**Project Report**

**On**

**Real Time Data Pipelines Using Spark-streaming**

California State University Fullerton

Department of Computer Science & Engineering,

Fullerton, California

Kashyap Sodha, Shruti Choudhary, Sajiri Mehdjoge, Shubhshree Anand

Abstract— In recent years, analyzing data streams has attracted considerable attention in different fields of computer science. Spark is an open-source cluster computing framework for big data processing [1]. In this Project, Twitter API will be used as a message queue to extract the real-time data streams which will act as a trigger to Spark framework where the data streams will be analyzed and processed based on various criteria. Also, Spark streaming makes it feasible to build scalable, robust stream processing applications. Spark streaming solves the problem of real-time data processing. Once the data has been analyzed and processed, the generated output will be accumulated in a persistent storage and displayed on a dashboard where basic functions will be to publish and monitor data for a specific time span.

Keywords— Spark Streaming, Data pipelining, Data Integration, Stream Processing, Python, Twitter API

# Introduction

## Big Data:

Big data is defiantly a hot topic in the data industry. It has gained enough attention to change the approach towards the data storage mediums of the people around the world. It is defined in variety of ways by different individuals and organizations. It’s popular definition or often known to be characteristics will be three defining characteristics or dimensions of big data which are 3Vs (volume, variety and velocity) where Volume refers to the amount of data, variety refers to the number of types of data and velocity refers to the speed of data processing. In simple language, it’s a data in incredibly large quantity. It contains combinations of structured, semi structured and unstructured data which is so huge in volume and complex that traditionally used tools and techniques are inadequate to process it [3].

Storing such a large amount of data has been a challenge from long but from recent years the volume of data isn’t the only problems causing compactions. A lot of new data formats are developed recently and moreover the data is in the unstructured format generating a need to add one more task to convert it into a structured being initiating the data processing. Last but not the least the big data is growing incredibly fast and rapidly that is a huge difference in the speed of generation of such a data (required to be processed) and speed of data processing. Hence there still is a need to find better technology that could work with this situation.

## Batch processing vs Real-time data processing:

There are multiple resources of data and of the methods to receive it. Data sets typically has following two types of data as:

* Data at rest:
  + Stagnant data
* Data in motion:
  + Batch processing
  + Real-time stream processing

Stagnant of flat data is one of the simplest data type to process. This kind of data doesn’t exhibit any significant change into it. As the data sixe is fixed here, it’s harder to depict the data without any data variance. Hence, comparatively it’s really easy to process this kind of data [5].

Batch processing is the next concept where the data is collected and stored into the memory for a finite and fix amount of time before processing [4]. It requires the most minimum manual interaction since there is no need of interaction, display or a feedback provided to the user during execution. It is an efficient way to handle and process data in large quantity. It requires three individual processes. First data is collected and stored usually after a specific duration of time, second, it’s processed by appropriate processing technique and third, data is output. Here, a group of one or more programs often referred as a job will perform a particular task or a set of data that at a specific instance of time without requirement of manual observation. A suitable example of this will be any inventory bill system where a bill is set to be generated at specific date and time and doesn’t require special attention. Real-time data processing is the execution of data in a short timeframe period, giving close momentary output. In other words, the real-time data processing has the requirements of continuous flow of input of data, continuous processing of that and a steady output. The processing is done as the data is inputted, so it needs a persistent stream of data so as to give a consistent and continuous output. It allows the users to be able to take immediate action in all those items whenever an action within minutes or even seconds is significant.

Real time data processing is utilized by Point of Sale (POS) Systems to update the inventory, give its history, and offers of a specific thing - enabling an organization to run the desire payment in real time. Real-time data processing is otherwise called stream processing because it requires a continuous stream of input data for processing so that the output can be obtained for that moment. The advantages of this type of processing would be as said the data is processed not only immediately but also repetitively. Near real time processing is also known to be another processing techniques. The processing speed matters equally or carries the same importance over here. Good examples of real-time data processing frameworks are bank ATMs, movement control frameworks and current PC frameworks, for example, the PC and cell phones \ the continuous stream of data is generated in a random time interval and processed accordingly. Interestingly, data and after that procedures every one of the data in mass in a later time, which likewise implies output is gotten at a later time.

