Lambda Architecture for sentiment analysis

Cosimo Giani
cosimo.giani@stud.unifi.it
University of Florence, Italy

Abstract

The following work is a full term project for the Parallel Computing course held by professor Marco Bertini, University of Florence. In this project is proposed a Lambda Architecture to perform sentiment analysis on Twitter data in real-time. A GUI has also been implemented for viewing the results with the aim of facilitating the understanding of the benefits deriving from the use of this type of architecture.

1. Introduction

The subject matter of this report consists in the implementation of a Lambda Architecture designed to analyze tweets downloaded from the popular social network Twitter. A **Lambda Architecture**, in detail, represents an architecture designed specifically for the processing of a large amount of immutable and ever-increasing data. The implementation of such a software system is in fact ideal when you need to process not only a large set of data stored in a dataset, but also when it is necessary to add a real-time one to this computation on new incoming information.
In fact, this architecture benefits from the subdivision of the processing on different levels:

- ***batch layer***: it stores the master copy of the dataset and precomputes batch views on that master dataset. It needs to be able to do two things: store an immutable, constantly growing master dataset, and compute arbitrary functions on that dataset.

- ***speed layer***: compute a batch view is a very time-consuming task. Data that came after the beginning of the last completed batch view processing are not represented in the batch view. The aim of this layer is to elaborate these new data until they are represented in the batch view, making use of low-latency techniques.

- ***serving layer***: it is a specialized distributed database that supports batch updates and random reads but does not need to support random writes.

Real-time views contain only information derived from the data that arrived since the batch views were last generated and are discarded when the data they were built from is processed by the batch layer. The batch and real-time views are combined to create query results [4].
Sentiment analysis (also known as *opinion mining*) is a field of natural language processing that deals with building systems for identifying and extracting opinions from text. It is based on the main methods of computational linguistics and textual analysis. Sentiment analysis is used in many sectors, for instance politics, marketing, social media analysis, evaluation of consumer preferences.
For these reasons it may be appropriate to perform this analysis on certain keywords through the aforementioned Lambda Architecture.

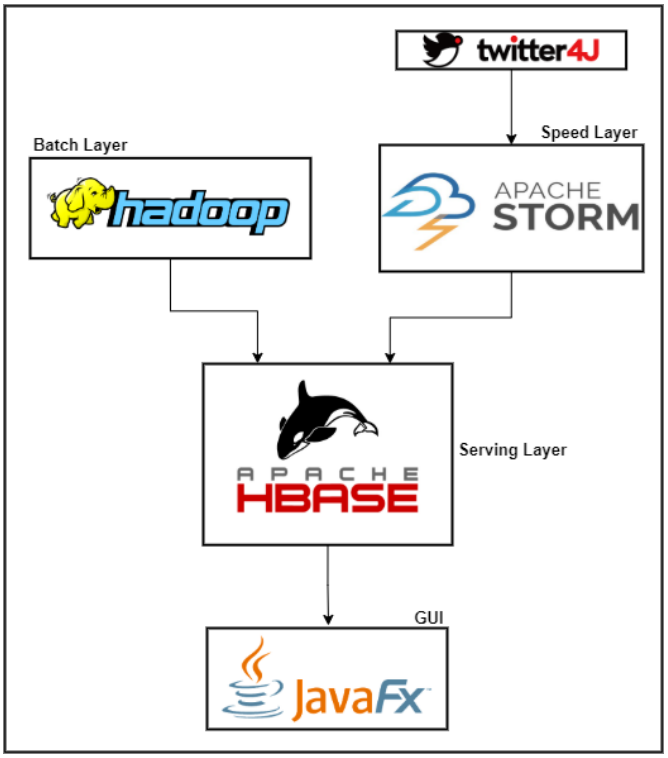

Figure 1. Lambda Architecture scheme

2. Implementation

The creation of a Lambda Architecture, as described in the introduction, involves the implementation of the three components relating to batch layer, speed layer and serving layer. With regard to the project presented here, the *Apache Hadoop* and *Apache Storm* software systems were used respectively for the batch and real-time part and the distributed database *Apache HBase* for the functionalities which the serving layer is used. These systems were executed in a pseudo-distributed mode on a local cluster.
A software module has been implemented as well, that deals with formulating queries to the serving layer to be able to represent graphically the results obtained. Figure [1] shows the architecture scheme described so far.
The speed layer represents the main part of this project, in fact it is the first module to be started. In addition to building the topology, it takes as input the keywords on which the sentiment analysis will be performed and creates the tables that will be used by the serving layer.
Once started, the batch layer and the GUI are executed to complete the architecture in its entirety. To avoid running the architecture for a long period of time, a stream of tweets complementary to the real-time one was provided to increase the number of tweets managed by the system [5].
Finally, the code for the synchronization between speed and batch layer that makes use of timestamps has been implemented in this level.

