Project Report

Real Time - YouTube Video Sentiment Analysis

Big Data & Cloud Deployment
Akarsh Gupta
Sharan Girdhani

- **Abstract**

Big data trend has enforced the data-centric systems to have continuous fast data streams. In recent years, real-time analytics on stream data has formed into a new research field, which aims to answer queries about "what-is-happening-now" with a negligible delay. The challenge with real-time stream data processing is that it is impossible to store instances of data, and therefore online analytical algorithms are utilized. This project lays the groundwork for designing & implementing a real-time sentiment analysis of YouTube videos based on its comments. The following discussion and the implementation after that shows how any YouTube video can be rated based on the comments of its previous viewers. Using already existing concept of sentiment analysis i.e. Stanford-Corenlp, we visualize the sentiments of the users for top videos based on different search keywords**.**

- **Introduction**

In big data, sentiment analysis is one of the most popular methods for analyzing the likes and dislikes of users for any entity which is available for commenting. It would be even better, if an analysis like this could be made real time. In this project, we have tried to achieve a similar goal by doing a real-time sentiment analysis on any topics that we search on YouTube. This project is mainly divided into 3 parts i.e. Data Ingestion from YouTube APIs, Data Processing, and Web Application for Data Visualization. Here is a small example of how the application works:

At the start, we ask the user to pick a topic on which he/she need to do the sentiment analysis. Just for the sake of this example, let's say that topic is "soccer". We fetch comments of top videos on this search keyword using YouTube's 'Search' and 'Comment' APIs and organize this data in a desired format. To make a data pipeline for streaming data, we send this data to an Apache Kafka producer, written in Java. Using Spark Streaming in Scala, we process this streaming data in small batches i.e. take the sentiment score of all the comments using the Stanford Corenlp library and group them based on each video. This resultant data is then stored on a local Redis Server to be picked up by the event stream created in Python (Flask). Finally, we do a data visualization on this data using D3.js library in JavaScript.

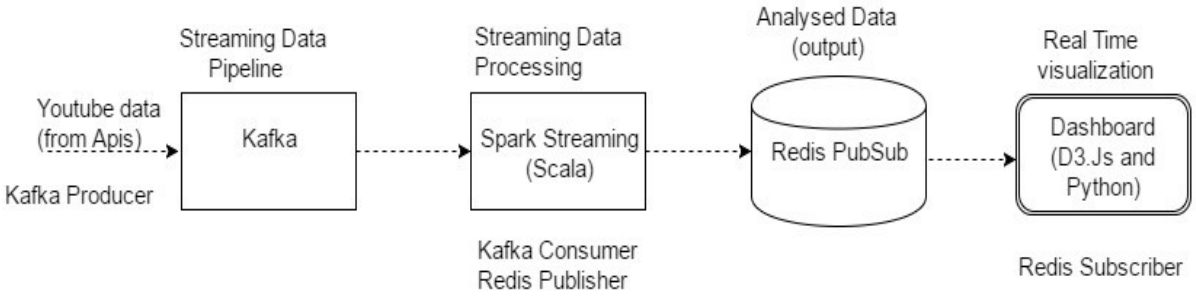
The detailed analysis and usage of each technology or languages is described further in the Technical Description and experiment section of this report.

- **Related works:**

Real time sentiment analysis is a very popular concept in Big Data. There has been some similar works on real time data provided by Twitter Streaming APIs. Talking about the libraries, we also looked at other libraries like Tinga, NLTK (Python) etc. But given the multi-language and it's availability in Scala, we decided to go ahead with Stanford Corenlp. Other technologies like Apache Kafka, Redis, Python Flask & D3.js were completely based on our research based on the requirements and architecture of the project.

- **Technical Description and experiment**

Infrastructure and System architecture is an important factor for a Big Data or Cloud Applications, as our project is a real-time solution that we designed an architecture which is reliable and scalable and fast enough to process in real-time. Our architecture has Apache Kafka, Spark Streaming, Redis PubSub with languages like java, Scala, python and D3.js. Below is a figure of our architecture.



We started from retrieving data from YouTube, Google provides YouTube data API's which gives a plethora of data related to videos, users, publisher, channels, comments, region etc. We thought of using comments data and analyses sentiments of comments using text analysis in real time. We created a console where you can provide a keyword, YouTube search api searches for that keyword and gives top videos results for that keyword and then we start retrieving all the comments for each video and produce the comments data to Kafka to create streaming data pipeline. Kafka consumer is written inside Spark Streaming, where Spark Streaming creates a stream by consuming Kafka stream and then Spark Streaming processes the comments data and calculates its sentiment (0 being most negative and 4 being most positive). For calculating the sentiment, we have used Stanford Corenlp library which internally creates a TF IDF vector of all words and matches its positivity or negativity from its dictionary.

