Twitter Sentiment Analysis using Lambda Architecture

Federico Magnolfi
Università degli Studi di Firenze
Florence, Italy
federico.magnolfi2@stud.unifi.it

Iacopo Erpichini
Università degli Studi di Firenze
Florence, Italy
iacopo.erpichini@stud.unifi.it

***Abstract*—Calculating arbitrary functions on large real-time data sets is a difficult task. In this study a possible approach to problems of this type is shown: as an example task we consider the analysis of the feelings of the tweets. Instead of getting tweets through the Twitter API, we simulate issuing them by reading them from the sentiment140 dataset.**

In our scenario we create a Lambda Architecture that uses Apache Hadoop for the Batch Layer, Apache Storm for the Speed Layer and HDFS for data management. We use LingPipe to classify tweets using computational linguistics.

This architecture allows to harness the full power of a computer cluster for data processing, is easily scalable, and meets low latency requirements for answering queries in real time.

***Index Terms*—Lambda-Architecture, Sentiment Analysis, Apache Hadoop, Apache Storm, LingPipe**

I. Introduction

The advent of Big Data has brought about a sea change in data processing over recent years. Big Data differs from traditional data processing through its use of parallelism: only by bringing multiple computing resources to bear we can contemplate processing terabytes of data.

For this purpose, it is essential to build horizontally scalable systems: it means that you can get more processing power by adding more machines to your resource pool, without the need to update the existing ones. Scale horizontally a system is a good option to facilitate the management of a cluster and reduce its management costs. The Hadoop framework [3] and the other tools in its ecosystem (such as Storm [4]) allow you to create a system with these properties.

In its simplest form, Hadoop allows you to run distribuited jobs, with input and output data located on the Hadoop Distribuited File System (HDFS), or in a distribuited database. Hadoop abstracts the communication between the machines in the cluster: the programmer can define the job to be performed as a simple pipeline of Map-Reduce tasks.

The Hadoop Map-Reduce framework allows you to easily process large amounts of data but has the disadvantage that the time required to perform this calculation can be quite long: this is a problem if the system must be able to answer queries in real-time with up-to-date results.

The Lambda Architecture (Figure 1) is a particular approach to Big Data popularized by Nathan Marz [1] [6], derived from his time at BackType and subsequently Twitter. With this architecture, it is possible to design systems that exploit all the privileges of using the Hadoop framework, without the usual disadvantages: this is possible thanks to the fact that the heavy calculation is separated from the calculations on recent data.

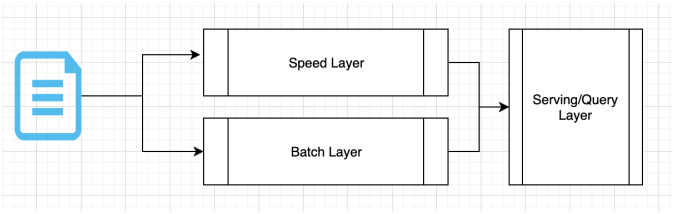

Fig. 1. Lambda Architecture.

Sentiment Analysis is the automated process of analyzing text data and classify it into positive, negative or neutral sentiment. Performing Sentiment Analysis on Twitter data using machine learning can help companies understand how people are talking about their brand. In this paper, we want to implement a Lambda architecture that classifies a big quantity of data. For tweets classification we use the LingPipe library: LingPipe's architecture is designed to be efficient, scalable, reusable, and robust [2].

In the next sub-section we will discuss about general concepts of batch processing, real time processing.

II. Lambda Architecture

Batch layer

The batch layer needs to be able to do two things: store an immutable, constantly growing distributed master dataset, and compute arbitrary functions on that dataset. The batch layer output (*Batch View*) is repeatedly computed as a function of the whole master dataset (Algorithm 1).

Algorithm 1: runBatchLayer()

Input : dataset
Output: batchView
1 **while** *(true)* **do**
2 batchView = computeBatch(dataset)
3 **end**

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 the speed layer is to elaborate on a temporary view of these new data until they are represented in the batch view.

Data comes in-stream or small batches and they are processed by the Speed Layer as soon as they arrive (Algorithm 2): this is useful to make real-time data analysis while data are still coming.