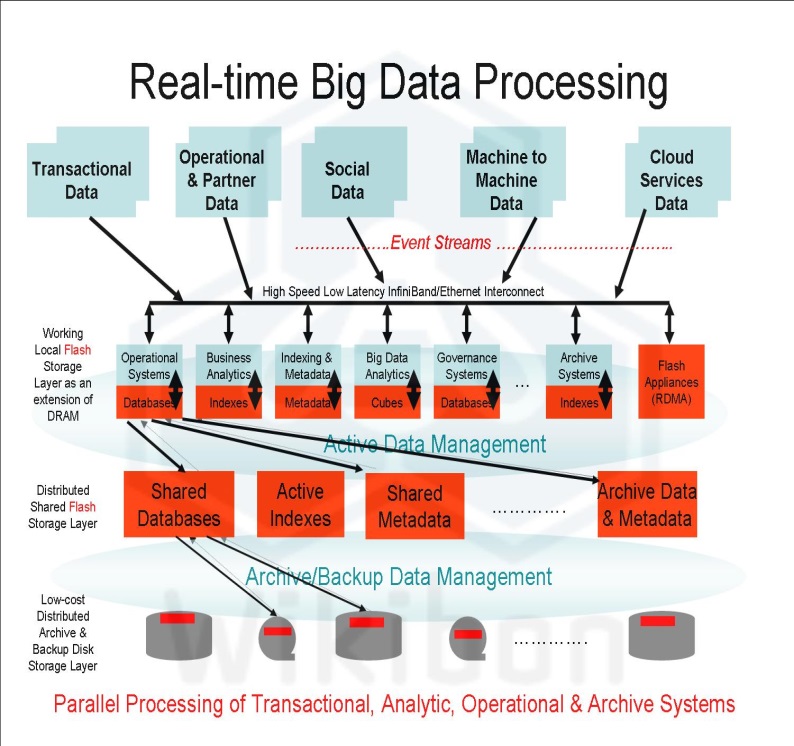


Fig1: Real-Time Big Data Processing

## Flaws in Traditional Systems

Big data processing requirement is not just means getting bigger data centers to store the data chunks. It’s more about using the most or at least more efficient technique to process and manage massive number of streams with faster speed and less complex. Hence the need is emerged for more advanced approaches to deal with this overwhelming amount of data that’s has to be analyzed.

# Data Processing

## **Data Input Technology**

**Twitter API**- In our project, we will be fetching the real-time data from an open source, and then we will process that data on spark. Here, we will fetch the real-time tweets from the Twitter [6]. Various tools and API’s are offered by the Twitter’s developer. The publisher tools like “Twitter for website” and “Twitter Kit” makes it feasible to show tweets on the web and on the Mobile App.

***To display the twitter content on the Web and the App:***

The “Twitter for website” and “Twitter Kit” are the publisher tools used to display the tweets and the stories on the Web and the App. To find and curate the content, the TweetDeck is being used. It brings the relevant, real-time and engaging content for the users.

1.*To find and curate Tweets:*

TweetDeck is being used to find and organize the tweets. The searches for various trending hashtags and interesting topics can be created and then the results can be filtered by the location, engagement and users. TweetDeck is like the twitter content hub, as it has number of columns for the timelines or earlier tweets used on the Web and the App.

2.*Embedding the content on the Web:*

For embedding a tweet on the web or even an entire column timeline, there are options available in the menu.

3.*Showing the content in the Mobile App:*

Twitter Kit SDK is used to show the twitter content in the Mobile App. Tweets and timeline views can also be created to use in the IOS or Android App.

***To build an app on Twitter:***

The Twitter’s API platform helps in building an app on Twitter by providing number of free available endpoints. Twitter’s basic REST and streaming API provides such free access to many endpoints.

1*.Creating an app*: An app must be created first and the OAuth-based authorization system in order to use any endpoint.

2.*Using endpoints:* To use the endpoints, one must setup an account. For different programming languages, many utilities and libraries are available.

## **Apache Spark**

Apache Spark is an exceptionally fast cluster computing innovation, intended for quick calculation. It depends on Hadoop MapReduce and it stretches out the MapReduce model to effectively utilize it for more sorts of computations, which incorporates interactive queries and stream preparing. The fundamental component of Spark is its in-memory cluster computing that expands the preparing rate of an application.

Spark is intended to cover an extensive variety of workloads, for example, batch applications, iterative algorithms, interactive queries and streaming. Aside from supporting all workload in an individual framework, it diminishes the administration weight of keeping up particular instruments.

***Introduction to Spark Streaming****:*

The architecture for Spark Streaming focusses on programming benefits for spark developers owing to its ever-growing user base- Amazon, Pinterest, Yahoo, Netflix, eBay and Uber. 50% of spark users has mentioned that Spark Streaming is most used and essential spark component. Apache Spark is a technology in Big Data which is remarkably important and worth taking note of. Various companies working with big data are using spark streaming component to enhance business productivity. So the answer to the question “what is spark streaming” is that it is a data processing framework to build up streaming applications.

