Data-Intensive Computing - Project Report

Sri & Nik - 31st Oct 2019

Problem description

As outlined in our project proposal, we have developed a scalable pipeline for live youtube-trend visualization. The knowledge of trends like the *hottest topics* or the *most likes* on youtube can be valuable to stakeholders. Additionally, we can see which content creators are able to reach out to the most amount of people in a single video category, such as educational content or music. This can prove useful to marketing teams or to the youtube content creators themselves. It's also possible to display the difference between regions.

Approach

The main idea behind the project implementation is to experiment and get hands-on experience with different data-intensive computing tools. We also wanted to get experience with implementing a scalable pipeline that can stream the data from YouTube and present the user with some beautiful metrics. To accomplish these goals we have implemented our system using Kafka and PySpark.

The implemented system comprises 4 major components:

1. The YouTube API handler class in *youtube/bridge.py*. It is responsible for querying the **YouTube Data API** queries. A query result contains the following information:
    a. root JSON object: *kind*, *etag*, *id*, *snippet*, *statistics*
    b. snippet JSON object: *publishedAt, channelId, title, description, thumbnails, channelTitle, categoryId, liveBroadcastContent, localized*
    c. statistics JSON object: *views, likes, dislikes, favorites, comments*

2. The second is the **Kafka** broker located in *youtube/stream.py,* which uses the YouTube API handler class to get the data and acts as a producer by making the query results available in a *youtube_stream* topic queue.

3. The third is **Spark**, located in *dashboard/analytics.py,* which consumes the messages in the above-mentioned topic. In the Spark (PySpark) context, data is parsed into the desired form and the required information for the current needs are stored as a table in a **spark-warehouse**.

4. Finally, the fourth component is a neat **Jupyter** notebook located at *dashboard/dash.ipynb*. The notebook has a spark session that is connected to the earlier spark context (of the Kafka consumer) and reads the data saved in the spark-warehouse table. The content of the table is then sorted with respect to the number of likes of the video and a dashboard with information is dynamically plotted using **Matplotlib.**

Dataset

Though there is a huge dataset publicly available in Kaggle (dataset page), it is just static data and streaming that data at rest did not seem to be the right choice since we want something “live”. Hence the data used is the real-time response from the API (link to API doc and console).

Instructions to run the system

To run the system a kafka server needs to be set up. The script *server_setup/run_servers.bash* takes care of initialising the environment variables, starting **Zookeper** and **Kafka** as well as initialising the *youtube_stream* kafka topic.

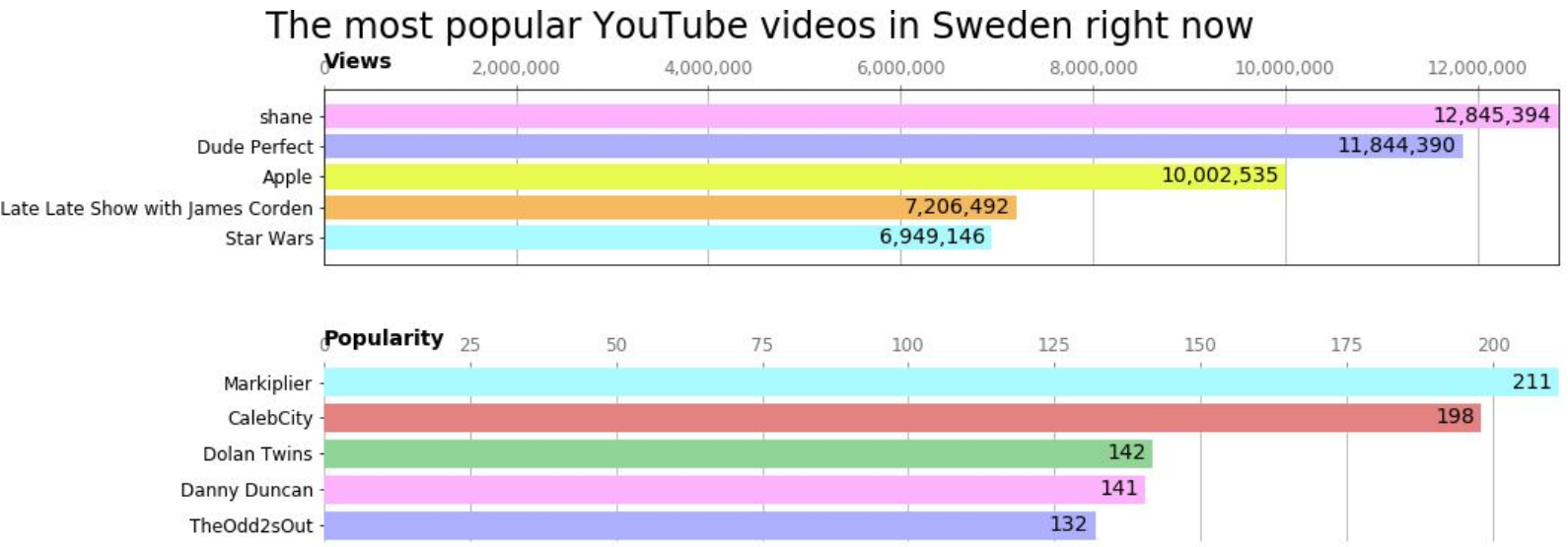
To set up **Spark**, the script *server/launch_streaming.bash* sets up spark by initialising environment variables and submitting the launch of *dashboard/analystics.py*. It, therefore, starts the consuming process.

Running *stream.py* starts the process of querying youtube and producing to the *youtube_stream* topic.

In a jupyter-notebook session, *dashboard/dash.ipynb* can be run to dynamically see the data change.

Results

With everything in place as mentioned above, the system is now capable to stream YouTube data through Kafka. It can retrieve a high volume of data by passing by changing the parameters of the script. Due to constraints of the free-to-use YouTube API, we can only make 10k queries per day, which limits the frequency at which we can update the dashboard. Because we use Spark to process the youtube queries, it can be parallelized by adding more nodes to the spark session. The results are saved in a warehouse. Reading the table frequently can be used to provide nice visualizations as in the image below.



Achieved goals

In the project proposal we had set out the following goals:

1. Fetch data from YouTube API
2. Perform queries on the data using pyspark
3. Visualize the trends using plotly

Task 1 and 2 have been successfully implemented. Additional complexity was added by using a Kafka broker in the process. Instead of using plotly, we used matplotlib instead, which is not a big difference.