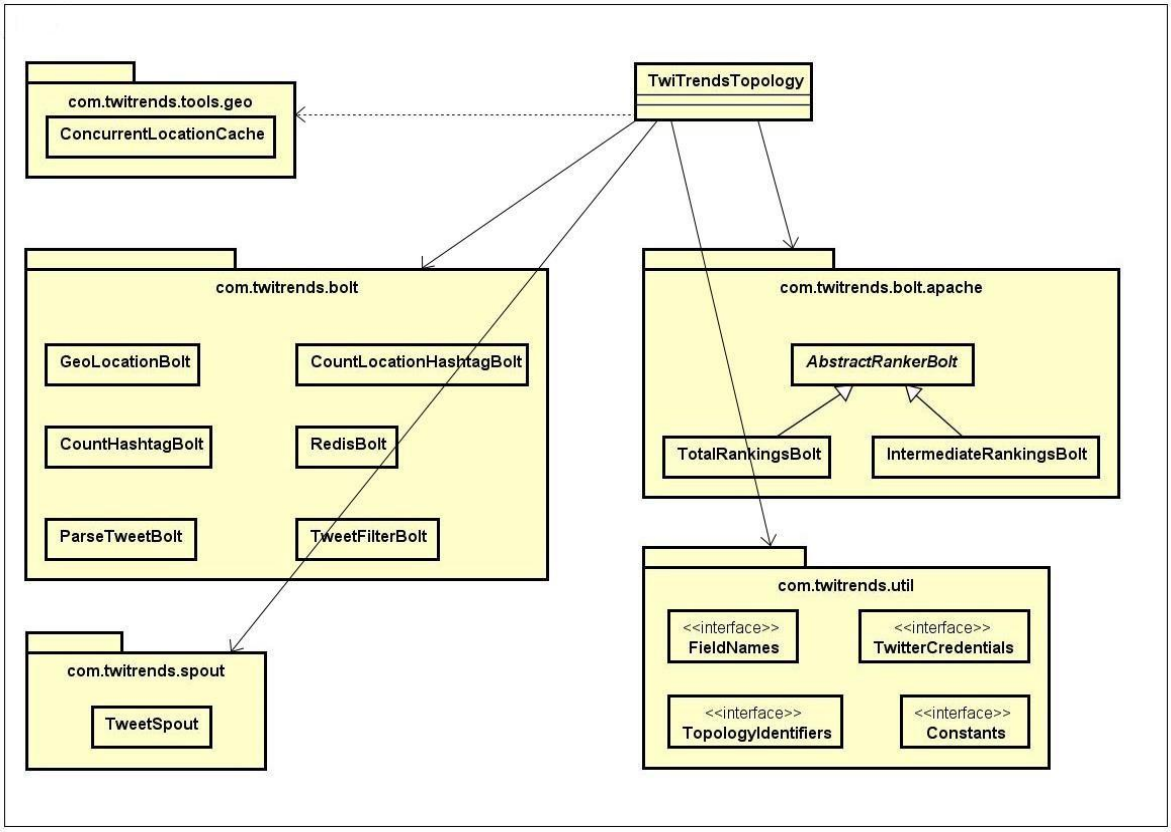

Fig 5. Simplified class diagram of the TwiTrends system

similar to *RedisBolt* which is a data collecting bolt at the global level, and was made available to the external environment.

The two frameworks, Apache Storm and Heron offer a series of implementation for the different grouping semantics of the streams to increase complementary the processing speed. The grouping methods used in the Twitter topology are the following:

Shuffle grouping: The tuples are distributed randomly by the tasks of the elements of type bolt. The grouping ensures that the distribution is uniform, so that each task will receive an equal number of tuples
Fields grouping: The stream of tuples is partitioned based on the fields specified in the grouping. For example, if the stream is grouped after the field hashtag, the tuples that contain the same hashtag will be processed in the same bolt instance, but the tuples that contain a different hashtag can be processed by other instances
Global grouping: The entire system of data is transmitted to a single take corresponding to an instance of a bolt. In the case that there is more than one instance, is chosen the one with the smallest identifier.
The components that only process and forward data in the topology use the shuffle or random grouping. They only save intermediate data, so they do not depend on the level of value that are contained in the stream tuple which they process. Bolts like *CountHashtagBolt* or *CountLocationHashtagBolt* save the value of the hashtag of the geolocation for a processed tweet. In order to reduce the regrouping time of the issued streams, usage of a grouping at the hashtag level or the location, avoid the presence of the same keys in maps from different instances.