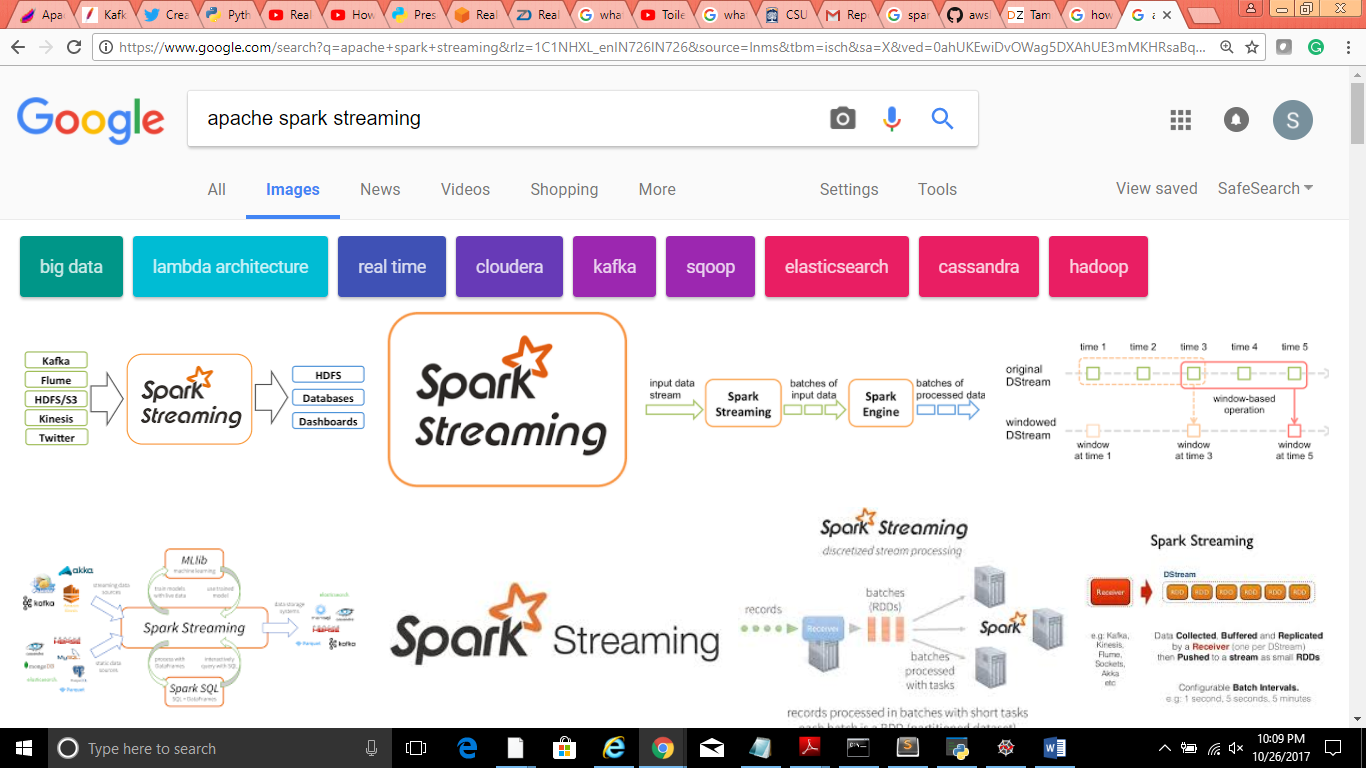


Fig 2: Project Workflow

***Requirement for Spark Streaming:***

Spark Streaming has gained huge popularity and attention in big data computation industry. Earlier programmers use to create two stacks, where one was for batch processing and another for streaming to process that data. The preexisting frameworks for processing were not sufficient to achieve both either they could perform batch processing of 100 TB of data with very high latency or could perform stream processing of 100's of terabytes of data with low latency. This was really challenging for developers to maintain dual effort for implementation and operational effort. This move to encourage both stream processing and batch processing was not at all easy for even fast flying web companies. The main purposes to use spark streaming are:

* Highly Scalable
* High Level Language operators for Data Streaming
* Modular and Simpler
* Support for Data Merging to Historical Data
* Fault-Tolerant & Easy for Code Reuse

***Architecture for Spark Streaming:***

Spark has Resilient Distributed Datasets(RDD) which maintain a lineage graph of how every partition of data is created. If there is a failure, it can recreate the data and run the computation again. If data stream is incoming then we can take a sliding wondow and pick a little data in this window and run it like a batch as DStream.

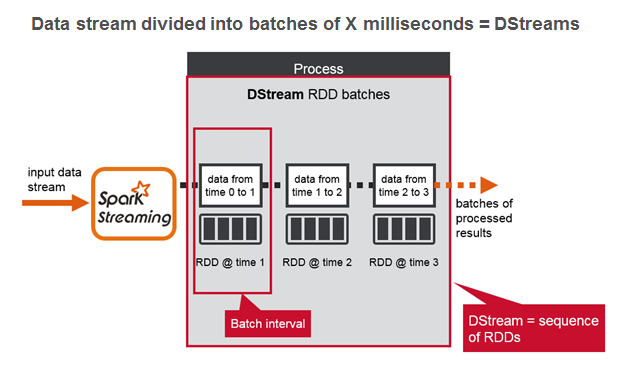


Fig 3: A Complete Workflow from Twitter API to Dashboard

This process can be repeated. In Spark Streaming, key abstraction is Decretized Stream(DStream) built on RDD’s. DStream represents a stream of data distributed into small batches [10]. Spark Streaming calls the live data stream and it is chopped into small batches of y seconds. Spark considers each batch of data as an RDD and process using different RDD operations [9]. The results receieved in batches those will be moved to any other streaming system. Spark Streaming architecture is summed up of three crucial components:

***Master Node*** – It is responsible for tracking the DStream lineage graph and also schedules various tasks to compute any new RDD partitions.

