Twitter Sentiment Analysis using Lambda Architecture

Parallel Computing 2019/2020

Claudia Raffaelli

Department of Information Engineering
University of Florence
Via di Santa Marta 3, Florence, Italy
claudia.raffaelli96@gmail.com

Abstract

Social networks such as Twitter have gained great attention nowadays. The heavy presence of people on social networks causes a big amount of data ready to be analyzed. However, there are a lot of issues related to the reviewing of large chunks of information. For this task come to aid a few computational tools that can be summarized under the name of Lambda Architecture [1].

The goal of this paper is to present a functioning Lambda Architecture built to compute a sentiment analysis upon tweets, according to specific keywords.

The sentiment analysis or opinion mining is the process of determining if a particular block of text is expressing a positive or negative reaction upon something. Clearly, the rise of social media has fueled interest in sentiment analysis. Online opinion has turned into something to take into account by businesses looking to market their products, identify new opportunities and manage their reputations.

1. Introduction

Big data is a field that treats ways to analyze, extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software. Also this big amount of data is usually kept in different machines.

Lambda Architecture has born as a solution to the processing of massive quantities of data. In order to do so, are needed a variety of tools. The main idea of the Lambda Architecture is to build Big Data systems as a series of layers. At the lower level there is the Batch Layer. Subsequently there is the Serving Layer, and lastly the Speed Layer.

Hereafter, and in Figure 1, are presented broadly the main features of each layer.

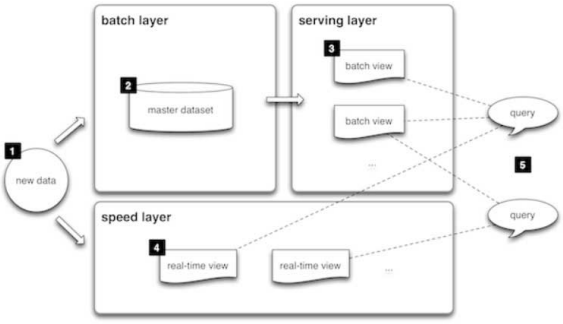

Figure 1: Lambda Architecture Overview

1. **Batch Layer:** New data comes continuously, as a feed to the data system. It gets fed to the batch layer and the speed layer simultaneously. The batch layer precomputes the master database (that contains all data) into batch views so that queries can be resolved with low latency. The batch views are computed each time upon the entire database.

   This task is realized using a distributed processing system that can handle very large quantities of data. For the Batch Layer will be used Apache Hadoop, which is the leading batch-processing system.

2. **Speed Layer:** Essentially, the speed layer is responsible for filling the ”gap” caused by the batch layer’s lag in providing real-time views based on the most recent data.

   This layer’s views may not be as accurate or complete as the ones eventually produced by the batch layer, but they are available almost immediately after data is

received, and can be replaced when the batch layer's views for the same data become available.

To perform the task of providing speed layers views is used Apache Storm while the storing of the views is realized through Apache HBase, along with the master database.

3. **Serving Layer:** The outputs from the batch layer in the form of batch views and those coming from the speed layer in the form of real-time views get forwarded to the serving. This layer is then responsible for merging both data in order to provide a complete view concerning the data.

Putting this layers together provides some key features such as:

- Robustness and fault tolerance: distributed systems presents few challenges, linked to keeping distributed databases consistent, handle duplicated data and concurrency.
- Low latency reads and updates: of course latency requirements depends on different kinds of applications. Regardless, is always good to keep latency as low as possible.
- Scalability: is the ability to maintain performance as data increase. Scaling is accomplished by adding more machines.
- Query the database: to obtain insights upon the data stored.

The aim is to create a pseudo-distributed system. This is nothing but a configuration that is run on a single node. By doing so is simulated the real distributed system, but without having to think about the resources and other users sharing the resource.

The main idea of this paper is to describe a system that is capable of reading a continuous stream of Tweets from a CSV file. This Tweets are handled by both storing them raw in a pseudo-distributed database where there will be pre-computed batch views, and by forwarding them to the speed layer for real-time views. Eventually the data from both layers will be merged to provide a complete view.