Let's discuss the use of every technologies used in this project:

<u>YouTube Data API's:</u>

We are using two of YouTube data API's. One is for Search and second is for Comments.

The Search API takes a keyword as parameter and give video's as results. The Comment API, takes video id's and gives list of comments as response. Below are the URLs of YouTube data API's used in the project.

Search API: https://www.googleapis.com/youtube/v3/search

Comments API: https://www.googleapis.com/youtube/v3/commentThreads

We have used Java to fetch data continuously, this java app is starts and asks for a keyword once the keyword is given it uses the search api and starts pulling all videos and then it calls the comments api, once comments data starts coming the app starts publishing the data to Kafka topic "YouTube". The app runs till it retrieves all the data for the searched videos.

APACHE KAFKA:

Apache Kafka is distributed a publish-subscribe messaging system. Kafka has several features that make it a good fit for our requirements: scalability, data partitioning, low latency, and the ability to handle large number of diverse consumers. We have configured Kafka with a single topic named YouTube. As Kafka is pub-sub messaging system, it has two parts which are producer and consumer. Producer is the way of entry of messages in the system and consumer is way of accessing those messages from the system. Messages are published to topics, there can be multiple topics as well as multiple producer and multiple consumers or groups which has users/consumers.

Kafka in our application act as data pipeline which provides streaming data, Kafka producer is written in java which hits the YouTube Data API and starts publishing data in Kafka as producer as text messages delimited by "|". This text message consists of video id, video title, video description and comment text.

In our application, we are using Kafka as standalone which can be scaled to clusters when required, the main reason to use Kafka is because it is distributed system so scaling of application can be done easily and it has very high throughput and keeps messages in buffer so no streaming data is lost.

SPARK STREAMING:

Spark Streaming runs on top of Spark, it operates by creating micro batches from the streaming. This creation of micro batches creates batches provided by the configuration and takes all the consumes all the streaming data within that time and creates micro batches which are internally RDDs. Spark Streaming supports real time processing of streaming data, such as production web server log files (e.g. Apache Flume and HDFS/S3), social media like Twitter, and various message queueing systems like Kafka, Flume, ActiveMQ, RabbitMQ.

In our application Spark Streaming is used to consume streams from Kafka. We have written the Spark Streaming code in Scala. Firstly, we are consuming the stream from Kafka in Scala and creates a direct stream using Spark Streaming, As we discussed earlier about the format of our messages coming from Kafka, here we split the messages and convert that it into

data frames. We are using Stanford Corenlp library to analyses the sentiments of comment text from the messages received from the stream. After the sentimental analysis, we are querying into this data-frame using Spark SQL and taking an average sentiment score of all the comments of a particular batch by their video id's and publish it on our Redis server.

REDIS SERVER:

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyper-log logs and geospatial indexes with radius queries.

In our Project, we are using Redis Pub-Sub feature i.e. using it as a message broker which can be used to create event streams for our web server, so our web server can push data to clients without client's request. The use of this is to create a real-time visualization. Here we have created a channel named "YouTube" where analyzed data from Spark Streaming is published to this channel which will be consumed by our web server and further pushed to the client application. One of the most important reason of choosing Redis is that it can be used as a distributed system, where we can create cluster (multiple instance) which is highly scalable and fast because of in-memory database.

PYTHON as WEB SERVER-SIDE APP(FLASK):

Python is a high-level programming language which has a great library support for data analysis and web frameworks, we are using Python Flask a micro web-framework. The reason behind using python flask is because the development is very rapid and it is highly scalable. We have used Flask for its simplicity and its features. we only require a small web server which can consume the messages from Redis channel and push the data to client as soon as it consumes the data from Redis channel. Flask was the perfect option based on scalability, performance, and its rapid development. In our application, we have created a web server and an endpoint URL for **"text/event-stream"**. We have created **"/stream"** endpoint where our python server keeps pushing data as soon as it consumes from Redis server. The format of messages pushed to client is Video Id, Video Title, Video description, Average Sentiment Score for Comments separated by delimiter ",".

CLIENT APPLICATION (DASHBOARD):

We are using JavaScript, JQuery and D3.js for our dashboard. D3.js is an amazing visualization JavaScript library. We have used it to create bar charts for all the videos which we are analyzing. You can notice the changes here in real-time, i.e. as soon as the stream keeps coming and analyzed and sent to Redis our visualization keeps updating automatically. The performance was quite good and we noticed changes in visualization in every 1-2 seconds. Along with the bar chart we are also updating our UI as soon as a chunk of data comes in a table which displays the Video Id, Video Title, Video description and batch average comment

sentiment score. This batch is a streaming batch i.e. all the data analyzed in one batch of streaming (as we know spark streaming is processed in micro-batches).