2.1. Sentiment classifier

Sentiment analysis involves classifying opinions in text into categories like *positive* or *negative*. To achieve such a thing, a sentiment classifier model has been trained using the *LingPipe* library [3] with *sentiment140* [2], a dataset which contains 1.6 million of annotated tweets.
After the training process, the classifier model has been saved for being usable by the batch and the speed layer and it has also been evalueted with another dataset [5]: the classification accuracy is

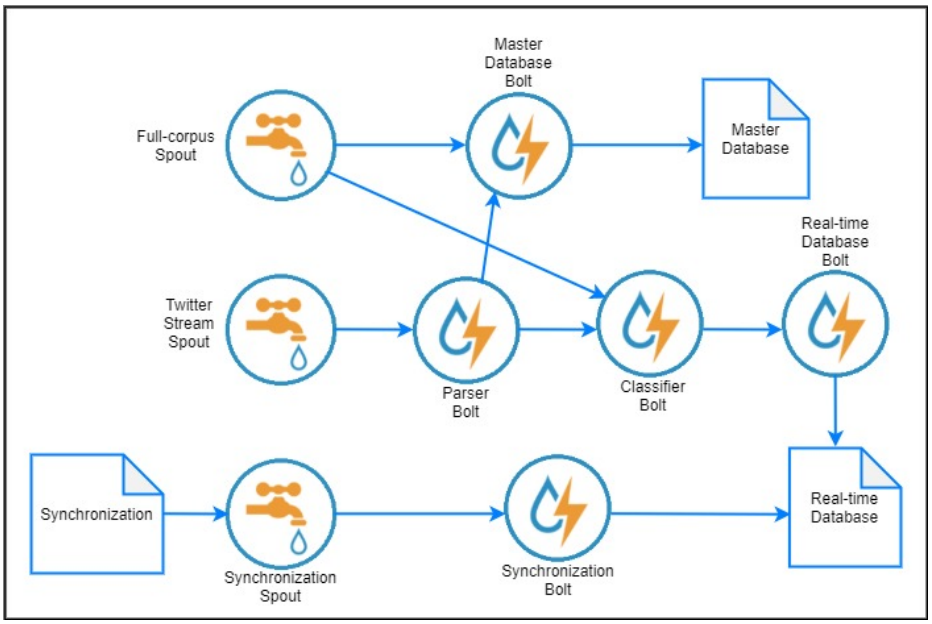

Figure 2. Topology scheme

about 68%, but the creation of a well-designed machine learning model with high-accuracy and good generalization ability is out of the scope of this project.

2.2. Speed layer

For the purpose of implementing the speed layer, the *Apache Storm* framework was used. As already mentioned, this takes as input the keywords on which to carry out the sentiment analysis and creates, if not already present, the tables necessary for the serving layer. It should be noted that for the use of the *Twitter4j* (a Java library for the Twitter API [1]) stream it was necessary to save in a text file the access keys to this stream.
The topology of this level, shown in Figure [2], was defined as follows:

- ***twitter stream spout***: makes use of the Twitter4j library to simulate a stream of tweets in real-time. In particular, it filters tweets to use only those that contain the keywords we are looking for and these tweets will be sent out only if written in English.
- ***full-corpus spout***: although in a real application this spout does not make sense to exist, in the context of this project, as already mentioned, it has been inserted in a complementary way to the Twitter4j stream to increase the number of tweets processed by the architecture.
- ***synchronization spout***: takes care of checking if the batch processing start and end timestamps have both remained unchanged since the last check. If not, it communicates the new start timestamp to the related bolt.
- ***synchronization bolt***: as soon as a timestamp is received from the aforementioned spout, it deletes the rows with timestamps prior to the one received from the real-time tweet table, as long as they exist.
- ***parser bolt***: it takes care of parsing the data coming from the *twitter stream spout*, i.e. given a tweet as input, checks which of the searched keywords are present and outputs a tuple with the structure `tweet_ID, text, keywords`.
- ***classifier bolt***: given the previously trained classifier and a tuple from the *parser bolt*, classifies the text of a tweet and for each

value of the `keywords` field produces the `keyword-sentiment` pair.

- ***master database bolt***: inserts a row in the master database of the architecture for each tuple that arrives, so that the batch will also have real time tweets.
- ***real-time database bolt***: takes the tuples from the *classifier bolt* and arranges them in an row in the appropriate real-time table.

2.3. Serving layer

For the serving layer it was decided to use *Apache HBase*, an open-source, distributed and non-relational database. The tables used by this module are those created in the speed layer topology. In this context, the following tables have been constructed:

- ***Real-time database***: it is the database for the real-time part where each row contains the keyword and the related sentiment for each tweet and obviously the fundamental timestamps for synchronization.
- ***Synchronization***: it contains only two lines that respectively indicate the start and end timestamps of a batch processing. These are used precisely in the synchronization between the batch and the speed layer, particularly useful since they are used to understand when it is necessary to discard information within the real-time database.
- ***Master database***: the master database used by the architecture as previously mentioned.
- ***Batch view***: every time the batch layer finishes processing, it saves information about the result of its work in this table. Each keyword is associated with a row with a relative number of positive or negative sentiment.

