Distributed Computing and Storage Architectures

Juan Jose Soriano Escobar

Redona Brahimetaj

Master in Applied Computer Science and Engineering
Vrije Universiteit Brussels
Brussels Belgium
January 15, 2017

Contents

List of Figures

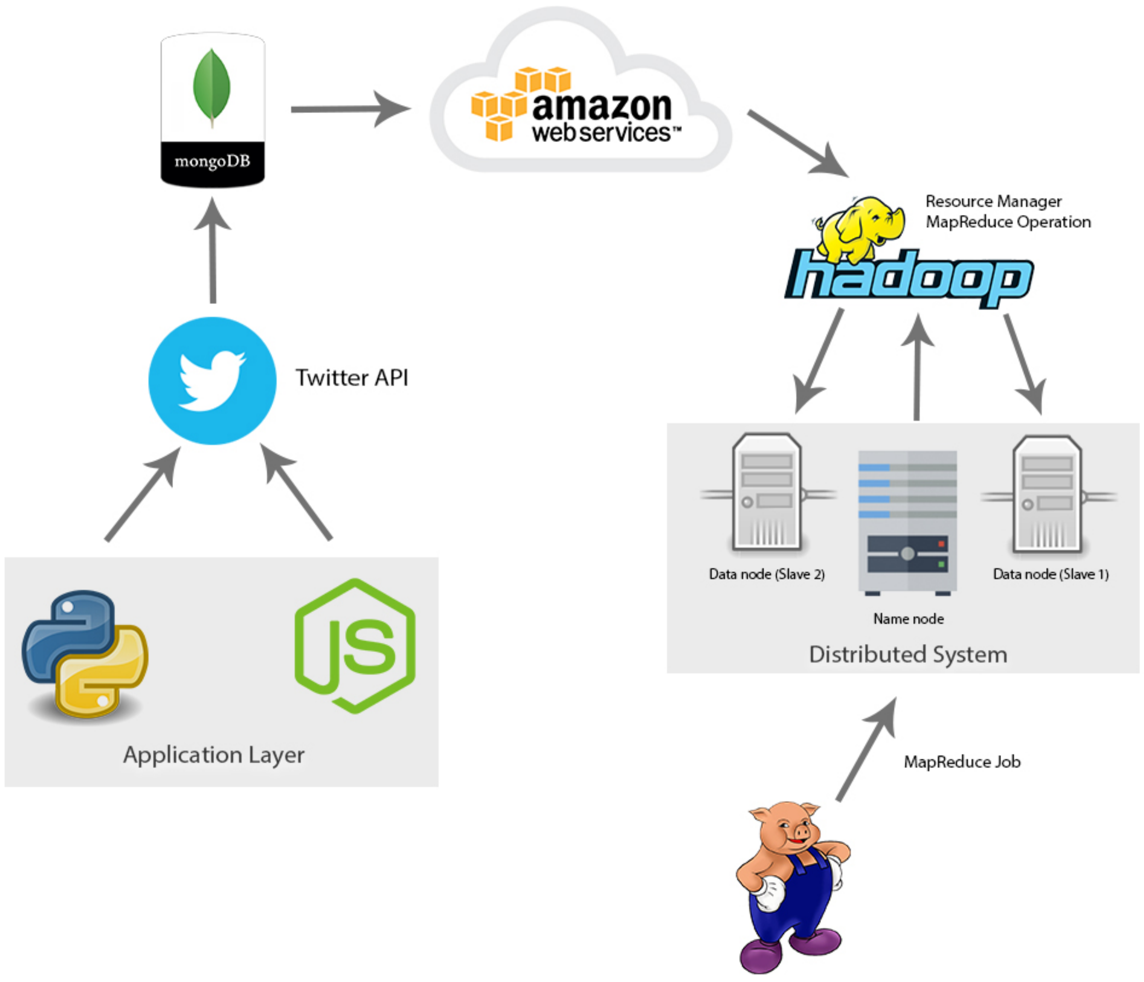
A Introduction

The importance of Distributed Computing is increasing a lot nowadays. It refers to a large collaboration between networked processing units that allows for their processing capacity to be put at the service of a large problem. Many systems and applications are being distributed for a variety of reasons: fault-tolerance, processing performance, security as well as geographical spreading of the data or the problem requirements.

In this project, we were asked to deploy an end-to-end big data harvesting and analysis chain which includes mining,processing and analysis of big data. The data that are being used, are taken from Twitter by using its public API to access them. These data, are tweets that represent million of opinions in many different topics. By deploying a Hadoop cluster and using MAP-REDUCE, we had to keep the most popular 10 hash-tags and words used as well as to count the number of occurrences of each word in the dataset. By applying what was asked, a better understanding of how distributed computing works and the advantages if offers was achieved.