3.4 Reading, filtering and parsing of a tweet

The reading, filtering and the parsing of a tweet is made through three main classes: *TweetBolt*, *TweetFilterBolt* and the *ParseTweetBolt*.

In order to implement a component of type spout, Apache Storm imposes the extension of a class type spout implemented in the framework and the overwriting its method to be implemented in a personalized functionality. The implementation of a bolt is similar to one of a spout, but the class needs to be extended to one corresponding to the framework Apache Storm, what differs in respect to the overwriting of the methods. The component of the topology that read the real tweets accessing the Twitter Stream API trough the library Twitter4J is called *TweetSpout*. The Spout authenthicates to have access to the data stream, after it reads one tweet and it saves in a structure of type *LinkBlockingQueue* which is used as a buffer, and after that, each tweet from the queue is issued in the topology.

Top N Hashtags: The determination of "trending topics" is based on a method which identify the hashtags that determine that topic. So, for determining the most discussed subjects, TwiTrends validates and parses a tweet, after it counts each apparition of every hashtag. Based on this, it counters a classification based on the most mentioned N hashtags, the value of N being a predefined constant in the application. In determining the most mentioned N hashtags three bolts of the TwiTrend topology are involved: *CountHashtagBolt*, *IntermediateRankerBolt* and *TotalRankerBolt*. [13]

Geolocation module. In the stream analysis of the tweets, TwiTrends realizes also a classification at the geolocation level. The necessary information is obtained from the object of type Status, which represent a tweet. This contain the coordinates of longitude and latitude of the location from where it was posted. Based on this coordinates, TwiTrends computes the name of the city and the country where the hashtag or hashtags contained in that message. This conversion is made using Bing Maps API, which returns the details of the location based on the specified coordinates in the request sent. To obtain a better performance, TwiTrends uses a mechanism of caching all location already computed, avoiding the access to Bings Maps API for the identical locations, in order to a much faster execution.

The classification of a hashtag at the geolocation level represent in TwiTrends the most expensive operation from the execution time point of view. In the case that every location would be computed using Bing Maps API, the latency will be increased substantially. Using a cache memory to store the values already computed, will improve the execution time considerably. Moreover, the initialization of the memory before the actual execution of the topology, bring an increase in performance for the cases mentioned before.

The model of execution of the TwiTrends topology can be resumed to the following steps for a tweet: read tweet, verify if is valid, parse it, process it and store the results. Finally, the results obtained after the analysis are aggregated and classified according to the model "Top N hashtags". For this comparative performance analysis, 200 different pairs (longitude and latitude) for computation of the location were realized, choosing 1000 times, a random pair from this set.

3.5 The execution model

The execution model of the topology can be resumed to a set of necessary steps to process a tweet. In the first step, the tweet is read by the application and issued further in the topology after validation. A *valid tweet* represents a tweet whose message is encoded according to the validation realized by the TwiTrends topology. A tweet whose content does not correspond to that criteria is ignored by the topology, and the processing is finished. A valid tweet is issued further under the form of a tuple in the stream and arrives to the parsing state. The parsing of a tweet involves the extraction of all tweets from the message and the information relating to the location of the issued tweet

