PES UNIVERSITY
100 feet Ring Road, BSK 3rd Stage
Bengaluru 560085 INDIA

Department of Computer Science and Engineering
B. Tech. CSE – 6th Semester
Jan – May 2024

UE21CS343BB3
DATABASE TECHNOLOGIES (DBT)

PROJECT REPORT
on

Batch and Stream processing of Youtube dataset

Sai Harshith Narra	PES1UG21CS513	6I	Shreya Prasad	PES1UG21CS576	6J
Shresht V G	PES1UG21CS566	6J	Sumukha Ganesha	PES1UG21CS641	6K

Class of Prof. Raghu B. A.

Table of Contents

1. Overview

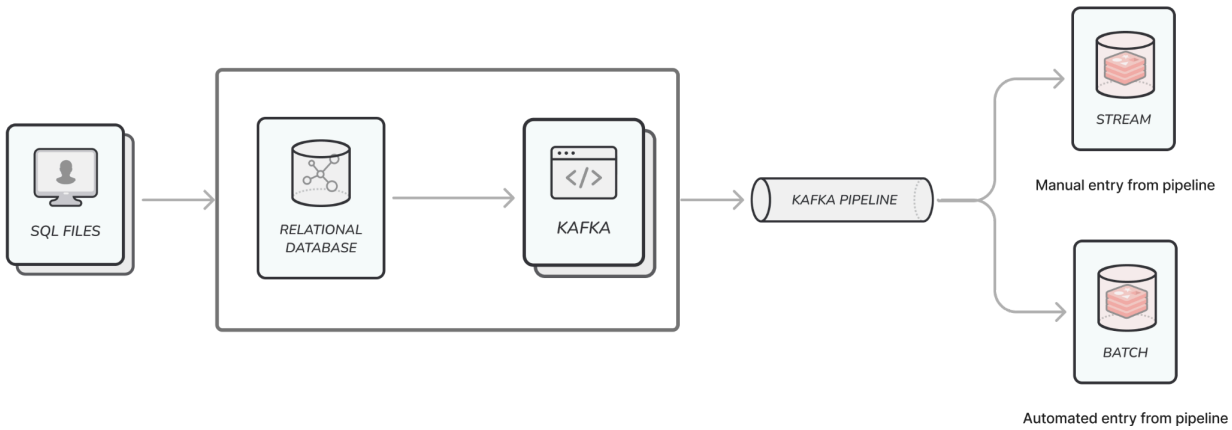
1.1 Introduction

We've undertaken a sophisticated project centered around the implementation of a Lambda architecture, leveraging Kafka for real-time query simulation and Spark for both stream and batch processing. In this architecture, Kafka serves as the publisher, facilitating the ingestion of simulated real-time data, while Spark seamlessly handles subscription and processing tasks. We are simulating the real-time data instead of using any real-time APIs. This is achieved by randomised streaming of data from the database.

1.2 Problem description

The project tackles the formidable task of processing vast volumes of YouTube data, necessitating intricate actions, transformations, and aggregations. Utilizing Kafka, we establish multiple topics — channels, video views, likes, comments and subscribers to continuously stream data from MySQL. Kafka ensures real-time streaming, mirroring Twitter API-like performance. Apache Spark complements this architecture by handling subscription and processing tasks, including batch processing. The project offers insights into efficiently analyzing real-time streaming data, particularly from YouTube, showcasing Spark and Kafka's prowess. Furthermore, it explores comparisons between streaming and batch processing for specific topics, incorporating windowing techniques for enhanced analysis.

1.3 Software architecture



2. Software Installation

2.1 Software Installation

1] Apache Kafka (3.2.3): A distributed streaming platform that allows for the building of real-time data pipelines and streaming applications. We used this for streaming of data and as a publisher.
LINK : https://downloads.apache.org/kafka/3.2.3/kafka-3.2.3-src.tgz

2] Apache Spark Streaming (3.4.0): A real-time processing engine built on top of the Apache Spark platform, allowing for the processing of data streams in real time. Also, gives way for batch processing.
LINK: https://spark.apache.org/downloads.html

3] ZooKeeper (3.9.2): ZooKeeper is a centralized service used for maintaining configuration information, providing distributed synchronization, and offering group services
INSTALLATION: brew install zookeeper

2.2 Data Preprocessing Tools

We have used https://tableconvert.com/csv-to-sql#google_vignette to convert CSV files to SQL files.

2.3 Streaming Apps/Tools

1] Apache Kafka
2] Apache Spark Streaming

2.4 Repository/Database

We have used MySQL 8.0.31

3. Input Data

1.1 Source

YouTube videos and channels database collected from Kaggle in the form of a .csv file

