**CSP 554: Big Data Technologies**

**TWITTER SENTIMENT ANALYSIS FINAL REPORT**

**Group Members:**

**1: Deepika Padmanabhan- A20456289**

**2: Lei Yu- A20340548**

**3: Arjuna Anilkumar - A20446963**

**4: Jason Yeoh - A20457826**

**Introduction:**

In the past few years, there has been a growth in the use of microblogging platforms like twitter which has spurred the growth of companies and media organizations that are seeking ways to mine twitter for information about what people think about their products and services.

Information, in this big data era, has become vast and there is a need to collect and process these huge volumes of data as quickly as possible or in real time to make valuable business decisions on the same day the data was generated as waiting a couple of days would make the data obsolete.

In this project, we will be using streaming services like spark and kinesis to collect twitter data in real time, process the data, apply classification algorithms and conduct sentiment analysis on the collected data.

**What is sentiment analysis?**

Sentiment analysis is a practice of machine learning to classify the polarity of any word, sentence or document. Results are usually classified in “positive” , “negative” and “neutral”. We can turn tweets in text into feature vectors and feed them to classifiers like Naive Bayes to train the model for classification.

**Technologies used**

* **Apache spark**
  + Apache Spark is an open source, Hadoop-compatible, fast and general purpose cluster-computing platform created at AMPLabs in UC Berkeley and includes SPARK SQL, SPARK streaming, MLib and GraphX. Spark is designed to cover a wide range of workloads including batch applications, iterative algorithms, queries and streaming. Apache spark is known for its speed in processing large datasets as it runs computations in memory.
* **Spark streaming**
  + Spark streaming is a spark component that enables processing of live streams of data in the form of micro batches of logs, messages or events. Spark streaming provides an API to manipulate data streams that closely match the Spark Core’s RDD API.
* **Hadoop** 
  + Hadoop is an open source software framework that enables reliable, scalable storage and processing of large datasets in a distributed environment. Hadoop is designed to scale up from single servers to thousands of machines, each offering local computation and storage rather than rely on hardware to deliver high-availability. The library itself is designed to detect and handle failures at the application layer delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

* **AWS Kinesis**
  + Amazon Kinesis Data Firehose is the easiest way to reliably load streaming data into data lakes, data stores, and analytics services. It is a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration. It can also batch, compress, transform, and encrypt your data streams before loading, minimizing the amount of storage used and increasing security.
* **AWS Lambda**
  + AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration
* **AWS Cloudwatch Logs and Logs Insights**
  + Amazon CloudWatch Logs to monitor, store, and access the log files from Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS CloudTrail, Route 53, and other sources. In this project, lambda function generated logs.CloudWatch Logs Insights to interactively search and analyze the log data.

* **AWS Cloudwatch metric API**
  + Metrics produced by AWS services are standard resolution by default. Here, we published custom metrics(sentiment values). Hence, we published with high resolution. We published the metrics using the boto3 API.

**Implementation Details:**

**Architecture 1 : Spark and kafka**

**Workflow:**

Twitter -> Kafka -> Spark on AWS EMR-> Cloud watch

## **Kafka:**

The first step was to create a twitter account and get the twitter API credentials to get access to the twitter stream. In the python script, pykafka package was used to connect kafka to twitter and get the stream of tweets. Only the message of the tweet was extracted from the json format and sentiment analysis was performed on it using the AFINN module in python. The AFINN lexicon is one of the simplest and most popular lexicons that can be used extensively for sentiment analysis. The current version of the lexicon is AFINN-en-165. txt and it contains over 3,300+ words with a polarity score associated with each word. AFINN detects the sentiment and gives a sentiment value. This along with the message was produced to the kafka topic ‘twitter’ which was then consumed by spark for further processing.

## **Kafka - spark integration:**

Originally the kafka stream was meant to be processed as spark DStreams instead of spark dataframe but since spark streaming with DStreams has depreciated and the packages required for this to work did not work, spark structured streaming was used instead. The right version of kafka had to be installed for this integration to work. Since amazon cluster EMR had a spark version of 2.4.6 which has a scala version of 2.11, kafka must have the same version of scala to avoid version conflicts so a kafka version of 2.11-2.4.1 was chosen.

**Spark pipeline:**

With streams from Kafka pushing strings of tweet content with a sentiment score, the pyspark script(sparkSS.py) will receive the stream and turn it into a streaming data source of spark data frame. There are two transformations and three queries before visualization. First, we turn the sentiment score into a sentiment result:`` POSITIVE'', `` NEGATIVE” and” NEUTRAL”. Then we do a word count on the sentiment resulting a three-row data frame recording each sentiment result and their count. Finally, we query the data frame to extract three count value of visualization.

