PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage
Bengaluru 560085

Department of Computer Science and Engineering
B. Tech. CSE - 6th Semester
Jan – May 2023

UE20CS343

DATABASE TECHNOLOGIES (DBT)

Project Report

Stream and Batch processing of Twitter data using Apache Spark and Kafka

TEAM #: 338_345_351_378
PES1UG20CS338: Rishab Agrawal
PES1UG20CS345: Rohan C
PES1UG20CS351: Rohan N
PES1UG20CS378: Sanjana Swaminathan

Class of Prof. Raghu B. A. Rao

Table of Contents

1. Introduction

The objective of this project is to assess the capabilities of Apache Spark and Kafka in processing streaming data. This involves executing multiple workloads, such as Spark SQL queries to perform actions, transformations, or aggregations on input data. The input data for this project is Twitter feeds, which are stored in a MySQL Database. The data is continuously streamed from the MySQL database at random intervals and processed by Kafka. Kafka produces four topics, each of which is the result of a particular SQL query. The four topics considered in this project are: tweets, hashtags, top_tweets, and top_hashtags.

Subscribers, including both streaming and batch consumers, randomly subscribe to a particular topic. The same queries are run on the same data in both streaming and batch modes. The accuracy and performance of both modes are compared. The computation examples include counting tweets within the window and applying aggregate functions on numeric data within each tumbling window. The window size of the project will be significant and suitable for the chosen domain problem, such as 15-30 minutes of tweets as one window.

2. Installation of Software [include version #s and URLs]

a) **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

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

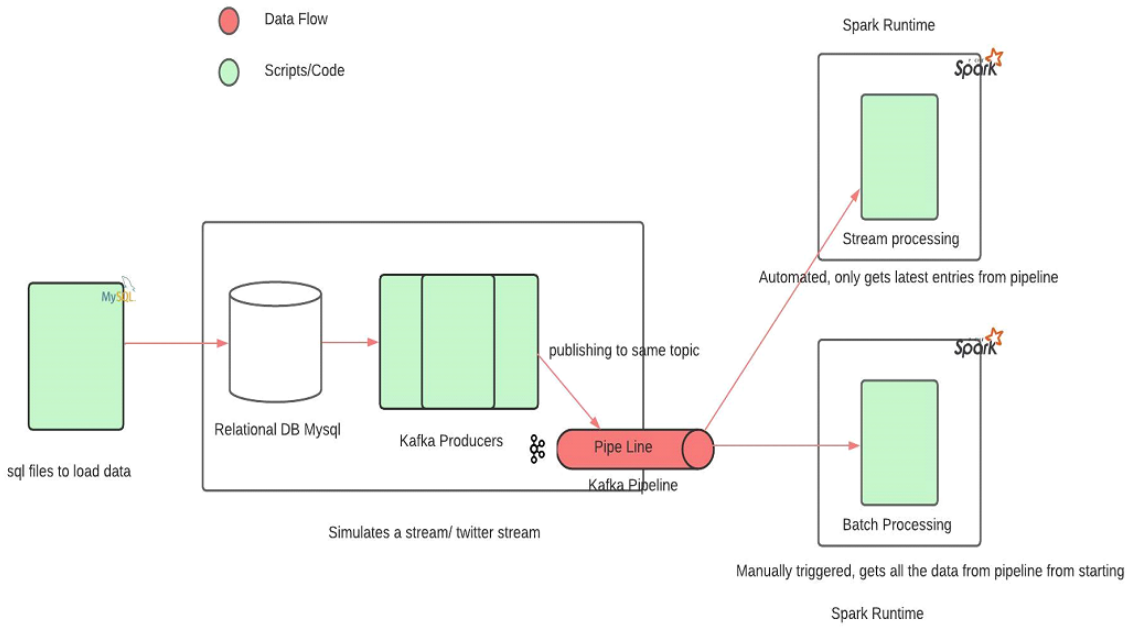
c) **Pyspark (3.4.0)**: A streaming engine just like apache kafka, but exclusively meant for python and its associated functionalities.
TO INSTALL : pip install pyspark==”3.4.0”

d) MySQL-8.0.32
Link of our remote hosted database :
https://www.db4free.net/phpMyAdmin/index.php?route=/