B Implementation Architecture

In this project, we tried to go further with the implementation by combining concepts that we learned in previous courses of the master, such as Web technologies and Databases. We tried to simulate in a ”real case” environment, where several devices with different applications or different back-end (e.g. mobile application in Python and web application in Nodejs) are used to collect a specific information and save it in a pre-defined database.



Source: self-made
Figure 1: Distributed Computing Implementation.

In the Figure 1, the global implementation that we used is Illustrated. At the bottom level, it presents how two different application in Python and Nodejs uses the Twitter API to collect and save tweets in a MongoDB database. The idea behind this implementation is to illustrate how information could be retrieve from different sources and be processed as one data source. Consequently, the information is saved in a MongoDB database in an Amazon Web Server that is later connected with Hadoop distributed storage.

At the same time, in Amazon Web server are three EC2 instances running with Hadoop with one NameNode as a master, assigning work to two slave nodes working as DataNodes to process the MapReduce task in parallel. Finally to create the MapReduce job, PIG framework (cite??) was implemented and installed in the name node as high-level platform which allows to use MapReduce tasks in a simple way, implementing a familiar SQL syntax.Finally the results are stored in the NameNode server and displayed in a simple HTML file.

C Application layer

The application layer is related with the way how the data are retrieved and saved in the database by using Python and JavaScript to access the Twitter API. In order to avoid the same implementation in two different programming languages, we decided to implement the solutions in two different scenarios. One implementing JavaScript without data processing and a second solution with Python with the respective data cleaning and preprocessing, to visualize how the results of the top ten words and hashtags may change with a "clean" and "noisy" data. This comparison would *show* clearly *the importance of having clean data* and the impact it might have to the final results if the data is not preproccessed before.

The code implemented for the Python implementation can be find in the github repository:

The code implemented for the Nodejs implementation can be find in the github repository:

C.1 Data Pre-processing

The data preprocessing part is made by a function that receives the streamed tweet from the Twitter API and it tokenizes the tweet in words, to later remove the stop words, special characters, links, re-tweets and user tags. This step is required in order to extract the main words that potentially represent the meaning or topic of the tweet. Aditionally, Lemmatization is applied in the line *14* to convert every word to the root form, with the goal of grouping all the words that could semantically mean the same.

```
1            def cleanTweet(tweet):
2                wordnet_lemmatizer = WordNetLemmatizer()
3                stop = set(stopwords.words('english')) #stop words!
4                framents = tknz.tokenize(tweet)
5                clean_fragments = []
6                for f in framents:
7                    if f not in stop: # not included in the stop words
8                        f = f.lower() #lowercase fragment
9                        f = re.sub(r'[.,"!~_:|?\']+', '', f,flags=re.MULTILINE) # Special characters
10                       f =  re.sub(r'\.\.\.', '', f,flags=re.MULTILINE)) # 3 dots
11                       f = re.sub(url_expression, '', f,flags=re.MULTILINE) # links
12                       f = re.sub(r'@[a-z,A-Z,0-9 ]*', '', f, flags=re.MULTILINE) # clean at person references
13                       f = re.sub(r'RT @[a-z,A-Z]*: ', '', f, flags=re.MULTILINE) # Remove retweets
14                       f = wordnet_lemmatizer.lemmatize(f)
15                       if f:
16                           clean_fragments.append(f)
17               return(" ".join(clean_fragments))
18
```

Listing 1: Python Cleaning Function

C.2 Twitter API Implementation

Connecting with MongoDB, implementation in Python.

```
            class listener(StreamListener):

    def on_data(self, data):
        #This is the meat of the script...it connects to your mongoDB and stores the
    tweet
        try:
            client = MongoClient('localhost',27017)
            # twitterdb is the new db that I created to store the tweets
            db = client.clean_tweets


            # Decode the JSON from Twitter
            datajson = json.loads(data)
            #Get just the tweet
            tweet = datajson['text']
            # apply the cleaning function to the tweet
            clean_tweet = cleanTweet(tweet)
            # Python Object that would be save in the db.
            pyObject = { 'text': clean_tweet, 'fullResponse': datajson }

            #This is optional, I saw it as a suggestion from a website.
            #grab the 'created_at' data from the Tweet to use for display
            #print out a message to the screen that we have collected a tweet
            #print("Tweet collected at " + str(created_at))

            #It will insert into 'tweets' collection the data that are streamed
            db.tweets.insert(pyObject)
        except Exception as e:
           print(e)

    def on_error(self, status):
        print(status)

auth=OAuthHandler(consumerkey,consumersecret)
auth.set_access_token(accesskey,accesssecret)
twitterStream=Stream(auth,listener())
twitterStream.filter(locations=[-122.995004, 32.323198,-67.799695, 49.893813])