Algorithm 2: runSpeedLayer()

Input : newData
Output: speedView
1 **while** *(true)* **do**
2 speedView = computeSpeed(speedView, newData)
3 **end**

Serving layer

The role of the Serving Layer is to allow arbitrary queries on pre-calculated views (both Batch and Speed). It can take advantage of a specialized distributed database, which has to support random reads on it, but not random writes. Random writes cause most of the complexity in databases, so a Serving Layer database can be extremely simple. This layer can be summarized as a set of query functions like: $result = query(batchView, speedView)$

III. Technologies

In this project we use *Apache Hadoop 2.9.2* for batch processing (Batch Layer) and *Apache Storm 2.1.0* for real-time processing (Speed Layer).

Apache Hadoop is a software framework for distributed storage and processing of big data using the MapReduce programming model. Apache Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.

All the layers use *Hadoop Distribuited File System* (HDFS) as distributed storage, like a distributed database. The *Serving Layer* answer queries reading the Batch and Speed views on the HDFS.

IV. Implementation

In our architecture there are five main software modules:

- Generator
- Classifier
- Batch Layer
- Speed Layer
- Query

Our program is able to analyze feelings per *keyword* since sentiment analysis is usually used to understand emotions related to a certain brand, product or hashtag. A parameter controls the period of time for which you want to summarize the sentiment analysis (e.g.: per year, per day, per hour...).

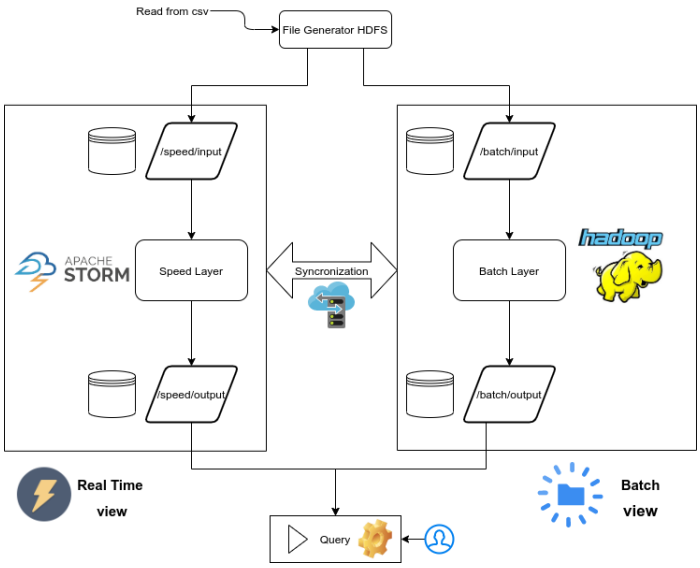

Fig. 2. Our architecture schema. The tweets are generated by sampling the sentiment140 dataset and saved in the HDFS. Batch and Speed layers produce batch and realtime views, respectively. Finally, the query module allows user to get statistics based on both views.

A. Generator

The Generator is a continuously-running process that simulates the emission of tweets by users. Tweets are sampled from the sentiment140 dataset [5], buffered (until the buffer is full or a certain amount of time is elapsed) and then writed as a new file into the `/batch/input` and `/speed/input` HDFS folders.

The parameters $maxFileDim$ and $maxDeltaTime$ control the buffer dimension and the max waiting time before write, respectively.

The generator associates each generated tweet with the current timestamp. The timestamp and the text of the tweet are concatenated to form a line in the output file.

B. Classifier

We used LingPipe 4.1.2 [2] to classify the tweets' sentiment into positive or negative.

We trained the classifier on the whole *sentiment140* dataset, which contains 1.6 Millions of annotated tweets.

After the training process we saved the classifier for being usable by Hadoop and Storm. The classification accuracy is about 80%, but the creation of a well-designed machine learning model (with high-accuracy and good generalization ability) is out of the scope of this study.

C. Batch Layer

The batch layer computation is designed as a Map-Reduce job (Figure 3). The mapper (Algorithm 3) receives a tweet (text and timestamp), it makes the classification and then sends the output to the reducer as a set of key-value pairs of the form `<period/keyword, sentiment>`. The period is determined by the timestamp, while the keywords are obtained from the text of the tweet.

The reducer (Algorithm 4) simply takes care of counting the feelings associated with the key `period/keyword` and then it save the result into the HDFS output folder specified by the driver.