3) Problem Description

The primary challenge that this project addresses is the processing of large volumes of Twitter data and performing various actions, transformations, and aggregations on it. To tackle this challenge, the project involves creating multiple Kafka topics, including tweets, hashtags, top_tweets, and top_hashtags, and continuously publishing data from the MySQL database to these topics. The architecture ensures that Kafka streams data from the database in real-time, simulating Twitter API-like performance. The project aims to showcase the efficient use of Apache Spark and Kafka for processing and analyzing real-time streaming data, specifically Twitter feeds, to gain valuable insights. The project also demonstrates the use of Spark for batch processing queries. Additionally, we see Spark handling the subscription aspect of the model. Finally, the project includes a comparison between stream and batch processing of topics, as well as the application of different types of windows.

4) **Architecture Diagram**



5) **Input Data**

a) **Source**

Twitter dataset collected across multiple sources in the form of .CSV files from Kaggle

b) **Description**

This dataset contains the actual format of a general tweet message which includes the following metadata : “id, conversation_id, created_at, date, time, timezone, user_id, username, name, place, tweet language, mentions, urls, photos, replies_count, retweets_count, likes_count, hashtags, cashtags, link, retweet, quote_url, video thumbnail, near, geo, source, user_rt_id,

user_rt, retweet_id, reply_to, retweet_date, translate, trans_src and trans_dest

But, we choose only the relevant data, that we need for our project.

The following is the database and table , that is obtained form this dataset

```
CREATE TABLE tweets (tweet_id BIGINT PRIMARY KEY,
tweet TEXT, date_time VARCHAR(40), language
VARCHAR(10));

CREATE TABLE hashtags (hashtag_id BIGINT PRIMARY
KEY, hashtag VARCHAR(300) collate utf8mb4_bin);

CREATE TABLE tweet_hashtags (tweet_id BIGINT,
hashtag_id BIGINT, count BIGINT, PRIMARY
KEY(tweet_id, hashtag_id), FOREIGN KEY(tweet_id)
REFERENCES tweets(tweet_id) ON DELETE CASCADE,
FOREIGN KEY(hashtag_id) REFERENCES
hashtags(hashtag_id) ON DELETE CASCADE);
```

The Kafka Topics (STREAMING + PRODUCER)

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
TimestampType
from pandas import *
import random
import time
topic = ""
```

```
print("WELCOME TO SPARK STREAMING CONSUMER")
print("Enter the topic you want to choose : \n 1. top_tweets \n 2. top_hashtags \n
3. tweets \n 4. hashtags \n 5. random topic\n")
choice = random.randint(1, 4)
print("The chosen topic is : ", choice)
if choice == 1:
    topic = "top_tweets"
elif choice == 2:
    topic = "top_hashtags"
elif choice == 3:
    topic = "tweets"
elif choice == 4:
    topic = "hashtags"

# create a SparkSession object
spark = SparkSession.builder \
    .appName("MySparkApp") \
    .master("local[*]") \
    .getOrCreate()

# Define the schema for the incoming data
schema = StructType([
    StructField("id", IntegerType()),
    StructField("text", StringType()),
    StructField("date_time", TimestampType()),
    StructField("language", StringType()),
    StructField("hashtags", StringType())
])

# Create the streaming dataframe
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", topic) \
    .option("startingOffsets", "latest") \
    .load() \
    .selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
```

```
    .select("data.*")

# Define the window and group by clauses
windowedCounts = df \
    .withWatermark("date_time", "30 minutes") \
    .groupBy(
        window(col("date_time"), "30 minutes"),
        col("hashtags")
    ) \
    .agg(count("id").alias("tweet_count"))

# Sort the data in ascending order by the window start time and hashtags
sortedCounts = windowedCounts \
    .sort(
        col("window.start").asc(),
        col("hashtags").asc()
    )

# Write the output to the console
console_query = sortedCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .option("truncate", "false") \
    .start()

# Write the output to a CSV file
csv_query = sortedCounts \
    .writeStream \
    .outputMode("complete") \
    .foreachBatch(lambda df, epochId: df.toPandas().to_csv("output.csv",
index=False, header=True)) \
    .start()