The computation to be performed upon the Tweets is a **Sentiment Analysis**. The Tweets that contains specific keywords are classified in order to predict their positive or negative sentiment regarding that keyword. The job to be executed so, is the counting of the sentiment of every keyword.

Even if Lambda Architecture presents numerous benefits, some cons need to be considered too. Of course both batch and realtime sides require to be kept in sync in order to provide a full view of the computed data. Lambda architecture is inherently complex but with the proper tools in can be simplified.

2. Overview of the approach

In this section are described the various components of the computation that is the Sentiment Analysis task, and the three layers of the Lambda Architecture.

2.1. Sentiment Analysis

The realization of the Sentiment Analysis is carried out by making use of the LingPipe library [4]. This library provides the main methods for the training and use of the classifier. For the training phase has been used the dataset [3], while the other dataset [2] is used for continuously inject new tweets into the system.

This is done in order to simulate a continuous stream of tweets, reproducing a behavior similar to the Twitter platform.

At the start of the computation is chosen a set of keywords, upon which is done the sentiment analysis. If in a certain tweet is present at least one of this keywords, then the tweet must be classified in order to provide the proper sentiment.

The classifier is trained in such a way that is capable to recognize two levels of sentiment:

- A negative sentiment, using the label 0
- A positive sentiment, with label 1

2.2. Batch Layer

As already anticipated, the Batch Layer is responsible for pre-computing batch views of the whole database. Each time a new batch view is available, the old one is not immediately discarded, not until the next one is ready. This is made possible by the fact that every time that a new batch view is computed, the system goes through the whole master database from scratch, taking into account even the older, and already computed records of the past batch view.

Keeping this in focus, the task of computing batch view is carried on by Apache Hadoop system. Each time a new view needs to be prepared, is performed a MapReduce processing on the whole master database.

MapReduce is a framework composed of a map procedure, which executes some filtering and sorting, and a reduce method, which performs a counting operation. MapReduce is so important because it allows to orchestrates the processing of a massive amount of data, even

distributed on different servers. Also it does so running the various tasks in parallel and managing the communications between the different servers, but still guaranteeing redundancy and fault tolerance. Therefore the main idea of MapReduce is to ensure scalability and fault tolerance.

In the particular case of sentiment analysis upon tweets, the MapReduce is explained in an example in Figure 2 below.

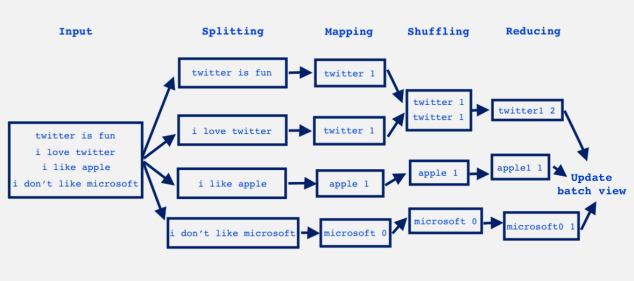

Figure 2: MapReduce schema

MapReduce is managed by taking as input for the Map function the whole master database. For each row of the database is checked if the tweet text contains at least a keyword. If that is the case, is computed the sentiment of the tweet and outputted the pair $(keyword, sentiment)$.

The results are shuffled by the shuffle nodes, so that data based on the same output key (produced by the map function) is redistributed. By doing so all data belonging to one key is located on the same worker node making the performances better. The reducers are responsible for counting how many positive and negative tweets there are for that keyword. The final output is saved in the batch view table.

2.3. Speed Layer

Of course because of the nature of the Batch Layer computation, for large database, generating a new batch view can take some time. This is why is extremely important to make available a layer that is capable of handling new tweets really fast, in order to integrate the batch view, that might not be up to date.

To provide this kind of structure, has been used the computation system Apache Storm. Apache Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.

The idea behind Storm is to use a series of Spouts and Bolts to allow distributed processing of streaming data. A Storm application is designed as a "topology" in the shape of a directed acyclic graph with spouts and bolts acting as the graph vertices. The topology acts as a data transformation pipeline. At a superficial level the general topology structure is similar to a MapReduce job, with the main difference being that data is processed in real time as opposed to in individual batches.

To topology used to perform the task is described in the Figure 3 below.