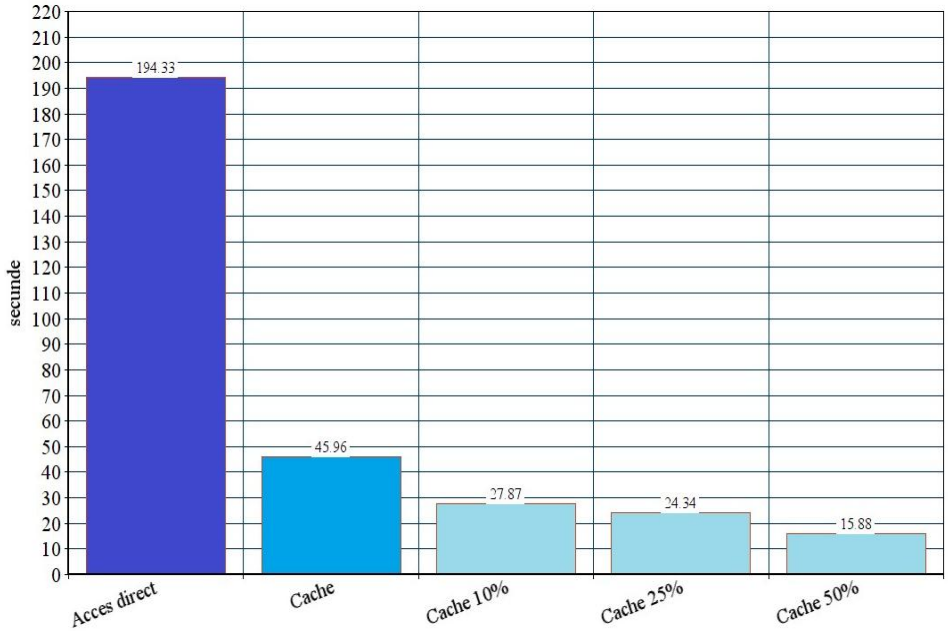

Fig 6. Performance for the calculation of the twit-geolocation

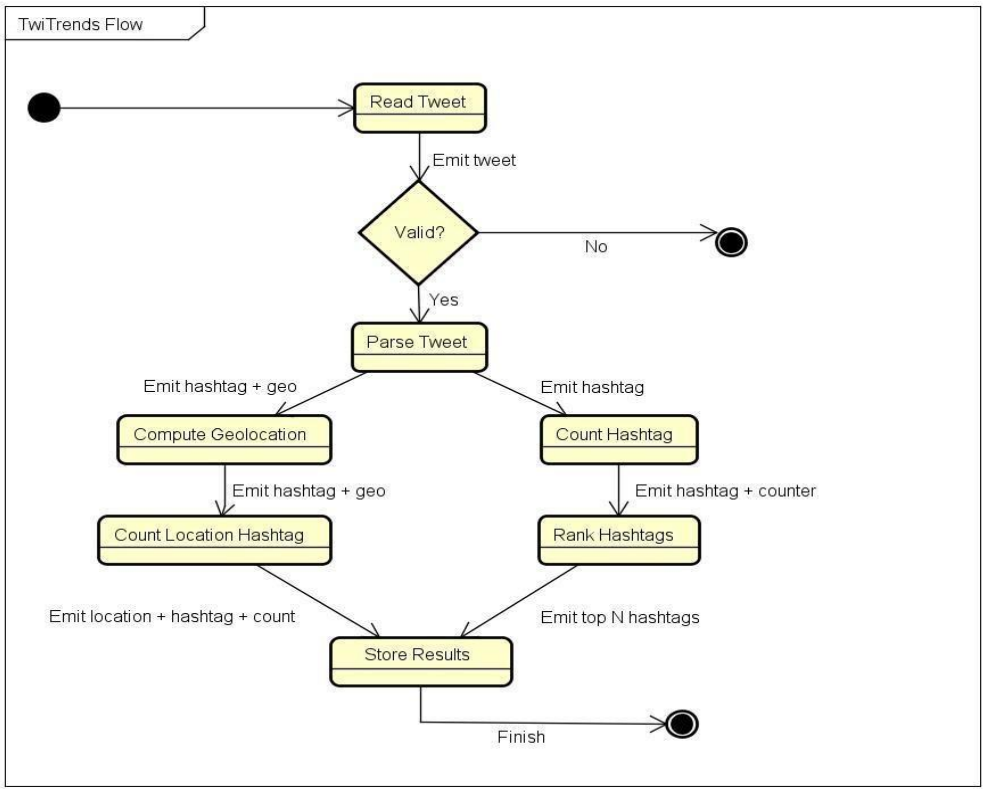

Fig 7. TwiTrends system execution steps

It should be noticed that after the validation, the presence of a hashtag is guaranteed, but the information of the geolocation is not. Whether it occurs or not, a tuple is issued for every hashtag accompanied by its location data. After the parsing of the tweet, the hashtags are processed in two parallel states: one for determining the most discussed subject and one for the classification of the hashtag at the geolocation level. For determining the trending topics, the hashtag contained in the read tweets to which initially is assigned the value 1. In the case the TwiTrends has encountered this hashtag, the associated counter is incremented, and after that, the pair (hashtag, counter) is issued for the classifying itself. In the case of a bug report at the geolocation level, the information about the geolocation is extracted from the tuple, and the city and also the country from where are issued are computed. If the information extracted from the tuple is not present or not enough for determining the concrete location, TwiTrends considers that the hashtag comes from an unknown location. The classification of a hashtag takes place after the association of a counter with. Each is received from the previous state, and the aggregated results are obtained through analysis up to that moment. Analogous with the counting of hashtags, reporting to a location takes place through associating a counter to each hashtag, but based on the location from where they are issued. Even if the hashtag was already mentioned, if it comes from another city, or country, a new counter will be assigned and initialized. In the case when the hashtag was already analyzed for the same city and the same value of the hashtag, the existing counter will be incremented.

The data resulted in the states “Count Location Hashtag” and “Rank Hashtag” are aggregated in the terminal state. If a hashtag was duplicated in the parsing state, the results of the two alternative processing are grouped in that moment. After a tweet was read, validated, parsed, processed and stored, its execution is over, and this reached the final state as in Figure 6. The execution model is continuous, so more than one tweet is executed simultaneously, and the state that a tweet is in the execution flow presented does not determine the status of another tweet, as long as it does not cause processing delays for it, which is isolated from the rest of the data in processing.

3.6 TwiTrend in a Storm cluster

To scale the system for a massive parallel execution, in a context of high volume input data and in a low latency manner, Apache Storm offer the execution of a topology in a cluster, this being dedicated to the developing and testing of complex applications.