# Wait for the queries to terminate
console_query.awaitTermination()
csv_query.awaitTermination()

time.sleep(100)
```

```
# Stop the queries
console_query.stop()
csv_query.stop()
```

6) Streaming Mode Experiment

a) Windows

The code sets a sliding window of 15 minutes and groups the data within that window based on the specified topics. The choice of a 15-minute window indicates that any query or data beyond this time frame is considered outdated and will be discarded.

b) Workloads

The code subscribes to a Kafka topic (selected randomly), processes the data in a streaming fashion using Pyspark, calculates the count of tweets per hashtag within a 30-minute window, sorts the results by window start time and hashtag, and writes the output to both the console and a CSV file. This approach utilizes Pyspark to subscribe to the Kafka topic provided.

c) Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StructType, StructField, IntegerType, StringType,
TimestampType
from pandas import *
import random
import time
topic = ""
print("WELCOME TO SPARK STREAMING CONSUMER")
print("Enter the topic you want to choose : \n 1. top_tweets \n 2. top_hashtags \n
3. tweets \n 4. hashtags \n 5. random topic\n")
```

```
choice = random.randint(1, 4)
print("The chosen topic is : ", choice)
if choice == 1:
    topic = "top_tweets"
elif choice == 2:
    topic = "top_hashtags"
elif choice == 3:
    topic = "tweets"
elif choice == 4:
    topic = "hashtags"

# create a SparkSession object
spark = SparkSession.builder \
    .appName("MySparkApp") \
    .master("local[*]") \
    .getOrCreate()

# Define the schema for the incoming data
schema = StructType([
    StructField("id", IntegerType()),
    StructField("text", StringType()),
    StructField("date_time", TimestampType()),
    StructField("language", StringType()),
    StructField("hashtags", StringType())
])

# Create the streaming dataframe
df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", topic) \
    .option("startingOffsets", "latest") \
    .load() \
    .selectExpr("CAST(value AS STRING)") \
    .select(from_json(col("value"), schema).alias("data")) \
    .select("data.*")

# Define the window and group by clauses
```

```
windowedCounts = df \
    .withWatermark("date_time", "15 minutes") \
    .groupBy(
        window(col("date_time"), "15 minutes"),
        col("hashtags")
    ) \
    .agg(count("id").alias("tweet_count"))

# Sort the data in ascending order by the window start time and hashtags
sortedCounts = windowedCounts \
    .sort(
        col("window.start").asc(),
        col("hashtags").asc()
    )

# Write the output to the console
console_query = sortedCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .option("truncate", "false") \
    .start()

# Write the output to a CSV file
csv_query = sortedCounts \
    .writeStream \
    .outputMode("complete") \
    .foreachBatch(lambda df, epochId: df.toPandas().to_csv("output.csv",
index=False, header=True)) \
    .start()

# Wait for the queries to terminate
console_query.awaitTermination()
csv_query.awaitTermination()

time.sleep(100)

# Stop the queries
console_query.stop()
```

```
csv_query.stop()
```