Our original plan was to do the sentiment analysis with a pre-trained model in AWS comprehend, but we are having trouble implementing the method on the streaming data frame. Moreover, we observed that each batch of tweets only contains a few hundred records which is a smell for the pre-trained model that is instant to analyze one string. However, spark data frame applies transformation in parallel which do not have much efficient gain for repetitive small jobs. Since the analysis does not benefits form parallel, we decided that we do not have to do the analysis on the spark

**Cloud watch:**

Like what we did with Kinesis, we will be using Cloud watch for visualization, but will push metrics for plotting. The Numeric value we extracted with the query in sparkSS.py is the total count since the program started. To visualize the change of Sentiment trend we use the metric function RATE(): Returns the rate of change of the metric per second so that the graph is not three constantly increasing lines.

The value we showed in the graph is extracted from a streaming data frame. Different form transformation, we cannot apply action like query to a streaming data frame like a fixed data frame. What we have to do is to wrap the query and update the cloud watch metric in a user defined function. Then call this function in .foreachBatch() after .writeStream. This difference illustrate the general idea of processing a data frame with streaming source: Source -> transformation -> Sink, and to apply some function on the data frame, we need to set Sink to consume incoming batch of data frame with foreach Batch() to apply the function interactively to the batched in stream.

**Setup:**

1. Upload installBoto3.sh to Amazon S3 containing command:

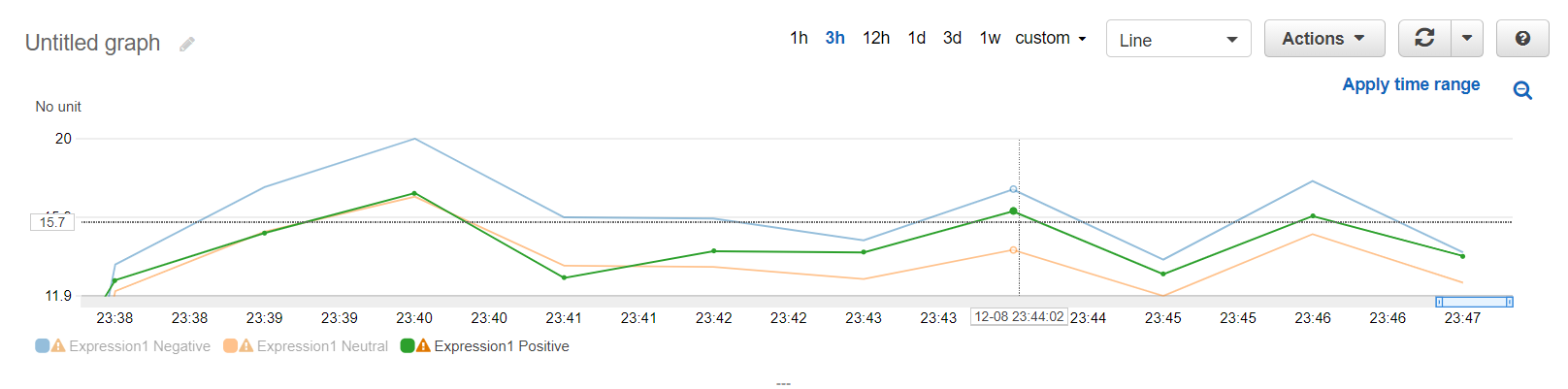
#!/bin/bash

sudo python3 -m pip install boto3

1. Create EMR cluster with spark, and run installBoto3.sh in bootstrapping step to install package on all node in cluster
2. Start Kafka stream with twitter-kafka.py on Kafka machine
3. Run starks.py on spark cluster
4. Go to Cloud watch page for metric on AWS

Result:

On the cloud watch page we can see an interactive graphing section with the metric which represents total count for each sentiment since the program started running. One can select the type of plot or apply some function to the metrics to produce the desired graph.



Future improvement can be applied to show timestamps at each step to get an evaluation of the delay in each step. We can also optimize sparkSS.py so that it won’t keep aggregating tweets in a data frame which will eventually flood the ram.

**Architecture 2: AWS Kinesis and Lambda**

Workflow:

**Producer(Kinesis client) ---> AWS kinesis---> lambda→ Cloudwatch logs→ Logs Insight**

Here the producer is the python script which invokes kinesis boto3 API to send tweets as records to AWS Kinesis streams.