```

Listing 2: Python Implementation

Connecting with MongoDB, implementation in Nodejs.

```
        const Twitter = require('twitter');
const credentials = require('./credentials');
const tweet = require('./tweet');
const mongoose = require('mongoose');

// Connection URL
const url = 'mongodb://localhost:27017/distributedComputing';

const client = new Twitter({
    consumer_key: credentials.keys.api_key,
    consumer_secret: credentials.keys.api_secret,
    access_token_key: credentials.keys.access_token_key,
    access_token_secret: credentials.keys.access_token_secret
  });


```

```
let stream = client.stream('statuses/filter', {locations: '-122.995004, 32.323198,
  -67.799695, 49.893813'});
stream.on('data', (event) => {
  console.log(event && event.text );
  const tweetObject = new tweet( {
      text: event.text,
      fullResponse: event
  });

  tweetObject.save();
});

```

Listing 3: JAvascript Implementation

D Server Configuration

As explained in Implementation Architecture section and presented in the Figure 1, the server consist of one NameNode and two DataNodes running in different Linux instances in Amazon Web Services.

Besides the security rules and after the Hadoop and Java installation, it was important to configure five files:

- **hadoop-env.sh:** This file contains some environment variable settings used by Hadoop. Frequenly use these to modify some aspects of Hadoop daemon behavior, such as where log files are stored, the maximum amount of heap used etc.
- **core-site.xml:** contains the configuration of the name node and the temporal folder where the data is stored during the MapReduce process.

```
                    <?xml version="1.0" encoding="UTF-8"?>
                    <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
                    <configuration>
                    <property>
                      <name>fs.default.name</name>
                      <value>hdfs://namenode:8020</value>
                    </property>
                    <property>
                      <name>hadoop.tmp.dir</name>
                      <value>/home/ubuntu/hdfstmp</value>
                    </property>
                  </configuration>

```

Listing 4: core-site.xml

- **hdfs-site.xml:** contains the configuration for HDFS daemons, the NameNode, SecondaryNameNode and data nodes.

```
                    <?xml version="1.0" encoding="UTF-8"?>
                    <?xml-stylesheet type="text/xsl" href="configuration.xsl"
?>
                    <configuration>
                    <property>
                      <name>dfs.replication</name>
                      <value>2</value>
                    </property>
                    <property>
                      <name>dfs.permissions</name>
                      <value>false</value>
                    </property>
                  </configuration>
```

```

```

Listing 5: hdfs-site.xml

- **mapred-site.xml:** contains the configuration settings for MapReduce daemons; it only need to specify in Hadoop that the yarn framework will be use for the mapreduce jobs.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
</configuration>

```

Listing 6: mapred-site.xml

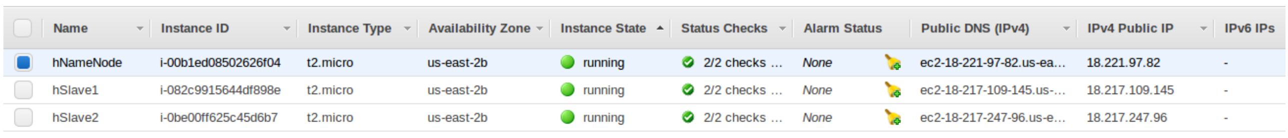
- **yarn-site.xml:** the properties in this file specify where the ResourceManager is running so that other processes can connect to it.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>172.31.19.73:8031</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>172.31.19.73:8032</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>172.31.19.73:8030</value>
  </property>
  <property>
    <name>yarn.resourcemanager.admin.address</name>
    <value>172.31.19.73:8033</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>172.31.19.73:8088</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>

```

Listing 7: yarn-site.xml

All the configuration files could be find in the github repository.