In order to execute a topology in a cluster, it is necessary the installation and the configuration of the following components from its architecture : Server Zookeeper, Nimbus, Supervisor, Framework Apache Storm.

The prerequisites for the configuration of a Storm cluster which have to be met are resumed to a pre-installed Java version over 1.7, and Python 2.6.6. The configuration of the cluster was realized using the Ubuntu 14.04.4 operating system, but according to the Storm documentation, the compatibility is not limited only to those. The drawbacks of the framework Apache Storm and the improvements brought by Heron, lead to the migration of the topology to the Heron architecture, the new created topology being TwiTrends-Heron. The migration process of the application was relatively simple, taking into consideration that the new framework is fully compatible with Apache Storm. For transposing the topology TwiTrends on the new Heron architecture, the necessary changes were the following: removing the dependencies for the Closure plugin, adding the dependencies for the Heron API and adding the dependencies for Heron-Storm

4. TESTING AND VALIDATION

The functional testing of the TwiTrend topology was made from the point of view of the functional behavior at the component level and at the topology level. Also, a type of truthfulness testing was made for the results obtained, after the analysis and the classification of the tweets. The mechanism proposed for computing the geolocation was tested extensively, because it represent the most complex computations made by the topology. For testing, the conversion of the coordinates was made with a precision of 0.05 degrees, for a set of pairs (longitude and latitude) on which the conversion and the build of a geolocation object was applied.

The classification method of a hashtag was tested based on 39.000 randomly chosen tweets, using the Twitter Streaming API. The running of the test was made in the 25 June 2016 at the 21:00 hour. The major events which characterize that time moment were, the withdraw of the Great Britain from the European Union (#Brexit) and European football championship Euro 2016(#EURO2016), and followed by the confrontation between Northern Ireland and Wales (# NIR #WAL #WALNIR). The results obtained running the TwiTrends topology where the expected ones, through the most discussed subjects form the social network being the ones mentioned above in Table 1.

Table 1 The most mentioned hashtags (25 June 2016)

Place	Hashtag	Mentions
2	#EURO2016	629
4	#WAL	458
6	#Brexit	353
7	#WALNIR	311
8	#NIR	301

5. CONCLUSIONS AND FURTHER DEVELOPMENTS

In this section we will underline the objectives that were achieved through this project and a set of improvements that can be brought to the TwiTrends system as further developments.