d) Inputs and corresponding results

```
23/04/24 20:00:55 INFO WriteToDataSourceV2Exec: Start processing data source write support: MicroBatchWrite[epoch: 0, writer: ConsoleWriter[nu
mRows=20, truncate=false]]. The input RDD has 1 partitions.
23/04/24 20:00:55 INFO SparkContext: Starting job: start at NativeMethodAccessorImpl.java:0
23/04/24 20:00:55 INFO DAGScheduler: Registering RDD 32 (start at NativeMethodAccessorImpl.java:0) as input to shuffle 2
23/04/24 20:00:55 INFO DAGScheduler: Got job 2 (start at NativeMethodAccessorImpl.java:0) with 1 output partitions
23/04/24 20:00:55 INFO DAGScheduler: Final stage: ResultStage 6 (start at NativeMethodAccessorImpl.java:0)
23/04/24 20:00:55 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 5)
23/04/24 20:00:55 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 5)
23/04/24 20:00:55 INFO DAGScheduler: Submitting ShuffleMapStage 5 (MapPartitionsRDD[32] at start at NativeMethodAccessorImpl.java:0), which ha
s no missing parents
23/04/24 20:00:55 INFO CheckpointFileManager: Writing atomically to file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/0/_metada
ta/schema using temp file file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/0/_metadata/.schema.9e7471b6-7fdf-48c0-966e-93127ad
d47d5.TID200.tmp
23/04/24 20:00:56 INFO MemoryStore: Block broadcast_6 stored as values in memory (estimated size 75.6 KiB, free 433.2 MiB)
23/04/24 20:00:56 INFO MemoryStore: Block broadcast_6_piece0 stored as bytes in memory (estimated size 33.1 KiB, free 433.1 MiB)
23/04/24 20:00:56 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on localhost:38855 (size: 33.1 KiB, free: 434.2 MiB)
23/04/24 20:00:56 INFO SparkContext: Created broadcast 6 from broadcast at DAGScheduler.scala:1535
23/04/24 20:00:56 INFO DAGScheduler: Submitting 200 missing tasks from ShuffleMapStage 5 (MapPartitionsRDD[32] at start at NativeMethodAccesso
rImpl.java:0) (first 15 tasks are for partitions Vector(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14))
23/04/24 20:00:56 INFO TaskSchedulerImpl: Adding task set 5.0 with 200 tasks resource profile 0
23/04/24 20:00:56 INFO CheckpointFileManager: Renamed temp file file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/0/_metadata/.
schema.9e7471b6-7fdf-48c0-966e-93127add47d5.TID200.tmp to file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/0/_metadata/schema
23/04/24 20:00:56 INFO StateStore: Retrieved reference to StateStoreCoordinator: org.apache.spark.sql.execution.streaming.state.StateStoreCoor
dinatorRef@3fe27957
23/04/24 20:00:56 INFO StateStore: Reported that the loaded instance StateStoreProviderId(StateStoreId(file:/tmp/temporary-f49ad1be-2fc8-44c8-
bfb6-bfc24a5356e3/state,0,0,default),53c92e78-6144-4b5f-ae9d-4b3bfb5c96c8) is active
23/04/24 20:00:56 INFO HDFSBackedStateStoreProvider: Retrieved version 0 of HDFSStateStoreProvider[id = (op=0,part=0),dir = file:/tmp/temporar
y-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/0] for update
23/04/24 20:00:56 INFO BlockManagerInfo: Removed broadcast_4_piece0 on localhost:38855 in memory (size: 32.8 KiB, free: 434.2 MiB)
23/04/24 20:00:56 INFO StateStore: Retrieved reference to StateStoreCoordinator: org.apache.spark.sql.execution.streaming.state.StateStoreCoor
```

```
        org.slf4j#slf4j-api;1.7.30 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.8.4 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   13  |   0   |   0   |   0   ||   13  |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-e52bd622-e4e9-4a48-88c5-7bc2c339e5c5
        confs: [default]
        0 artifacts copied, 13 already retrieved (0kB/28ms)
23/04/24 19:59:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
WELCOME TO SPARK STREAMING CONSUMER
Enter the topic you want to choose :
 1. top_tweets
 2. top_hashtags
 3. tweets
 4. hashtags
 5. random topic

The chosen topic is :  1
23/04/24 19:59:22 INFO SparkContext: Running Spark version 3.4.0
23/04/24 19:59:22 INFO ResourceUtils: ==============================================================
23/04/24 19:59:22 INFO ResourceUtils: No custom resources configured for spark.driver.
23/04/24 19:59:22 INFO ResourceUtils: ==============================================================
23/04/24 19:59:22 INFO SparkContext: Submitted application: MySparkApp
23/04/24 19:59:23 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , ve
ndor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Ma
p(cpus -> name: cpus, amount: 1.0)
```

```
smsraj@SMS-DESKTOP:~/Documents/DBT_PROJECT$ spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.3 spark_streaming_consumer.
py
:: loading settings :: url = jar:file:/opt/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/smsraj/.ivy2/cache
The jars for the packages stored in: /home/smsraj/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-e52bd622-e4e9-4a48-88c5-7bc2c339e5c5;1.0
        confs: [default]
        found org.apache.spark#spark-sql-kafka-0-10_2.12;3.2.3 in central
        found org.apache.spark#spark-token-provider-kafka-0-10_2.12;3.2.3 in central
        found org.apache.kafka#kafka-clients;2.8.1 in central
        found org.lz4#lz4-java;1.7.1 in central
        found org.xerial.snappy#snappy-java;1.1.8.4 in central
        found org.slf4j#slf4j-api;1.7.30 in central
        found org.apache.hadoop#hadoop-client-runtime;3.3.1 in central
        found org.spark-project.spark#unused;1.0.0 in central
        found org.apache.hadoop#hadoop-client-api;3.3.1 in central
        found org.apache.htrace#htrace-core4;4.1.0-incubating in central
        found commons-logging#commons-logging;1.1.3 in central
        found com.google.code.findbugs#jsr305;3.0.0 in central
        found org.apache.commons#commons-pool2;2.6.2 in central
:: resolution report :: resolve 2491ms :: artifacts dl 120ms
        :: modules in use:
        com.google.code.findbugs#jsr305;3.0.0 from central in [default]
        commons-logging#commons-logging;1.1.3 from central in [default]
        org.apache.commons#commons-pool2;2.6.2 from central in [default]
        org.apache.hadoop#hadoop-client-api;3.3.1 from central in [default]
        org.apache.hadoop#hadoop-client-runtime;3.3.1 from central in [default]
        org.apache.htrace#htrace-core4;4.1.0-incubating from central in [default]
        org.apache.kafka#kafka-clients;2.8.1 from central in [default]
```

```
23/04/24 20:02:26 INFO WriteToDataSourceV2Exec: Data source write support MicroBatchWrite[epoch: 0, writer: ConsoleWriter[numRows=20, truncate
=false]] is committing.
-------------------------------------------
23/04/24 20:02:26 INFO StateStore: Retrieved reference to StateStoreCoordinator: org.apache.spark.sql.execution.streaming.state.StateStoreCoor
dinatorRef@6be27a08
Batch: 0
-------------------------------------------
23/04/24 20:02:26 INFO StateStore: Reported that the loaded instance StateStoreProviderId(StateStoreId(file:/tmp/temporary-f49ad1be-2fc8-44c8-
bfb6-bfc24a5356e3/state,0,4,default),53c92e78-6144-4b5f-ae9d-4b3bfb5c96c8) is active
23/04/24 20:02:26 INFO HDFSBackedStateStoreProvider: Retrieved version 0 of HDFSStateStoreProvider[id = (op=0,part=4),dir = file:/tmp/temporar
y-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/4] for update
23/04/24 20:02:26 INFO StateStore: Retrieved reference to StateStoreCoordinator: org.apache.spark.sql.execution.streaming.state.StateStoreCoor
dinatorRef@4b2b47e9
23/04/24 20:02:26 INFO StateStore: Reported that the loaded instance StateStoreProviderId(StateStoreId(file:/tmp/temporary-f49ad1be-2fc8-44c8-
bfb6-bfc24a5356e3/state,0,4,default),53c92e78-6144-4b5f-ae9d-4b3bfb5c96c8) is active
23/04/24 20:02:26 INFO HDFSBackedStateStoreProvider: Retrieved version 0 of HDFSStateStoreProvider[id = (op=0,part=4),dir = file:/tmp/temporar
y-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/4] for readonly
23/04/24 20:02:26 INFO ShuffleBlockFetcherIterator: Getting 0 (0.0 B) non-empty blocks including 0 (0.0 B) local and 0 (0.0 B) host-local and
0 (0.0 B) push-merged-local and 0 (0.0 B) remote blocks
23/04/24 20:02:26 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
23/04/24 20:02:26 INFO CheckpointFileManager: Writing atomically to file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/4/1.delta
 using temp file file:/tmp/temporary-f49ad1be-2fc8-44c8-bfb6-bfc24a5356e3/state/0/4/.1.delta.e808562d-d8a4-4b12-be66-c11ab97fb08d.TID605.tmp
+------+--------+-----------+
|window|hashtags|tweet_count|
+------+--------+-----------+
+------+--------+-----------+

23/04/24 20:02:26 INFO WriteToDataSourceV2Exec: Data source write support MicroBatchWrite[epoch: 0, writer: ConsoleWriter[numRows=20, truncate
=false]] committed.
23/04/24 20:02:26 INFO CheckpointFileManager: Writing atomically to file:/tmp/temporary-a942d0c9-014f-4899-8e80-8b68d57a6667/commits/0 using t
```