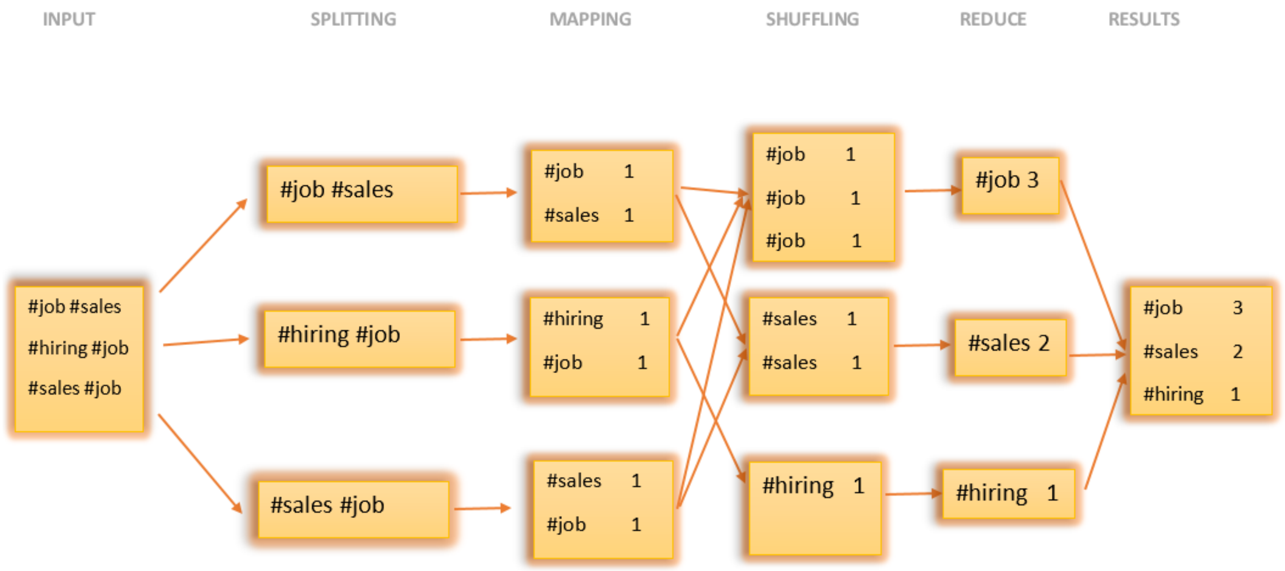


	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs
■	hNameNode	i-00b1ed08502626f04	t2.micro	us-east-2b	● running	✓ 2/2 checks ...	*None*	ec2-18-221-97-82.us-ea...	18.221.97.82	-
☐	hSlave1	i-082c9915644df898e	t2.micro	us-east-2b	● running	✓ 2/2 checks ...	*None*	ec2-18-217-109-145.us-...	18.217.109.145	-
☐	hSlave2	i-0be00ff625c45d6b7	t2.micro	us-east-2b	● running	✓ 2/2 checks ...	*None*	ec2-18-217-247-96.us-e...	18.217.247.96	-

Source: Screenshot idk.
Figure 2: AWS instances.

E MapReduce Job

MapReduce is a framework that is used to process large amount of data in a distributed fashion over several machines. The core idea that stands behind MapReduce is mapping the data into a collection of ¡key,value¿ pairs and then reducing over all pairs with the same key. If we would have to explain it in more detailed steps, we would say that in the beggining the framework will split the data into segments passing each segment to a differen machine. Each machine then runs the map script on the portion of the data attributed to it. As it is already mentioned above, the map script takes the input data and maps it to ¡key,value¿ pairs according to the specifications that we give. In our case, the map script consisted of ¡word,count¿ to be our ¡key,value¿ pairs. Here it is very important to mention that we do not have aggregation since this is the task of the reduce script. The emitted ¡key,value¿ pairs are then shuffled which means that pairs with the same key are grouped and passed to a single machine which will then run the reduce script over them. The reduce script will take the ¡key,values¿ pairs and 'reduce' them according to the specified reduce script. In our word count example, we want to count the number of word occurrencies so on our reduce script we simply sum the values of the collection ¡key;value¿ pairs which have the same key. This is shown in figure 3.



Source: self-made
Figure 3: MapReduce process.

To build the MapReduce Job, Apache Pig was used to take advantage of its easy SQL oriented approach to program, increasing the productivity and friendly for programmers that have not use Java before. Ten line of code in Pig Latin are equivalent almost to 200 lines of code in Java. (http://blog.cloudera.com/wp-content/uploads/2010/01/IntroToPig.pdf)

Apache Pig is then installed in the name node and connected to **YARN** to manage the job request.

Below is provided the job use to calculate the top ten used words and the top ten hashtags. From a global overview, the COUNT in Apache Pig is equivalent to a reduce and the Filter is translated to a map. Additionally the tokenize also represent a map function that split a string of words into a bag of words.

```
        REGISTER /home/ubuntu/hadoop/share/hadoop/common/lib/mongo-java-driver-3.6.0.jar;
        REGISTER /home/ubuntu/hadoop/share/hadoop/common/lib/mongo-hadoop-pig-2.0.2.jar;
        REGISTER /home/ubuntu/hadoop/share/hadoop/common/lib/mongo-hadoop-core-2.0.2.jar;

        data = LOAD '/home/ubuntu/distributedComputing/tweets.bson'
            USING
            com.mongodb.hadoop.pig.BSONLoader('fullResponse','text') AS (text:chararray);

        words = FOREACH data  GENERATE FLATTEN(TOKENIZE(text)) as word;
        grouped = GROUP words BY word;
        wordcount = FOREACH grouped GENERATE group, COUNT(words);

        ordered = ORDER wordcount by $1 DESC;
        top_words = LIMIT ordered 10;
        hash_filter = FILTER ordered BY $0 MATCHES '.*\\#\\p{Alpha}.*?';

        top_hash = LIMIT hash_filter 10;

        STORE top_hash INTO '/home/ubuntu/pig/scripts/top_10_hash';
        STORE top_words INTO '/home/ubuntu/pig/scripts/top_10_words'