***Client Library*** – Used to send data into the system.

***Worker Nodes***– They receive data, store partitions of the computed RDD’s and execute tasks.

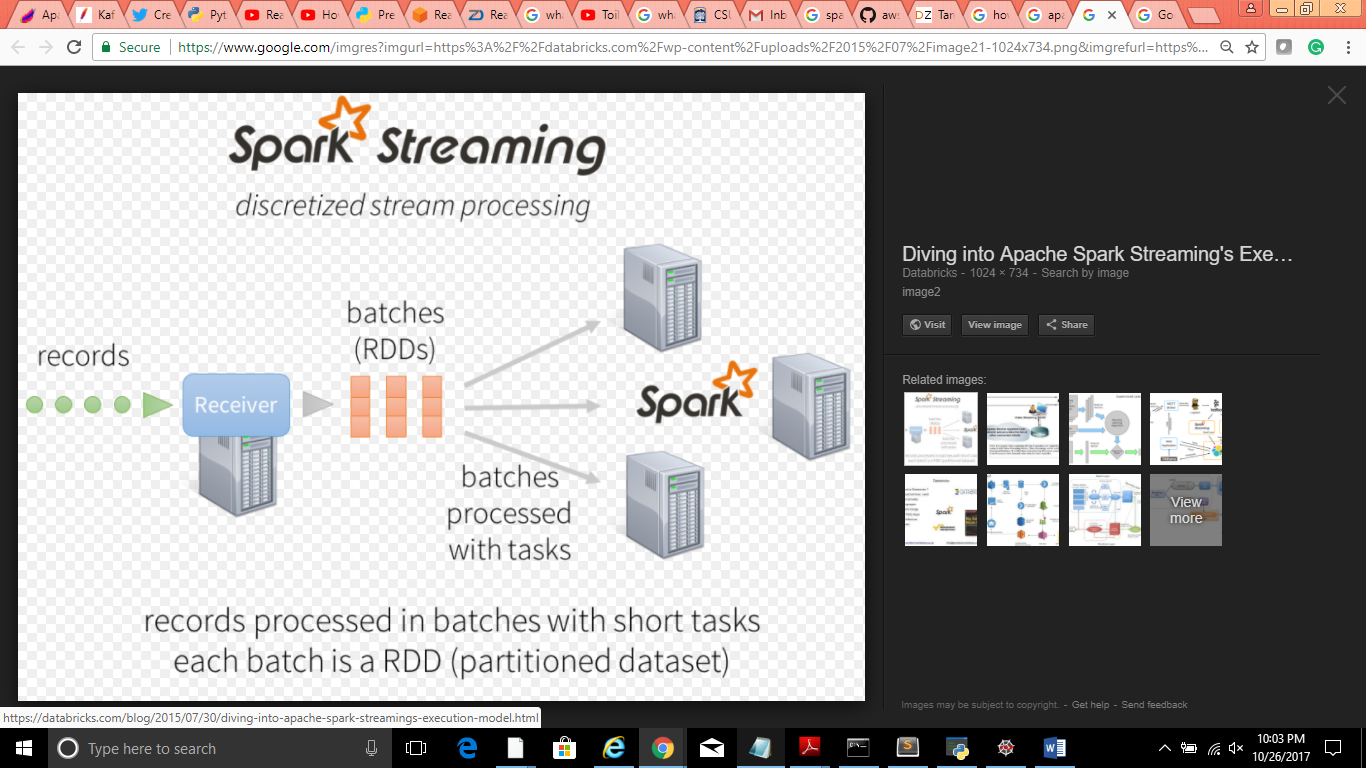


Fig 4: Discretized Stream Processing

Most notable difference betweeen Traditional streaming architecture and Spark Streaming architecture are distributed into short, deterministic tasks and stateless that run on any given node in the spark cluster or on multiple nodes.

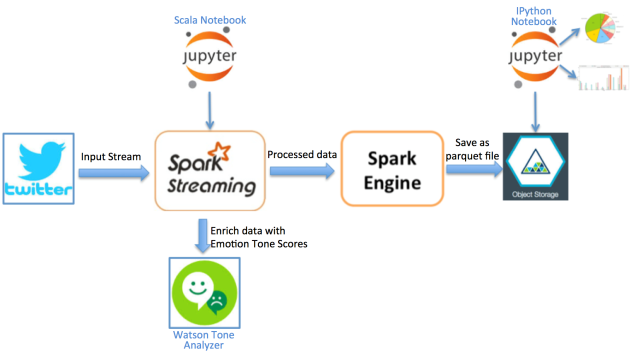


Fig 5: A Complete Workflow from Twitter API to Dashboard

***Data Sources (Possible) for Spark Streaming:***

There are various data sources which can be used as input of data streams:

* TCP Sockets
* ZeroMQ
* Apache Flume
* Twitter (Twitter API)
* Spark Streaming Amazon Kinesis
* Kafka Connect

***Spark Streaming for Real Time Data Pipeline:***

Spark Streaming is an expansion of the core Spark API that empowers versatile, high-throughput, blame tolerant stream preparing of live information streams. Information can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be handled utilizing complex algorithms communicated with abnormal state capacities like guide, lessen, join and window. At long last, prepared information can be pushed out to filesystems, databases, and live dashboards. Truth be told, you can apply Spark’s machine learning and graph processing algorithms on information streams. Inside, it works as shown below. Spark Streaming gets live info information streams and partitions the information into batches, which are then handled by the Spark motor to produce the last stream of results in batches.

## **Data Output technology- Kibana Dashboard**

Kibana is a visualization platform designed to work with Elasticsearch which gives a better understanding of the data. To search, view and interact with the data which is stored in Elasticsearch indices, Kibana is being used. It can execute various operations such as data analysis and visualizing the data in diverse types of charts, maps and tables. Large volumes of data can be made feasible to understand with the help of Kibana. With the simple, browser based interface of Kibana, we can quickly create and share the dynamic dashboards, which will show the changes to Elasticsearch queries in real time [11].

Elasticsearch is an open source search engine based on Apache Lucene. It provides the distributed, scalable search engine with HTTP web interface and schema free JSON documents. Elasticsearch is used as an underlying engine which is used in those kind of appliactions which have complex features and requirements. There are four main sections in the Kibana Interface:

**Discover**

**Visualize**

**Dashboard**

**Settings**

The Kibana dashboard is used to show a group of saved visualizations. To arrange and resize thevisualizations as per our requirements, we can use the edit mode of the dashboard.