1.2 Description

This dataset contains data of YouTube videos and their respective channel details. It contains channel ID, view count, subscriber count, duration of videos and other quantitative ratios such as views per video, to describe each video. However, only a subset of those parameters are taken into consideration for our project.

```
CREATE TABLE InteractionMetrics(
    channelId VARCHAR(10),
    likesPerSubscriber DOUBLE,
    videoId VARCHAR(10),
    dislikesPerSubscriber DOUBLE,
    viewsPerElapsedTime DOUBLE,

    PRIMARY KEY (channelId, videoId),
    FOREIGN KEY (channelId) REFERENCES Channels(channelId),
    FOREIGN KEY (videoId) REFERENCES Videos(videoId)
);

CREATE TABLE Videos(
    channelId VARCHAR(10),
    videoId VARCHAR(10),
    videoViewCount DOUBLE,
    videoPublished VARCHAR(10),

    PRIMARY KEY (videoId),
    FOREIGN KEY (channelId) REFERENCES Channels(channelId)
);

CREATE TABLE Channels(
    channelId VARCHAR(10),
    channelViewCount DOUBLE,
    videoCount DOUBLE,
    subscriberCount DOUBLE,
    channelelapsedtime DOUBLE,
    channelCommentCount DOUBLE,

    PRIMARY KEY (channelId)
);
```

4. Streaming Mode Experiment

4.1 Description

The code reads data from a Kafka topic, processes the data in a streaming fashion, calculates the count of tweets per hashtag within a 30-minute window, sorts the result by window start time and hashtag, and writes the output to both the console and a CSV file. Here Pyspark is used to subscribe to the topic provided by Kafka.

4.2 Windows

The code establishes a sliding window of 15 minutes, grouping data within it based on provided topics. This 15-minute window serves as a cutoff, ensuring that any query or data point delayed beyond this timeframe is discarded.

4.3 Results

Upon execution of the spark_stream.py file, users are prompted to select a topic. Subsequently, Spark initiates its lifecycle by establishing the input data source and creating partitions for the input RDD. It proceeds to validate parent stage dependencies and submits shuffle map stages. Temporary metadata files related to schema and state store are generated, and the state store is loaded and updated accordingly.
Following these initial steps, a log output detailing progress metrics for a streaming query executed by Apache Spark's Structured Streaming engine is displayed. This log tracks the progress of a streaming query identified by its runId. Notably, both numInputRows and inputRowsPerSecond register as 0, indicating no new data was ingested during this period. Similarly, processedRowsPerSecond is also 0, suggesting no rows were processed. Various timing metrics, including addBatch, commitOffsets, getBatch, latestOffset, queryPlanning, triggerExecution, and walCommit, are provided in milliseconds.

Additionally, the eventTime watermark defaults to the epoch value if no data with event times is present. The stateOperator named "stateStoreSave" is observed to track state data, yet metrics such as numRowsTotal, numRowsUpdated, and numRowsRemoved all indicate 0, signifying no state updates occurred during this period.

5. Batch Mode Experiment

5.1 Description

This program reads data from a Kafka topic in a streaming fashion using Spark Structured Streaming. The data is parsed using a predefined schema and processed to count the number of tweets based on their language within a sliding window of 5 minutes. The processed results are then output to the console using the complete mode of output.

5.2 Data Size

The size of the data is 20 rows per batch. It depends on the size of the data produced and stored in the Kafka topic.

5.3 Results

Upon execution of the spark_stream.py file, users are prompted to select a topic. Subsequently, Spark initiates its lifecycle by establishing the input data source and creating partitions for the input RDD. It performs the data retrieval of video ID and duration, in batches iteratively for a window duration of 5 minutes. Unlike Streaming data, batch data takes input of larger data files for a given duration.

6. Comparison of Streaming & Batch Modes

6.1 Results and Discussions

In our analysis, we employed both streaming and batch processing techniques, focusing solely on video IDs of all youtube channels. Our observations revealed significant disparities in processing statistics between the two approaches. Notably, the processed row count per second in streaming was minimal, hovering around zero, as the processing occurred nearly in real time. Conversely, batch processing yielded a few hundred processed rows, indicating a delay in data processing. This delay is inherent to batch processing, which necessitates data accumulation before analysis. Additionally, memory consumption and commit time were considerably lower in streaming compared to batch processing, underscoring the efficiency of the former in handling real-time data streams. These metrics provide valuable insights into the performance disparities between streaming and batch processing paradigms, facilitating a comprehensive evaluation of their respective strengths and limitations.

7. Conclusion

Streaming and batch processing are two fundamental paradigms in data handling, each with its own distinct advantages. Streaming shines when dealing with real-time or near real-time data, processing it as it comes in, while batch processing efficiently handles large volumes of data all at once.

In our showcased code snippet, we leveraged batch processing to analyze a substantial influx of tweets sourced from a Kafka topic within specific time windows. We organized this data based on the number of videos published by each channel and presented the results in the "complete" output mode on the console.

Choosing between streaming and batch processing depends on various factors such as the nature of the data and specific use case requirements. Both methods have their pros and cons, allowing data engineers and scientists to select the optimal processing mode tailored to their exact needs.

Furthermore, comparing metrics between these approaches provides valuable insights for informed decision-making in data processing workflows.

8. References

[1]https://medium.com/analytics-vidhya/apache-sparkstructured-streaming-with-pyspark-b4a054a7947d

[2]https://stackoverflow.com/questions/65809459/syntaxerror-on-self-async-when-running-python-kafka-producer

[3]https://towardsdatascience.com/how-to-build-asimple-kafka-producer-and-consumer-with-pythona967769c4742

[4] https://www.btelligent.com/en/blog/how-to-csv-tokafka-with-python

[5] https://lorenagongang.com/getting-started-with-kafka-tw itter-streaming-with-apache-kafka

[6] https://sparkbyexamples.com/pyspark/select-columns-fr om-pyspark-dataframe/