```

Listing 8: Tweet cleaning function

The first three register statements in the Pig Script is implementing the Java, MongoDB and Hadoop connectors to work across the the architecture defined for the 3 instances. Then the BSON file from the database is loaded and the tweet is saved as "text". In the following lines, the tokenize divides the tweets into words (map) to consequently group them. From that point it is needed to count the amount of times that a word is present in the all the tweets, using COUNT (filter).

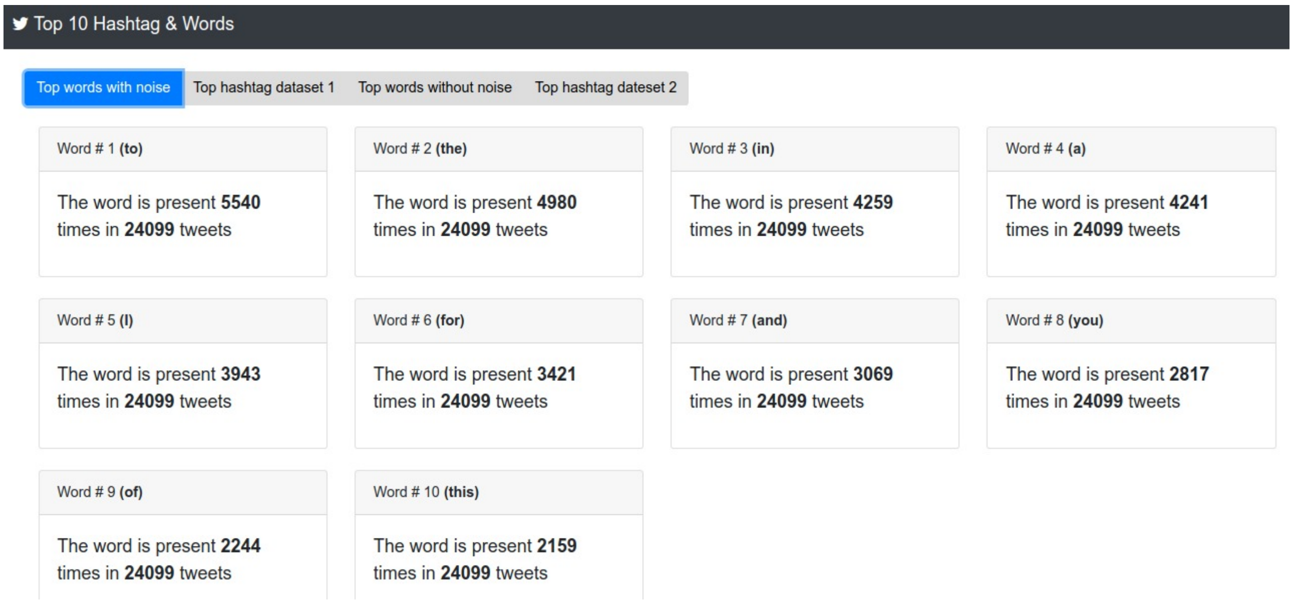
With the pair word and number of repetitions of the word, is just matter of order them in descended order to identify the top ten words and filter (reduce) all the words that start by the character # to extract also the top ten hashtags.

Finally with the STORE command, the results are locally saved in the server. ** Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets. (https://pig.apache.org/)

** YARN is the architectural center of Hadoop that allows multiple data processing engines such as interactive SQL, real-time streaming, data science and batch processing to handle data stored in a single platform, unlocking an entirely new approach to analytics. https://hortonworks.com/apache/yarn/

F Job Results

In figure 4, are shown on a screenshoot taken from a html file, the top 10 words that were retrieved from the un-processed data.