# Project Status

## **Summary:**

We will be fetching records i.e., tweets from Twitter API using various keys provided by Twitter. These records will then be passed into Spark Streaming for processing the fetched data. After applying various Stream functions, output of it will be displayed onto a live dashboard which in this case we will be using Chart.js to create a dashboard which will display Hashtag counts as per the categories of fields the tweets are being received in Real-time.

## **Project Work Done:**

Creation of web application was required for using Twitter APIs. Thus, the same was being done in the initial week of the project. In the later weeks of it, understanding the use of Twitter API was being done. Post the understanding of it, implementation of Twitter API, initially started with accessing Twitter using keys provided by the Twitter during creation of web application. Different libraries of Twitter were imported into python to use various functionalities. We have installed spark to work with python language with help of pyspark and here we are keeping the spark as standalone machine to work on Real-time tweet streams from Twitter Api. The Tweets are getting extracted using appropriate methods as **get\_tweets()** and **send\_tweets\_to\_spark()** post successful authentication of user credentials. Data streams are extracted with fewer conditions such as Hashtags count on categories in CSV file format. Various technical errors were also tackled during this period of work progress such as Data streams being converted to RDD’s (Resilient Distributed Datasets). Understanding of Spark has also been accomplished during this period and converting data into RDD Rows. Two kind of context such as Spark context and streaming context will be created from Spark. After processing data in Spark, we have pushed the data onto a dashboard which is created using Chart.js which is a JavaScript library to create an interactive dashboard.

## **Challenges in Project Work Progress**:

Sending the data from Twitter api to Spark processing and Understanding of Spark Streaming process with each RDD creation and defining DStreams as RDD rows were generating data black and then implementation for data extraction from pyspark to live dashboard which is created using Chart.js. We had challenges in understanding the Chart.js and from initial setup to visual display of data results.

# Implementation

## **Accessing Twitter API (Create An Application):**

Initially, a new simple web application needs to be created at twitter apps, post which twitter provides different keys in order to access the online streams in form of tweets from twitter using Python. Currently, using OAuth function credentials are verified and access is granted by twitter to access the tweets. Python language is being used to verify with created credentials and write methods to fetch the data streams. In order to access Tweets, we need to register on Twitter Apps by creating a new application and fill the requested details:

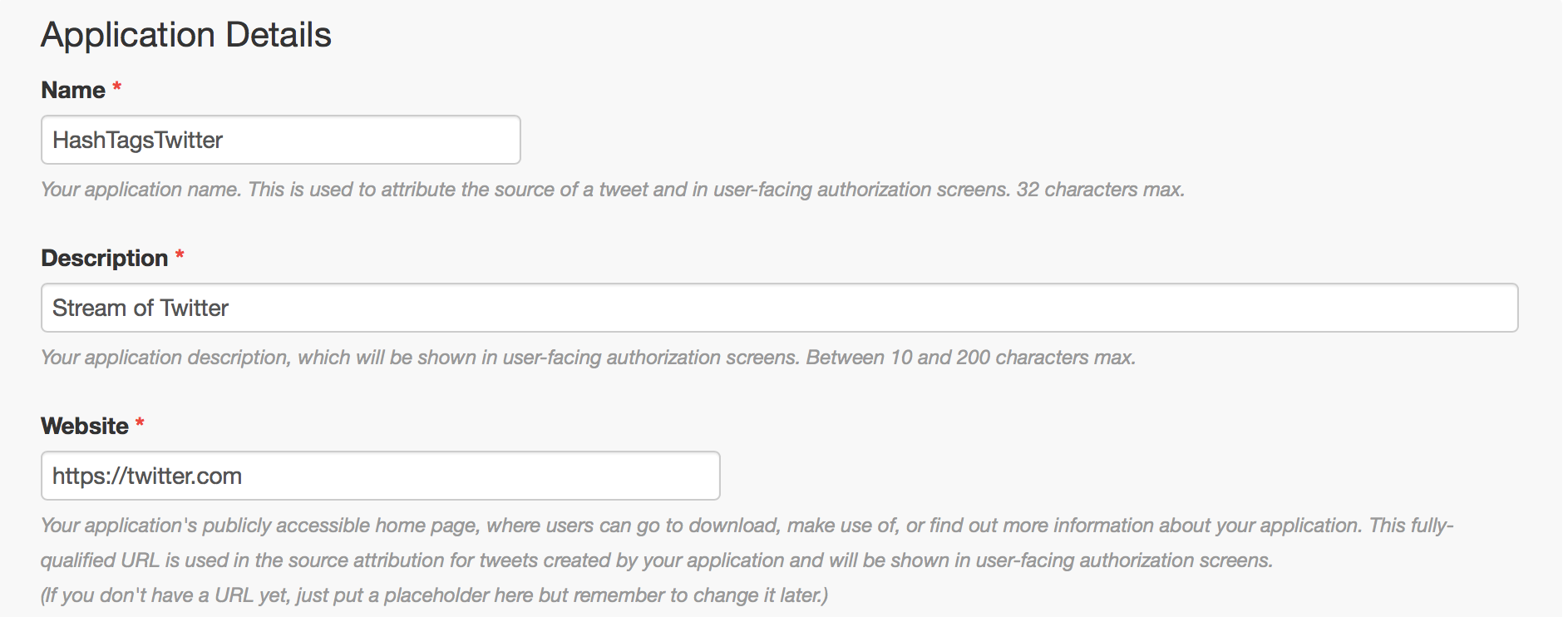


Fig 6: Twitter API –Create new Application

Post creation of these new app we need to get ‘Key and Access Tokens by generating access token. We have access toke, access token secret, consumer key (API key), and consumer secret (API secret) as credentials for fetching tweets from Twitter. There are various stages of implementation of the project which are: verify credentials, get tweets from Twitter using Twitter API, build a local host, feed the data to Spark, perform required processing on Spark, push the output to dashboard.