ab-4067-9925-9aaa7f3f4506.tmp to file:/tmp/temporary-a942d0c9-014f-4899-8e80-8b68d57a6667
3/04/24 20:02:27 INFO MicroBatchExecution: Streaming query made progress: {
"id" : "28b98120-4c63-4e5d-a5d1-f961ba3bac39",
"runId" : "bc3c784f-4f32-4aa0-b647-70dc69c796fa",
"name" : null,
"timestamp" : "2023-04-24T14:29:41.054Z",
"batchId" : 0,
"numInputRows" : 0,
"inputRowsPerSecond" : 0.0,
"processedRowsPerSecond" : 0.0,
"durationMs" : {
"addBatch" : 159199,
"commitOffsets" : 334,
"getBatch" : 8,
"latestOffset" : 3462,
"queryPlanning" : 2523,
"triggerExecution" : 166065,
"walCommit" : 493
},
"eventTime" : {
"watermark" : "1970-01-01T00:00:00.000Z"
},
"stateOperators" : [{
"operatorName" : "stateStoreSave",
"numRowsTotal" : 0,
"numRowsUpdated" : 0,
"allUpdatesTimeMs" : 3254,
"numRowsRemoved" : 0,
"allRemovalsTimeMs" : 0,
"commitTimeMs" : 138670,