Since the Hadoop Map-Reduce job requires an empty output directory when the job starts, to always have a readable batch view the driver alternates the batch output HDFS folder between `/batch/output0` and `/batch/output1`.

The driver also takes care of write which is the last written output directory, writing the path into the file `/sync/last_batch_output.txt`.

We'll discuss the details about the synchronization between Batch and Speed layers in the *Ping Pong Schema* section.

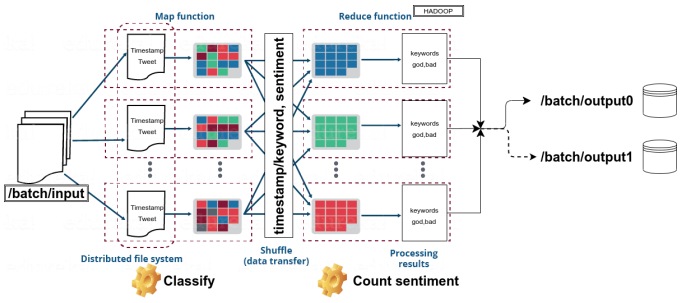

Fig. 3. The MapReduce schema in our batch layer.

Algorithm 3: Mapper

Input : tweet
Output: <timestamp/keyword, sentiment>

```
tweetTimestamp, tweetText = tweet.split(",")
classifier = loadClassifier()
sentiment = classifier.evaluate(tweetText)
for word in tweetText.split(" ") do
    outputKey = tweetTimestamp + "/" + word
    context.write(outputKey, sentiment)
end
context.write(tweetTimestamp + "/total", sentiment)
```

Algorithm 4: Reducer

Input : key, listOfSentiments
Output: <key, "numGood,numBad">

```
numGood = 0
numBad = 0
for sentiment in listOfSentiments do
    if sentiment == 0 then
        numBad++
    else
        numGood++
end
context.write(key, str(numGood) + "," + str(numBad))
```

D. Speed Layer

The topology developed using Storm is a simple concatenation of a spout and two bolts (Figure 4).

The role of the spout is to read the tweets from the input folder, send them to the *Classifier Bolt*, and archive them when they are completely processed.

The two bolts behave similarly to MapReduce in the *Batch Layer*. The first processing bolt is the mapper that classifies the tweet. The last bolt counts the histogram of sentiments for a specific `period/keyword` and saves the result on the HDFS.

The main differences with the Batch Layer are that each tweet is processed one time only and that the counters have to be reset when the batch layer finishes processing the related tweets.

More details about the synchronization between Batch and Speed Layer are reported in the next section.

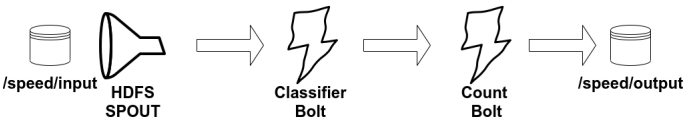

Fig. 4. The topology of our speed layer.

E. Ping-Pong Schema

Generated tweets can be:

1) Not yet seen by the batch layer
2) In processing by the batch layer for the first time
3) Already processed by the batch layer

Only those in the last group are already represented in the batch view. For the others, it is up to the speed layer to calculate the related statistics. Therefore is required a sort of communication between the batch and the speed layer. Furthermore, when the batch layer finishes calculating a batch view, all the tweets of the second group become of the third: it is therefore necessary that their statistics are removed from the speed view.

This synchronization is called *Ping-Pong schema* (Figure 5). To ensure this behaviour, the driver of the batch layer keeps updated two files:

- `/sync/progress_timestamp.txt`: contains the timestamp at which the in-progress batch computation started
- `/sync/processed_timestamp.txt`: contains the timestamp at which the last finished batch computation started

The *Speed Layer* reads the content of the progress timestamp file to know what to do with the incoming tweet. In output, there is a different folder for each progress timestamp: the statistics inside a folder refer to tweets with a more recent timestamp than the folder name. The counters are reset when the progress timestamp changes. If an incoming tweet has a timestamp prior to the one in progress, then the tweet is discarded because the Speed Layer is not counting anymore

such old tweets. Otherwise, the corresponding counter is in-memory incremented: output files are stochastically updated to not overwhelm the HDFS.