First stage of implementation includes verification of the credentials and could be implemented in following manner:

**Access\_token = ‘aaa’**

**Access\_token\_secret = ‘bbb’**

**Consumer\_key = ‘ccc’**

**Consumer\_secret = ‘ddd’**

**My\_auth = requests\_oauthlib.OAuth1(consumer\_key,consumer\_secert, access\_token, access\_token\_secret)**

## **Get Tweets from Twitter API**: Next step after verifying credentials is to get tweets from Twitter using Twitter API.

**url = 'https://stream.twitter.com/1.1/statuses/filter.json'**

**query\_data = [('language', 'en'), ('locations', '-122.75,36.8, -121.75,37.8, -74,40, -73,41'), ('track','#')]**

**query\_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query\_data])**

**response = requests.get(query\_url, auth=my\_auth, stream=True)**

**print (query\_url, response)**

## **Extarct Tweets in JSON:**

## Another part of getting tweets is extracting the tweets in JSON object.

**full\_tweet = json.loads(line)**

**tweet\_text = full\_tweet['text']**

**print("Tweet Text: " + tweet\_text)**

**print ("------------------------------------------")**

**tcp\_connection.send(tweet\_text + '\n')**

## **Open TCP connection**:

## The next step is the heart of the project, creation of app host connection. We’ll configure the IP to be ‘localhost’ on the same machine and port ‘9009’. Post creation of TCP connection, we will send the data generated in the below steps to Spark:

**TCP\_IP = "localhost"**

**TCP\_PORT = 9009**

**conn = None**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**s.bind((TCP\_IP, TCP\_PORT))**

**s.listen(1)**

**print("Waiting for TCP connection...")**

**conn, addr = s.accept()**

**print("Connected... Starting getting tweets.")**

**resp = get\_tweets()**

**send\_tweets\_to\_spark(resp,conn)**

## **Setting Up Apache Spark Streaming Application:**

We have built our Spark Streaming app to do real-time processing for the incoming tweets, extracting the hashtags from the tweets and calculate how many hashtags have been mentioned in these tweets.

***Create Spark & Streaming Context****:* First step is to create an instance (sc) of Spark Context and second to create the instance (ssc) of Streaming context from a batch interval in each two second that will do the transformation on all incoming streams received every two seconds. Also, here it is important to state that we have set the log level to “error” to prevent most of the unnecessary logs that Spark writes for incoming streams.

***RDD Creation Check*:** The next step is that we have defined a checkpoint to put checkpoint to verify the creation of RDD (Resilient Distributed Datasets) when data streams are processing continuously and it is very required to be used in our app because we will use stateful transformations. Next, we will define our primary DStream (Descritized Stream) which will connect to the socket server which we had already created earlier on 9009 and read the tweets from the same port. Each record being received as the DStream will be a tweet.

***Transformation Logic- Hashtag Count***: Defining the logic for transformation in our code will be our next step. First, we have to split up all the incoming tweets into words and insert them in RDD (Resilient Distributed Dataset). On these words, we will do processing later and filter only hashtags from all words and map the hashtags to the pair and put them in Hashtag RDD.

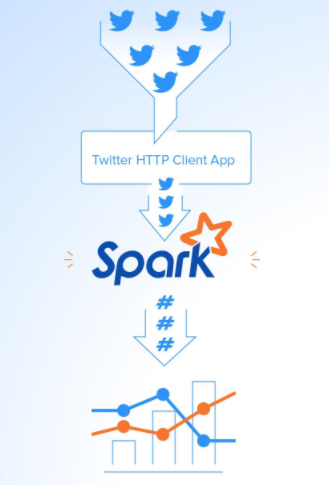


Fig 7: Twitter Client App to Dashboard

Here it is required to calculate the number of occurrence of hashtags. We will be implementing *reduceByKey* to count that how many times the hashtag has been mentioned in each batch of data. It will reset the count for each batch. For our code, we need to calculate the counts across all the batches generated from our data, and we will be using another function called *updateStateByKey* as this function allows us to maintain the current state of RDD while updating this function with new data. This way will be called *Stateful Transformation*. Below code generates RDD rows:

**def process\_create\_rdd(time, rdd):**

**print("----------- %s -----------" % str(time))**

**# Get spark sql singleton context from the current context**

**sql\_context = get\_sql\_context\_instance(rdd.context)**

**# convert the RDD to Row RDD**

**row\_rdd = rdd.map (lambda w:Row(hashtag=w[0], hashtag\_count=w[1]))**

**# create a Data Frameas DF from the Row RDD**

**hashtags\_df = sql\_context.createDataFrame(row\_rdd)**

**# Register the dataframe as table for hashtags**

**hashtags\_df.registerTempTable("hashtags")**

**# get the top 10 hashtags from the table using SQL context and print them here**

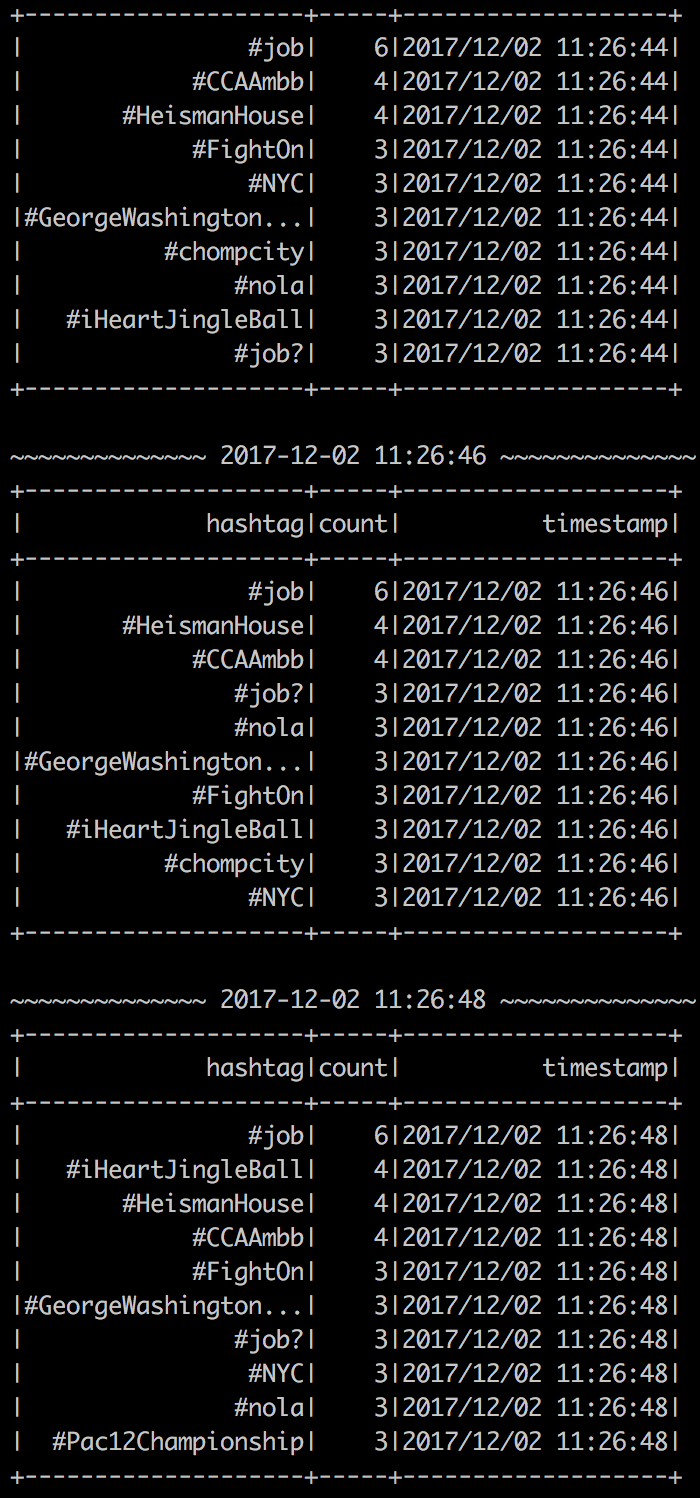
**hashtag\_counts\_df = sql\_context.sql("select hashtag, hashtag\_count from hashtags order by hashtag\_count desc limit 10")**

**hashtag\_counts\_df.show()**

**# call this method to prepare top 10 hashtags Data Frame and send them to display the output**

**send\_df\_to\_dashboard(hashtag\_counts\_df)**

***Aggregate Count logic***: The *updateStateByKey* expects a function with a parameter called the *update* function which runs on each item in RDD and does the required logic. In our code, we have written an update function called *aggr\_tags\_count* that will sum all the *new\_values* for each hashtag and add up all of them to the *total\_sum* which is the sum across all the batches and save this data into *tags\_count* RDD. Next, we will do processing on RDD, tags\_count in each batch in order to convert it to temp table using Spark Context and then perform a select statement in order to take out top ten Hashtags with their counts and put them into data frame named with *hashtag\_df* .



and so on...

Fig 8: Hashtag Count-Real Time Output

## **Final Output to Kibana Dashboard:**

## The last lap for us is that our Spark application sends the dataframe hashtag\_count\_df to the dashboard application which in our case it will be a live dashboard created using Kiaban Dashboard with use if Elastic Search which is fetching data from Spark Streaming. The data will be framed in two arrays, where one is for Hashtags and other is for their counts.