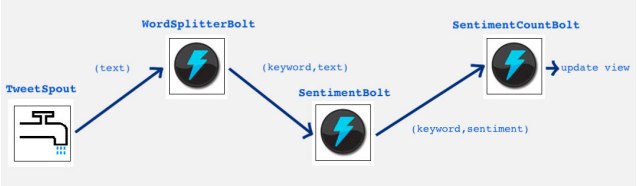

Figure 3: Storm Topology

The TweetSpout provides the systems of a stream of tweets. For each tweet is stored the date, the timestamp and the text in the master database, in order to maintain a raw version of the tweet in the system.

Every tweet is sent to the first bolt. The first bolt, WordSplitterBolt takes the text of the tweet as input and separate each word from the other. At this point it checks if that word is contained in a white list of keywords. If that is the case, it forwards the pair $(keyword, text)$ to the next bolt. This bolt, SentimentBolt, is responsible for computing the sentiment of the text containing the keyword, using the classifier already discussed. The output emitted by this bolt is a pair $(keyword, sentiment)$, that is captured by the last bolt, SentimentCountBolt.

This last step completes the speed layer job, putting together the count of positive and negative sentiment for each keyword and storing the result in the speed layer view in HBase.

2.4. Serving Layer

The Serving Layer plays the role of a supervisor of both Batch and Speed layers. It is responsible for querying the Hbase database in order to get an instance of batch view and real-time view. Once the data from both tables is retrieved, the Serving Layer is in charge of merging the two views, obtaining a full view of the sentiment of each keyword for the whole database.

As first thing, in the Table 1 is shown the structure of both batch and realtime view.

Table View	
Keyword	String
Sentiment	String
Count	Integer

Table 1: Realtime View and Batch View schema

The key of the record is also the keyword and saved as a String. The sentiment can be positive or negative, and is

also a String. Lastly the count is simply saved as an Integer.

The crucial aspect of the serving layer to pay particular attention to, is how the synchronization between the two layers is performed. The complete view provided, needs to be consistent, that is that the sum of the two tables needs to yield to a correct representation of the master database. In order to guarantee so, a few constrains needs to be complied.

The technique adopted consists in having always at hand two batch view tables. After the computation of a batch view through MapReduce, the result is copied back to a consistent batch view henceforth called BatchViewComplete, the only batch view table to be queried.

The other batch view table, BatchViewInComputation can be used for another computation, while the results more updated and consistent are stored safely in BatchViewComplete. Each time a new batch view is ready, the realtime view table is initialized, so that the new tweets that will arrive from that moment forward will be analyzed and saved to guarantee consistency with the master database.

To bring more clarity, below is reported the pseudocode, in 1, of what just described.

Algorithm 1 Batch View Compute Algorithm

```
1: procedure COMPUTENEWBATCHVIEW
2:     new batchViewInComputation
3:     batchViewInComputation to batchViewComplete
4:     realtime view is initialized
```

At any given time the two tables (realtime and batchComplete) can be queried and the result of the sum of both values is used to print out the complete view of the database.

3. Evaluation of performances

Taking into account the major issues linked to Lambda Architecture, the main experiments were focused on trying to establish if the synchronization between the two layers, provided by the serving layer, was correct. In order to do so, the behavior of the system was strictly analyzed by injecting few tweets at a time and checking closely that the views were consistent.

To conclude the analysis performed, the sentiment analysis appears to be really useful in order to provide important information regarding the thoughts of people upon a particular topic. Also a good management of big data is fundamental nowadays, a task that Lambda Architecture is fully capable to comply.

A future scope of work can be to use Twitter API in order to push inside the system a real stream of tweets directly from Twitter. Also the classifier can be improved for better performances.

References

[1] Nathan Marz, James Warren. *Big Data. Principles and best practices of scalable real-time data systems*. Manning, 2015.

[2] Go, A., Bhayani, R. and Huang, L. *Dataset sentiment140*. https://www.kaggle.com/kazanova/sentiment140

[3] Sanders Analytics. *Twitter Sentiment Dataset*. https://github.com/guyz/twitter-sentiment-dataset

[4] Alias-i. *LingPipe Library*. http://alias-i.com/lingpipe/index.html