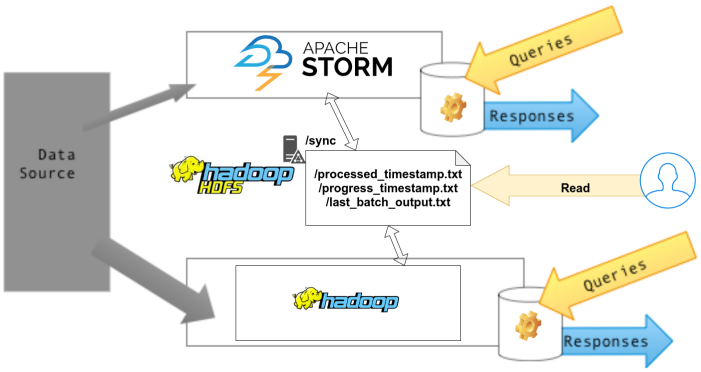

Fig. 5. Ping-Pong realized with Hadoop HDFS file system.

F. Serving Layer

The Serving layer is a Graphical User Interface (Fig. 7 in the appendix) that reads Batch and Speed views.

The GUI allows you to specify a time interval by entering the start and end date. The sentiment analysis considers the tweets sent in the chosen interval.

A *real-time summary* is provided for all tweets and for specified keywords.

V. Demo

Before running the processes, the parameters can be specified in the *Globals* class.

Main parameters include all the *input/output/sync* HDFS ***paths***, and a list of ***keywords***. For example, keywords can be brands: in the GUI it's possible to see the sentiments relative to these particular brands.

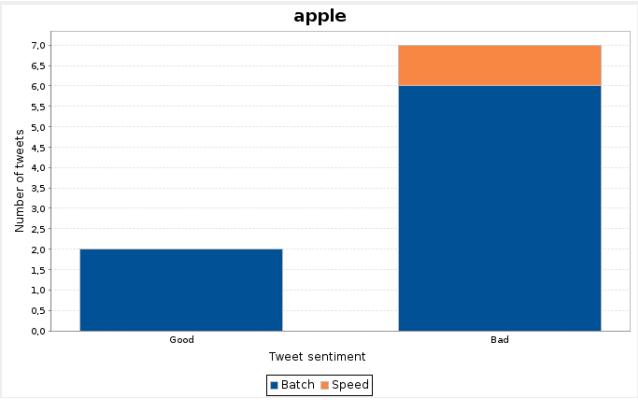

Fig. 6. Stacked bar chart that represent the sentiments associated to the *Apple* brand. Colors indicate from which layer the statistics are obtained: blue for Batch, orange for Speed.

VI. Conclusion

The benefits of using the Lambda Architecture go beyond just the possibility of scaling computing resources. It's possible to have the advantage of both heavy batch processing (reliability, scalability) and real-time processing (low latency).

A Lambda Architecture system is able to handle very large amounts of data, and you'll be able to get value out of it. It is also able to immediately deliver the results you want, allowing real-time monitoring of analytics like sentiment analysis.

Batch and speed processing together enable building of new interesting applications based on big data analysis.

We can summarize the main benefits of the *Lambda Architecture*:

- Raw input data remains unchanged
- Exploit power of PC clusters
- Reliability (it's fault tolerant)
- Easy scalability
- Low-latency results

References

[1] Marz, N. (2013). Big data: principles and best practices of scalable realtime data systems. [S.l.]: O'Reilly Media. ISBN: 978-1617290343
[2] LingPipe: Tutorial for sentiment analysis http://www.alias-i.com/lingpipe/
[3] Apache Hadoop: https://hadoop.apache.org/
[4] Apache Storm: https://storm.apache.org/
[5] Sentiment140 dataset for Academics: http://www.sentiment140.com
[6] Paul Butcher. 2014. Seven Concurrency Models in Seven Weeks: When Threads Unravel (1st. ed.). Pragmatic Bookshelf.

APPENDIX

Fig. 7. This Graphical User Interface is the Serving Layer of the Lambda Architecture. It's possible to specify the time interval of the sentiment analysis. In this example, analysis is made on all tweets and on three different brands: *Facebook*, *Apple*, *Google*.