SPARK TUNING:

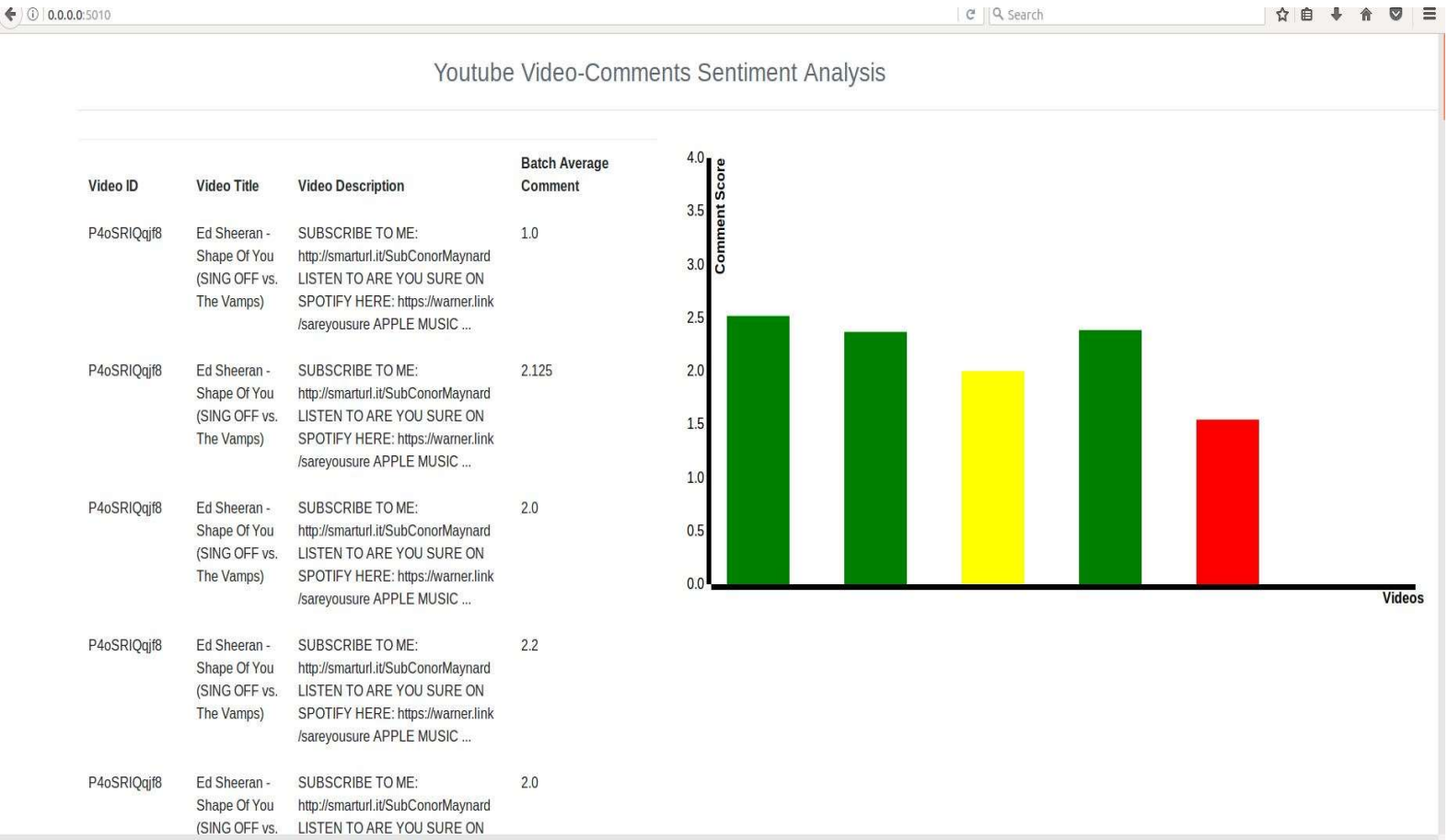
There are various parameters which can be tuned to increase the performance of Spark Streaming, one of them is by changing the allocated memory to master or the executors but unfortunately our environment has constraints, so we increased the memory by very little 2g and we are running Spark as a standalone mode so even increasing the memory did increase the performance as standalone mode has only one executors which is the driver node in our case. So there was not any performance tuning which we noticed.