The purpose of this paper was to introduce the basics of Big-Data concepts and the currently processing methods in the context of a real-world system . Based on the bibliographic study realized, the paper presents the fundamental concepts necessary in understanding the domain, the current data stream processing methods and two Apache frameworks for big-data processing. The paper presents different technologies of real-time analysis and it described in detail the Apache Storm framework, which exhibits a better performance for the last several years in the industry, for various applications. Moreover, even if recently introduced tool - Apache Heron was only in few projects integrated, the proposed implementation confirmed the results published by Twitter. The system proved to be scalable, tolerant to failure, and with a response time of the order of milliseconds which can say that the processing of data is made in real time. Moreover, according to the proposed objectives, the results obtained showed the truthfulness and reflected a realistic determination of the most discussed subjects. As expected, both Apache Storm and Heron proved their abilities to process the streams in real time. The Twitrends system has a set of characteristics, through which the important ones are a good processing speed, the reduced latency, the horizontal scalability, the ease of development of application, and also the code reusability.

Although the obtained results confer a validation for the proposed approach, a number of improvements can be brought to the system. Regarding the reporting of a hashtag to a geolocation, TwiTrends considers a zone of a certain radius, the dimension of this is given by the precision chosen for the coordinates. To obtain a more relevant reporting, the geolocation module ca be extended, to reports this values to different interest zones. Another improvement can be at the level of determination of the most discussed subjects form the Twitter social network. The method chosen was based on the hashtag criteria proved to be a good one, taking into consideration that the results obtained running the application TwiTrends determined the most discussed subjects globally. However, TwiTrends resumes the analysis at the hashtag level, not at the level of a set of hashtags which represents the same topic. By making more runs , the conclusion was that certain hashtags , as for example #Euro2016, #EURO2016, #Euro16 refer to the same discussed subject, but they are classified independently. Even if there are a a lot of further developments and improvements that can be brought to the system,it can be considered that with TwiTrends system it was obtained a reliable solution for determining automatically the "trending topics" for a twit-message in one of the most used social network –Twitter.

6. REFERENCES

[1] Jonathan Stuart Ward and Adam Barker, *Undefined By Data: A Survey of Big Data Definitions*, University of St Andrews, School of Computer Science, 2013. (accessed 24 april 2016)

[2] Thibaud Chardonnens, *Big Data analytics on high velocity streams*, University of Fribourg (Switzerland), 2013. (accessed 25 april 2016)

[3] Aftabl A. Chandio, Nikos Tziritas, Cheng-Zhong Xu, *Big-Data Processing Techniques and Their Challenges in Transport Domain*, Research Gate, februarie 2015. DOI: 10.3969/j.issn.1673-5188.2015.01.007 (accessed 24 april 2016)

[4] C.L. Philip Chen, Chun-Yang Zhang. *Data-intensive applications, challenges, techniques and technologies: A survey on Big Data*, Information Sciences Volume 275, pages 314-347, august 2014. [Online]: http://dx.doi.org/10.1016/j.ins.2014.01.015 (accessed 25 April 2016)

[5] Benoît Perroud. *A hybrid approach to enabling real-time queries to end-users*. Software Developer's Journal, 2013. (access 24 April 2016)

[6] Nathan Marz and James Warren, *Big Data Principles and best practices of scalable real time data systems*, Manning, aprilie 2015. (accessed 24 April 2016)

[7] Boyang Peng, *Elasticity and Resource Aware Scheduling in Distributed Data Stream Processing Systems*, Master Thesis, University of Illinois at Urbana-Champaign, 2015. (access 24 april 2016)

[8] Karan Patel, Yash Sakaria and Chetashri Bhadane, *Real Time Data Processing Frameworks*, International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.5, No.5, septembrie 2015, DOI:

10.5121/ijdkp.2015.5504. (accessed 24 may 2016)

[9] Martin Illecker, *Real-time Twitter Sentiment Classification based on Apache Storm*, Master Thesis, Innsbruck, 2015.

[10] Bc. Dávid Katuščák, *Dynamic Processing of Event Streams Using Java Tools*, Master's thesis, Brno, 2015.

[11] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, Siddarth Taneja, *Twitter Heron: Stream Processing at Scale*, in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015. [Online]: http://dx.doi.org/10.1145/2723372.2742788

[12] Arkaitz Zubiaga, Damiano Spina, Raquel Martinez, Victor Fresno, *Real-Time Classification of Twitter Trends*, Journal of the American Society for Information Science and Technology, March 2014. [Online]: http://arxiv.org/abs/1403.1451v1

[13] Doug Laney, *3D Data Management: Controlling Data Volume, Veocity, and Variety*, META Group, February 2001. [Online]: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data Management-Controlling-Data-Volume-Velocity-and-Variety.pdf