Word # 1 **(to)**
The word is present **5540** times in **24099** tweets

Word # 2 **(the)**
The word is present **4980** times in **24099** tweets

Word # 3 **(in)**
The word is present **4259** times in **24099** tweets

Word # 4 **(a)**
The word is present **4241** times in **24099** tweets

Word # 5 **(I)**
The word is present **3943** times in **24099** tweets

Word # 6 **(for)**
The word is present **3421** times in **24099** tweets

Word # 7 **(and)**
The word is present **3069** times in **24099** tweets

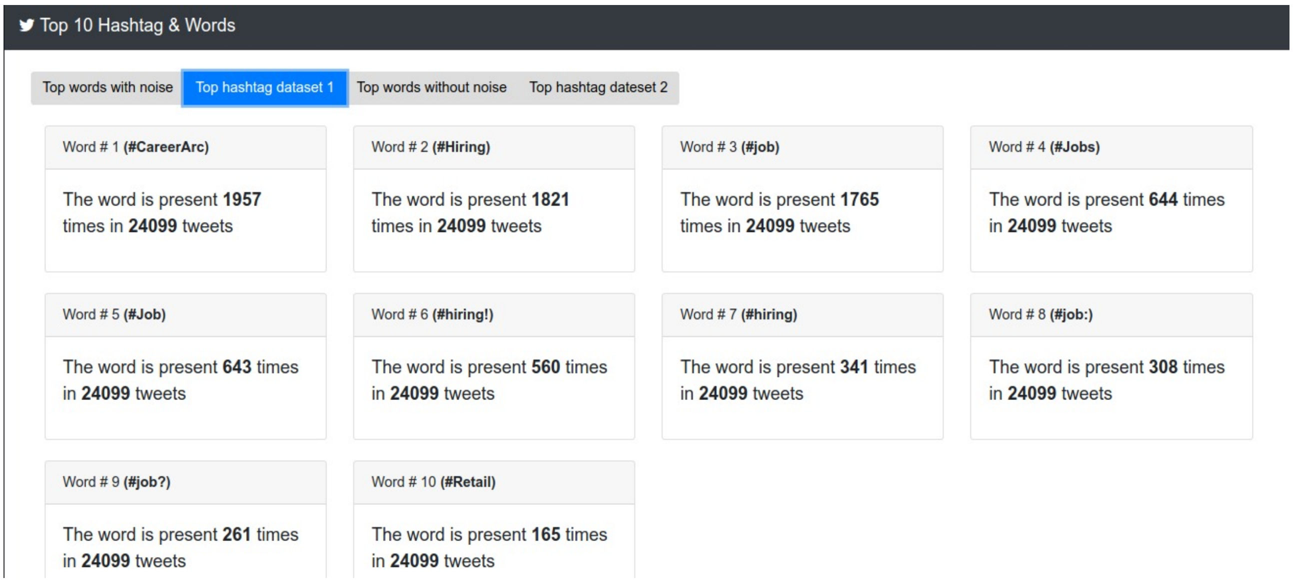
Word # 8 **(you)**
The word is present **2817** times in **24099** tweets

Word # 9 **(of)**
The word is present **2244** times in **24099** tweets

Word # 10 **(this)**
The word is present **2159** times in **24099** tweets

Source: self-made
Figure 4: Top Words with Noise

In figure 5, are shown the top 10 hashtags that were retrieved from the un-processed data.



Word # 1 **(#CareerArc)**
The word is present **1957** times in **24099** tweets

Word # 2 **(#Hiring)**
The word is present **1821** times in **24099** tweets

Word # 3 **(#job)**
The word is present **1765** times in **24099** tweets

Word # 4 **(#Jobs)**
The word is present **644** times in **24099** tweets

Word # 5 **(#Job)**
The word is present **643** times in **24099** tweets

Word # 6 **(#hiring!)**
The word is present **560** times in **24099** tweets

Word # 7 **(#hiring)**
The word is present **341** times in **24099** tweets

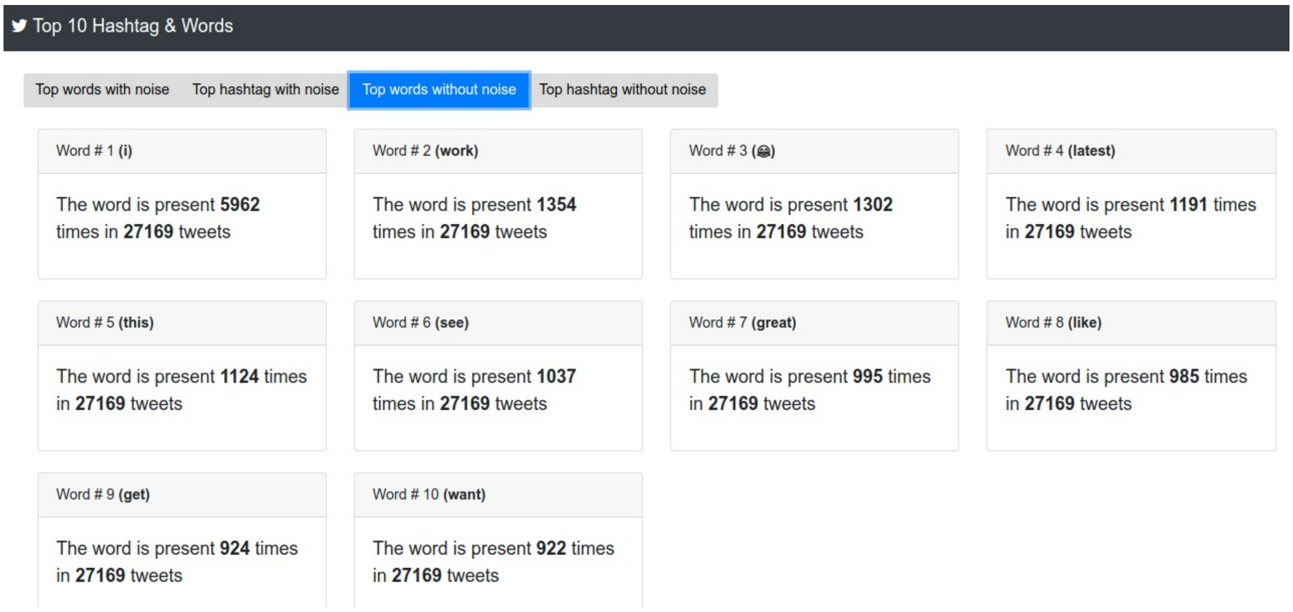
Word # 8 **(#job:)**
The word is present **308** times in **24099** tweets