**AWS Kinesis stream:**

The following is the snippet of kinesis stream description

{

"DeliveryStreamDescription": {

"DeliveryStreamName": "Twitter-stream",

"DeliveryStreamARN": "arn:aws:firehose:us-east-1:756254499329:deliverystream/Twitter-stream",

"DeliveryStreamStatus": "ACTIVE",

"DeliveryStreamEncryptionConfiguration": {

"Status": "DISABLED"

},

"DeliveryStreamType": "DirectPut",

"VersionId": "1",

"CreateTimestamp": "2020-12-09T12:03:57.465000-08:00",

"Destinations": [

{

"DestinationId": "destinationId-000000000001",

"S3DestinationDescription": {

"RoleARN": "arn:aws:iam::756254499329:role/service-role/KinesisFirehoseServiceRole-Twitter-strea-us-east-1-1607544090647",

"BucketARN": "arn:aws:s3:::tweetsanalysis-bigdata",

"Prefix": "",

"ErrorOutputPrefix": "",

"BufferingHints": {

"SizeInMBs": 1,

"IntervalInSeconds": 60

},

"CompressionFormat": "UNCOMPRESSED",

"EncryptionConfiguration": {

"NoEncryptionConfig": "NoEncryption"

},

"CloudWatchLoggingOptions": {

"Enabled": true,

"LogGroupName": "/aws/kinesisfirehose/Twitter-stream",

"LogStreamName": "S3Delivery"

}

},

"ExtendedS3DestinationDescription": {

"BucketARN": "arn:aws:s3:::tweetsanalysis-bigdata",

"Prefix": "",

"ErrorOutputPrefix": "",

"BufferingHints": {

"SizeInMBs": 1,

"IntervalInSeconds": 60

},

"CompressionFormat": "UNCOMPRESSED",

"EncryptionConfiguration": {

"NoEncryptionConfig": "NoEncryption"

},

"CloudWatchLoggingOptions": {

"Enabled": true,

"LogGroupName": "/aws/kinesisfirehose/Twitter-stream",

"LogStreamName": "S3Delivery"

},

"ProcessingConfiguration": {

"Enabled": true,

"Processors": [

{

"Type": "Lambda",

"Parameters": [

{

"ParameterName": "LambdaArn",

},

{

"ParameterName": "NumberOfRetries",

"ParameterValue": "3"

},

{

"ParameterName": "RoleArn",

},

{

"ParameterName": "BufferSizeInMBs",

"ParameterValue": "1"

},

{

"ParameterName": "BufferIntervalInSeconds",

"ParameterValue": "60"

}

]

}

]

}

We have enabled Kinesis Data Firehose data transformation, Kinesis Data Firehose buffers incoming data up to 3 MB by default. The buffering configuration in our project is 1 MB.

Kinesis Data Firehose then invokes the specified Lambda function asynchronously with each buffered batch using the AWS Lambda synchronous invocation mode.It then delivers the records to the lambda function. The Lambda function does the data transformation. Here, sentiment analysis of the tweets is performed.

**AWS Lambda:**

We created a python function in AWS Lambda to retrieve records from kinesis and analyze the sentiment of each record.

To analyze the sentiment of tweets , we used Amazon Comprehend service. Amazon Comprehend is a natural language processing (NLP) service that uses machine learning to find insights and relationships in text. A sample boto3 API to detect the sentiment of the text is as follows:

**comprehend = boto3.client(service\_name='comprehend', region\_name=<region-name>')**

**s = comprehend.detect\_sentiment(Text=<tweet>, LanguageCode='en')**

Using the above mentioned API, the sentiment of tweets will be detected and sent back to kinesis firehose.

Lambda functions use an [execution role](https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html) to get permission to write logs to Amazon CloudWatch Logs. In the lambda function, we print the message and its sentiment. Hence, they are logged in Cloudwatch Logs.

**AWS Cloudwatch Logs and Logs Insights:**

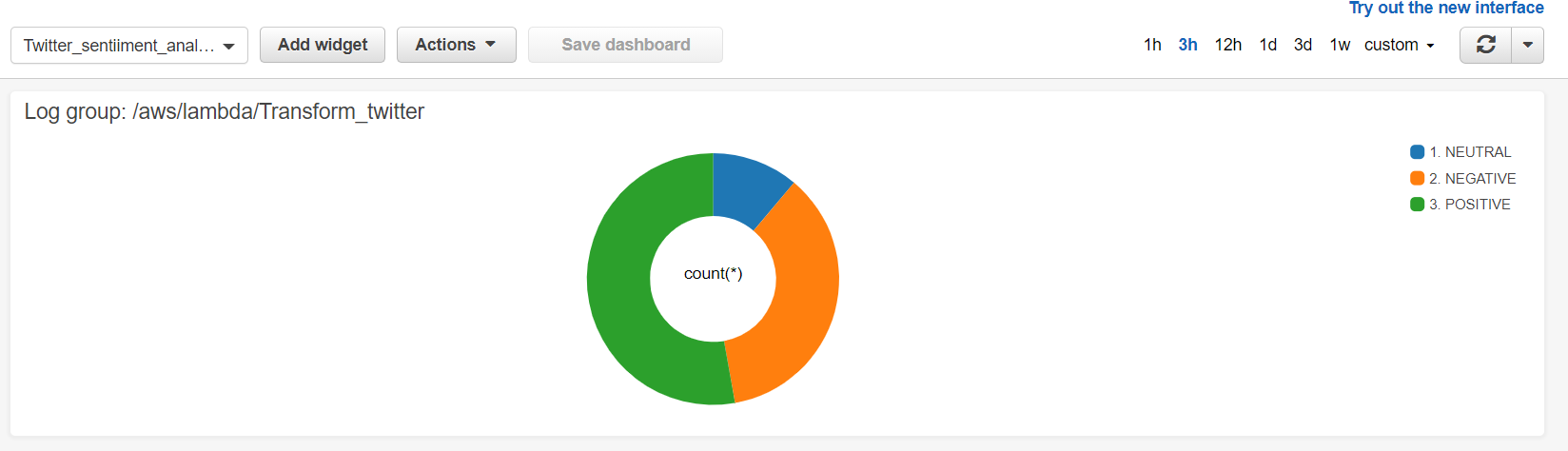
After every invocation of Lambda function, the console output is written to Cloudwatch Logs. Sample output is as follows:

CloudWatch Logs Insights enables us to interactively search and analyze your log data in Amazon CloudWatch Logs. We used the following query to view the count of positive and negative sentiments.

**fields *@timestamp*, *@message***

**| stats count(\*) by sentiment**

The following is the pie chart obtained using Logs Insights.

****

The Cloudwatch dashboard keeps getting updated as long as the producer streams tweets through Kinesis.

**Comparison:**

**Kinesis Firehose:**

* Firstly, AWS SDK using the asynchronous client like the put Record batch function is one of the prominent features which allows you to put a list of records in one put record request, which saves time and it's more efficient.
* Linking lambda functions to the stream to transform the data as it arrives before writing it in S3 is great to perform some aggregations or enrich the data with other data sources.
* Easier to implement and manage.
* Ability to have one single flow of data from multiple consumers simplified our architecture a lot ,one should just pull that data and then use multiple consumers.

**Apache Spark:**

* In the case of Apache Spark, one needs to optimize the code manually since it doesn’t have any automatic code optimization process
* Apache Spark doesn’t come with its own file management system. It depends on some other platforms like Hadoop or other cloud-based platforms.
* Implementation and management is not as easy as kinesis.

**References:**

1. [J. Gratch, G. Lucas, N. Malandrakis, E. Szablowski, E. Fessler and J. Nichols, "GOAALLL!: Using sentiment in the World Cup to explore theories of emotion," 2015 International Conference on Affective Computing and Intelligent Interaction (ACII), Xi'an, 2015, pp. 898-903, doi: 10.1109/ACII.2015.7344681.](https://ict.usc.edu/pubs/GOAALLL!%20Using%20Sentiment%20in%20the%20World%20Cup%20to%20Explore%20Theories%20of%20Emotion.pdf)
2. [Shirdastian, Hamid & Laroche, Michel & Richard, Marie-Odile. (2017). Using big data analytics to study brand authenticity sentiments: The case of Starbucks on Twitter. International Journal of Information Management. 48. 10.1016/j.ijinfomgt.2017.09.007.](https://www.researchgate.net/publication/320706936_Using_big_data_analytics_to_study_brand_authenticity_sentiments_The_case_of_Starbucks_on_Twitter)
3. [Hao Wang, Dogan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. 2012. A system for real-time Twitter sentiment analysis of 2012 U.S. presidential election cycle. In Proceedings of the ACL 2012 System Demonstrations (ACL '12). Association for Computational Linguistics, USA, 115–120.](https://www.aclweb.org/anthology/P12-3020.pdf)
4. [J. Ramteke, S. Shah, D. Godhia, and A. Shaikh, “Election result prediction using Twitter sentiment analysis,” in Inventive Computation Technologies (ICICT), International Conference on, 2016, vol. 1, pp. 1–5.](https://www.irjet.net/archives/V7/i2/IRJET-V7I2606.pdf)
5. [Khalid N. Alhayyan, D. Imran Ahmed “Discovering and Analyzing Important Real-Time Trends in Noisy Twitter Streams”](http://www.iiisci.org/journal/CV$/sci/pdfs/HB800YX17.pdf)
6. [M. Frampton, Mastering Apache Spark: gain expertise in processing and storing data by using advanced techniques with Apache Spark. Birmingham: Packt Publishing, 2015.](http://2.droppdf.com/files/lc0aC/mastering-apache-spark.pdf)
7. <https://github.com/kaantas/spark-twitter-sentiment-analysis>
8. <https://github.com/tthustla/setiment_analysis_pyspark/blob/master/Sentiment%20Analysis%20with%20PySpark.ipynb>
9. <https://docs.aws.amazon.com/lambda/latest/dg/with-kinesis.html>
10. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_AnalyzeLogData_VisualizationFieldQuery.html>