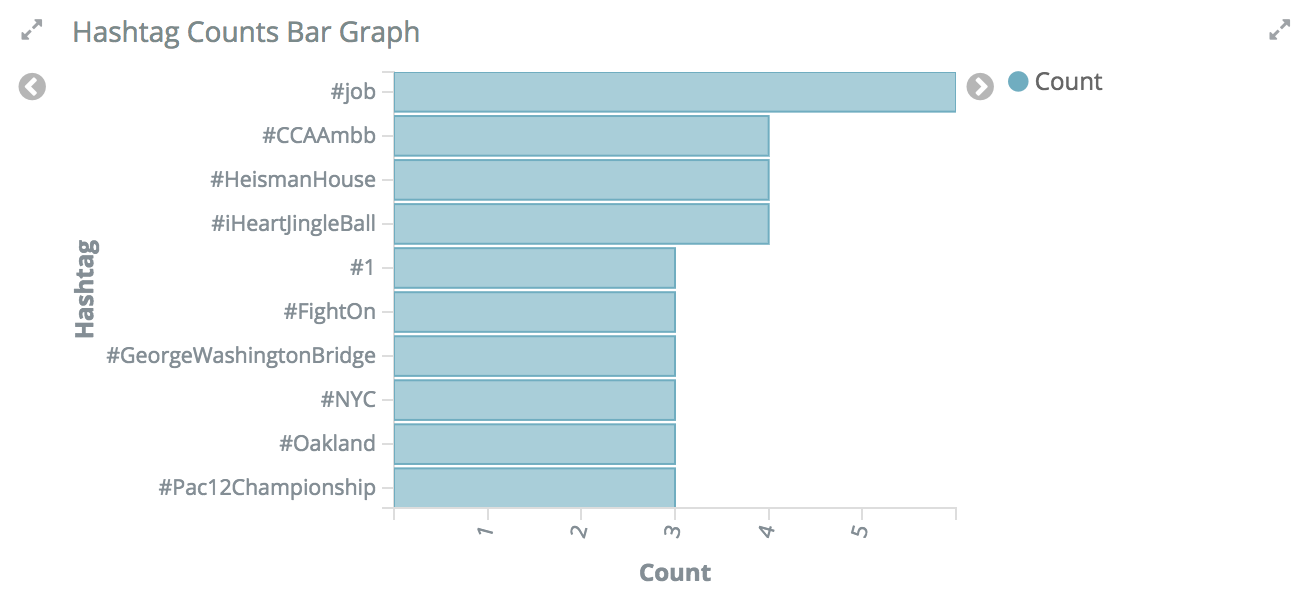


Fig 9: Display on Kibana Dashboard

# Conclusion

Online social networks provide communication channels to spread an idea, behavior, style or usage throughout the world village. Twitter is a special online service that provides both social network and microblog functions. Posting tweets through devices from desktop to mobile is the main activity of the microblog function, while following and retweeting offer the social network function. Users post tweets by encoding topics in the form of hashtags, which are summarized by Twitter to make a list of current trending tags. Even though hashtags are just a single word made by the tweet authors, many features can be extracted from a hashtag in Twitter. For example, tweets provide a context for the hashtag. Are there any hyperlinks or mentions in the tweet? Social influence resulting from the follow network in Twitter may impact the spread coverage of a hashtag as well. How many users have adopted the early tweets? What are the social influences of these early adopters? The influences can be judged from the number of followers, the number of statuses or the number of lists to which early adopters have been added. The time series associated with the early tweets adoption offers another feature for the prediction task. Differential series derived from the adoption series reveals whether the adoption of a hashtag is speeding up or slowing down. With using the Twitter Streaming API, we will be collecting tweets in real-time containing multiple hashtags. This paper givers a brief about all the transitioning technology in the field of big data. In this paper, we have implemented simple data analytics of data in real-time using Twitter API and Spark Streaming and integrating it directly with a simple data-frame using a RESTful web service. In the displayed example, we are able to see how Spark is powerful than other streaming technologies as it captures a massive stream of data, transforms it, and can be used to make decisions in no time.

# Future Scope

In future work, we will analyze other potentially useful features, and more importantly, propose more effective models for counting trending hashtags. So far, we have only looked at the hashtags contents of the tweet and there is much more data contained within that can be analyzed. Two other components that can be taken into consideration are the user mentions and the URLs in the message.

##### References

1. Zaharia, Matei & Chowdhury, Mosharaf & J. Franklin, Michael & Shenker, Scott & Stoica, Ion. (2010). Spark: Cluster Computing with Working Sets. Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 10. 10-10.
2. B. Yadranjiaghdam, S. Yasrobi and N. Tabrizi, "Developing a Real-Time Data Analytics Framework for Twitter Streaming Data," *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, 2017, pp. 329-336
3. Elgendy, Nada & Elragal, Ahmed. (2014), “Big Data Analytics: A Literature Review Paper”. Lecture Notes in Computer Science. 8557. 214-227. 10.1007/978-3-319-08976-8\_16.
4. M.Jayashree, S.Zahoor Ul Huq, “Beyond Batch Process: A BigData processing Platform based on Memory Computing and Streaming Data”.
5. Shahrivari, Saeed. (2014). Beyond Batch Processing: Towards Real-Time and Streaming Big Data. Computers. 3. . 10.3390/computers3040117.
6. LEKHA R. NAIR, D. S. (2015, october 20). STREAMING TWITTER DATA ANALYSIS USING SPARK. p. 350
7. S. Gaglio, G. Lo Re, M. Morana “**A framework for real-time Twitter data analysis”**
8. Makice, K. (2009, March). Twitter API: Up and Running. pp. 46,64,134.
9. Dojin  Choi1, Bosung  Kim2,  InsuBae2andSeokil  Song1, "In-memory Distributed Processing Method for Traffic Big Data to Analyze and Share Traffic Events in Real Time among Social Groups", International Journal of Software Engineering and Its ApplicationsV ol. 10, No. 1 (2016), pp. 51-58
10. Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, Ion Stoica,    “Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters”
11. J. Bai, "Feasibility analysis of big log data real time search based on Hbase and ElasticSearch," *2013 Ninth International Conference on Natural Computation (ICNC)*, Shenyang, 2013, pp. 1166-1170.