Word # 9 **(#job?)**
The word is present **261** times in **24099** tweets

Word # 10 **(#Retail)**
The word is present **165** times in **24099** tweets

Source: self-made
Figure 5: Top Hashtag Dataset 1

In figure 6, are shown the top 10 words that were retrieved from the pre-processed data.


Top 10 Hashtag & Words

Word # 1 **(i)**
The word is present **5962** times in **27169** tweets

Word # 2 **(work)**
The word is present **1354** times in **27169** tweets

Word # 3 **(😂)**
The word is present **1302** times in **27169** tweets

Word # 4 **(latest)**
The word is present **1191** times in **27169** tweets

Word # 5 **(this)**
The word is present **1124** times in **27169** tweets

Word # 6 **(see)**
The word is present **1037** times in **27169** tweets

Word # 7 **(great)**
The word is present **995** times in **27169** tweets

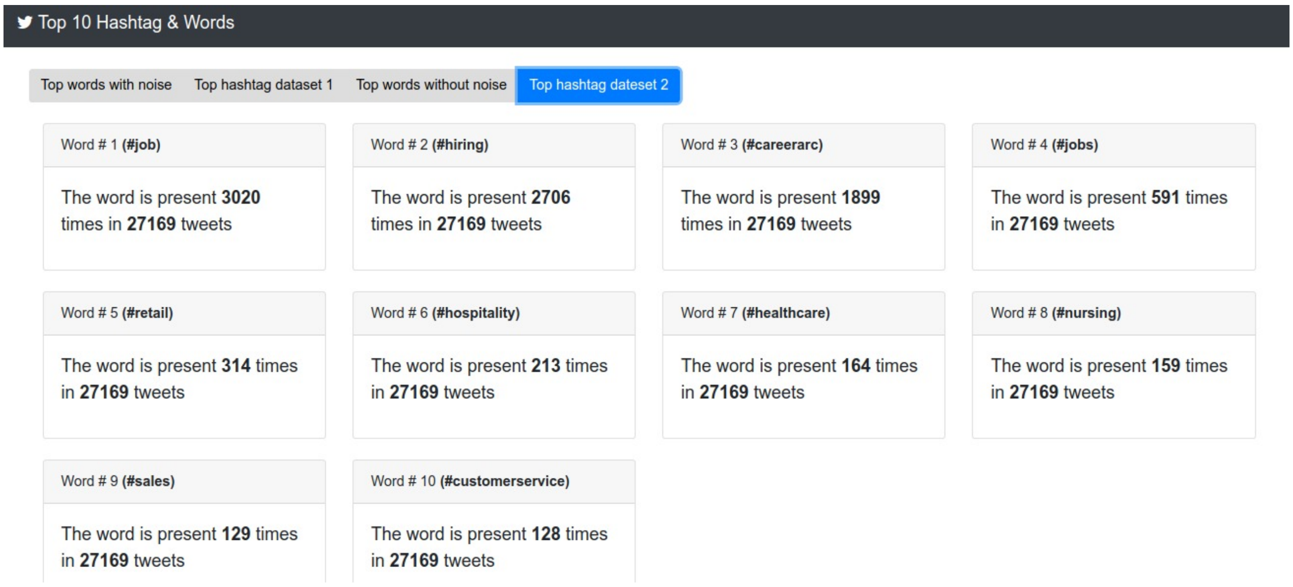
Word # 8 **(like)**
The word is present **985** times in **27169** tweets

Word # 9 **(get)**
The word is present **924** times in **27169** tweets

Word # 10 **(want)**
The word is present **922** times in **27169** tweets

Source: self-made
Figure 6: Top Words without Noise

In figure 7, are shown the top 10 hashtags that were retrieved from the un-processed data.