2.4. Batch layer

As for the implementation of the batch layer part of the application, as already mentioned, the *Hadoop* framework was used. The purpose of this component is to analyze the tweets contained in the master dataset and then save the results obtained in the appropriate table stored in *HBase*. Specifically, three software modules have been defined, which represent *Driver*, *Mapper* and *Reducer* of the *Hadoop* system respectively. Their implementation can be summarized as follows:

- ***Driver***: it executes a MapReduce job on the master database, writing the results obtained in the relative batch view table. It also takes care of writing the start and end processing timestamps in the appropriate table, so that, when the batch layer finishes its execution, it is possible to discard the data processed by it from the real-time view through the synchronization mechanism and the rows inserted before the start of the batch run will no longer be part of the real-time table.
- ***Mapper***: at each iteration it receives as input a line of the dataset containing the data of a tweet, it classifies the text by recalling the appropriate method of the model trained previously and produces, for each keyword of the tweet, a `key-value` pair where the `key` are keywords of the tweet and the `value` corresponds to the classified sentiment.
- ***Reducer***: at the end of the Mapper execution, the Reducer receives a pair of values for each of the considered classifications, where, in particular, the second term represents the list of sentiment related to the key. Therefore, a positive/negative sentiment is counted and its value increased accordingly in the batch view.

2.5. Graphical User Interface

In order to present the results of the proposed architecture, a **G**raphical **U**ser **I**nterface (**GUI**) was implemented with *JavaFX*, Figure [3].
This is simply composed of two views, one for the real-time part and the other for the batch part, and a bar chart where the results of the query made to the system are reported. Specifically, this was achieved by means of three files:

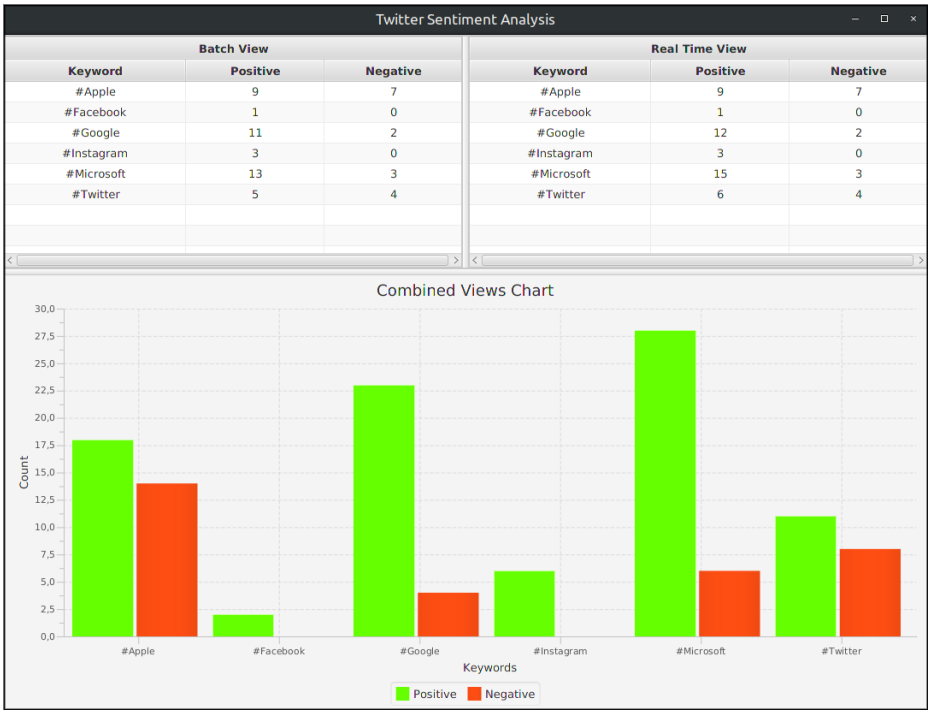

Figure 3. GUI representation

- ***GUI interface***: it is the class that contains the code to start the interface and it details its style.
- ***GUI.fxml***: this is the file with the FXML code that allows you to separate the interface from the application logic of the software.
- ***GUI***: is the GUI software, which retrieves the information from the batch and real-time views and refreshes the data every second, in order to keep the GUI always updated. In particular, for the retrieve of data from the real-time database some aggregation operations were necessary before being able to present the real-time results in the GUI.

3. Conclusions

To draw conclusions about this work, observing the results presented by the GUI, it is highlighted how the Lambda Architecture created is able to bring significant benefits to sentiment analysis: indeed a system of this type is able to manage large amounts of data, allowing to obtain results for real-time monitoring of analytics like sentiment analysis.

References

[1] Twitter4j for the twitter api. *http://twitter4j.org/en/*.

[2] R. B. A. Go and L. Huang. Twitter sentiment classification using distant supervision. *https://www.kaggle.com/kazanova/sentiment140*, 2009.

[3] Alias-i. Lingpipe 4.1.0. *http://alias-i.com/lingpipe*, 2008.

[4] N. Marz and J. Warren. Big data: Principles and best practices of scalable real-time data systems. *New York; Manning Publications Co.*, 2015.

[5] N. J. Sanders. Twitter sentiment corpus. *https://github.com/guyz/twitter-sentiment-dataset*, 2011.