- **Results and Conclusion**

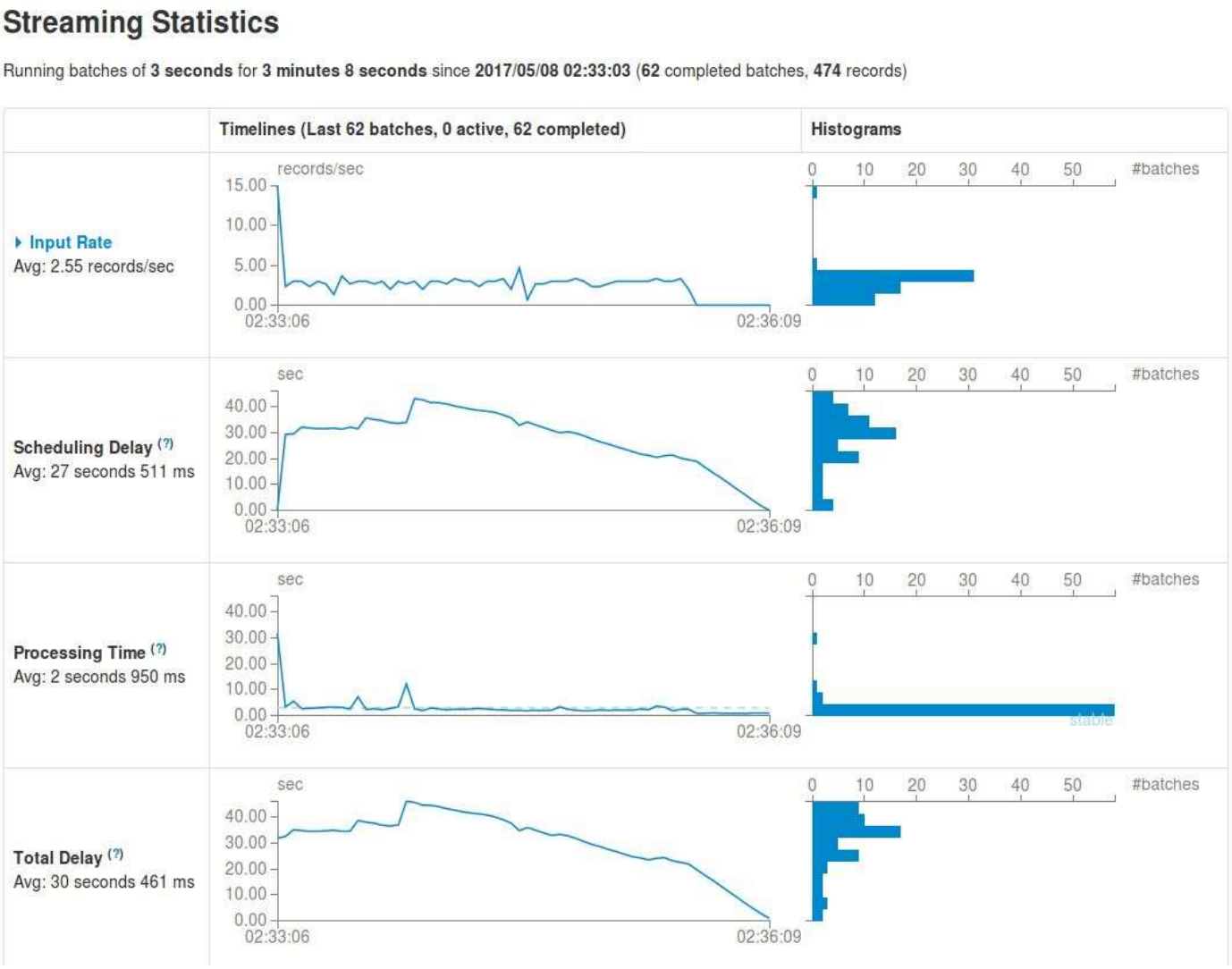
To conclude, we successfully analyzed the sentiment of YouTube videos by using text mining and sentiment analysis on comments data of videos. Though the application was experimented on a very low infrastructure, but the architecture which we designed is scalable and can be extended to run on clusters. All the technologies we have used to process data can be run on distributed environments(clusters) with multiple instances to increase the performance of application when there is increase in data flow. Below we have attached a

screenshot of our application result and spark streaming jobs executed for the same.

This is a screenshot of the final bar chart visualization that we got after the sentimental analysis of the top 5 videos, each bar is representing the overall average sentiment of all the



comments of a video. The table in the left, shows the video data and the average comment of each batch processed in spark streaming.



Color Coding for the above graph: Red: Negative, Yellow: Neutral, Green: Positive

The above screenshot provides the statistics like Input Rate, Scheduling Delay, Processing Time, and Total Delay for each batch run in spark streaming in the form of Visualizations.

- **Citations & References**
    - http://henning.kropponline.de/2016/12/25/simple-spark-streaming-kafka-example-in-a-zeppelin-notebook/
    - http://aseigneurin.github.io/2016/03/02/kafka-spark-avro-kafka-101.html
    - http://static1.squarespace.com/static/55007c24e4b001deff386756/t/55f590a9e4b044a1a342598f/1442156713857/B131+-+Patel,+Nixon.pdf
    - https://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/561/Real-time%20stream%20processing%20for%20Big%20Data.pdf
    - https://redislabs.com/docs/real-time-analytics-redis/
    - http://www.cs.utah.edu/wondp/guz.pdf