Top 10 Hashtag & Words

Word # 1 **(#job)**
The word is present **3020** times in **27169** tweets

Word # 2 **(#hiring)**
The word is present **2706** times in **27169** tweets

Word # 3 **(#careerarc)**
The word is present **1899** times in **27169** tweets

Word # 4 **(#jobs)**
The word is present **591** times in **27169** tweets

Word # 5 **(#retail)**
The word is present **314** times in **27169** tweets

Word # 6 **(#hospitality)**
The word is present **213** times in **27169** tweets

Word # 7 **(#healthcare)**
The word is present **164** times in **27169** tweets

Word # 8 **(#nursing)**
The word is present **159** times in **27169** tweets

Word # 9 **(#sales)**
The word is present **129** times in **27169** tweets

Word # 10 **(#customerservice)**
The word is present **128** times in **27169** tweets

Source: self-made
Figure 7: Top Hashtag Dataset 2

G Conclusion

In this project, by using Twitter API we managed to have access to a large amount of tweets (27169 tweets retrieved using Python and 24099 tweets retrieved using Javascript). A very important step was the data cleaning since it leads to different results if the process is not done correctly. As it was also described in the project requirements, we splitted the raw data into meaningful components by taking only the root of the words, removing punctuation marks ecc.

As a storage, we used MongoDB since we stored all the data there.This information is saved in an Amazon Web Server that is later connected with Hadoop distibuted Storage. Hadoop has become a key tool for managing pools of big data and supporting big data analytics applications.It takes the data,breaks the information down into separate pieces and distributes them to different nodes in a cluster, in our case in 2 slave nodes, allowing for parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. The best advantage that we profit from using it, is the time. We processed more than 20000 tweets in less than 5 seconds.

To express more clearly the difference between having cleaned and uncleaned data, we decided to present two types of scenarios, one where we used JavaScript to retrieve the data and we did not preprocess them, and one where we used Python to retrieve and preprocess. The differences are shown clearly. When the data is unprocessed and we apply MAP-REDUCE, the result we get when we want to find the top 10 most used words and the number of times they are found, influences the effectiveness of the whole process since we get as a result stop words that does not express the core idea of the task. Meanwhile, when the data was preprocessed, the words that we got as a result were meaningful and they expressed exactly the most used words among all the tweet. We would like to empathize that we gained a lot of practical experience and very good knowledge while working for this project.

H Group member contribution

For this project, we have had a very good coordination together. We have been working mostly in the lab in order to realize it and have discussed every difficulty and idea together. Regarding the contribution of each of us, Juan has been the one that have set up the infrastructure of the server and doing the implementation in Javascript meanwhile Redona has been working more on the Python Implementation.

I Difficulties encountered

We have encountered one difficulty that we could not solve it. It has to do with the data pre-processing part. We had removed the triple dots (”...”) as you can also see in section C1 at the code, line 10 but actually the triple dots were showing at the results. We also tried another way to express it after searching for this problem (f=re.sub(”[]̇”*3, ” ”)) but still without success. We had to implement manually the replacement of the dots in the text with empty spaces. What was strange for us, is the fact that even if we filtered the data to remove the single dot, it was still showing when we had triple ones. Logically it should work even when we have only one dot (concatation of 3 dots). For the rest of difficulties we have encountered, we managed to solve them.

References

[1] G. J. Alred, C. T. Brusaw, and W. E. Oliu, *Handbook of Technical Writing.* New York: St. Martin's, 2003 (seventh edition).

[2] M. Goossens, F. Mittelbach, and S. Rahtz, *The LaTeX Companion.* Reading, Mass.: Addison-Wesley, 1997.

[3] F. Mittelbach, M. Goossens, J. Braams, and D. Carlisle, *The LaTeX Companion.* Reading, Mass.: Addison-Wesley, 2004 (second edition).

[4] S. F. Gull, "Developments in maximum-entropy data analysis," in *Maximum Entropy and Bayesian Methods* (J. Skilling, ed.), pp. 53–71, Kluwer Academic, Dordrecht, 1989.

[5] K. M. Hanson, "Introduction to Bayesian image analysis," in *Medical Imaging: Image Processing* (M. H. Loew, ed.), vol. 1898 of *Proc. SPIE*, pp. 716–731, 1993.

[6] L. Lamport, *LaTeX: A Document Preparation System.* Reading, Mass.: Addison-Wesley, 1994.

[7] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of state calculations by fast computing machine," *J. Chem. Phys.*, vol. 21, pp. 1087–1091, 1953.

[8] L. C. Perelman, J. Paradis, and E. Barrett, *Mayfield Handbook of Technical and Scientific Writing.* Mayfield: Mountain View, 1997. http://mit.imoat.net/handbook/.