"numRowsRemoved" : 0,
"allRemovalsTimeMs" : 0,
"commitTimeMs" : 138670,
"memoryUsedBytes" : 89600,
"numRowsDroppedByWatermark" : 0,
"numShufflePartitions" : 400,
"numStateStoreInstances" : 400,
"customMetrics" : {
"loadedMapCacheHitCount" : 0,
"loadedMapCacheMissCount" : 0,
"stateOnCurrentVersionSizeBytes" : 32000
}
}],
"sources" : [{
"description" : "KafkaV2[Subscribe[top_tweets]]",
"startOffset" : null,
"endOffset" : {
"top_tweets" : {
"0" : 5180
}
},
"latestOffset" : {
"top_tweets" : {
"0" : 5180
}
},
"numInputRows" : 0,
"inputRowsPerSecond" : 0.0,
"processedRowsPerSecond" : 0.0,
"metrics" : {

7) Batch Mode Experiment

a) Description

This program uses Spark Structured Streaming to stream data from a Kafka topic named "tweets". 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 results are then output to the console using the complete output mode.

b) Data size

The data size for each batch is 20 rows. The actual size of the data is determined by the data produced and stored in the corresponding Kafka topic.

c) Code

```
import pyspark
from pyspark.sql.functions import *

spark = pyspark.sql.SparkSession.builder.appName("solution").getOrCreate()

# Batch :
df = spark.read.format("kafka").option("kafka.bootstrap.servers",
"localhost:9092").option("startingOffsets",
"earliest").option("subscribe","test").option("includeTimestamp","true").load
()
query =
df.select((df.key).alias("HashTag"),(df.timestamp).alias("time")).groupBy(win
dow("time","20 seconds"),"HashTag").count().sort(desc("window"))
query = query.selectExpr("CAST(window AS STRING)","CAST(HashTag AS STRING)",
"CAST(count AS STRING)")
query.show()
```

d) Inputs and corresponding results

```
23/04/24 20:11:13 INFO MicroBatchExecution: Starting [id = eb8b7a76-d542-49cf-96d3-d79cb50eadc2, runId = cebb8346-6b85-4a1d-b334-7f58be64a4
. Use file:/tmp/temporary-250cdd5c-7d44-47ae-84da-3fdba0b11d41 to store the query checkpoint.
23/04/24 20:11:13 INFO MicroBatchExecution: Reading table [org.apache.spark.sql.kafka010.KafkaSourceProvider$KafkaTable@43b07bb2] from Data
rceV2 named 'kafka' [org.apache.spark.sql.kafka010.KafkaSourceProvider@7589a8bd]
23/04/24 20:11:13 INFO OffsetSeqLog: BatchIds found from listing:
23/04/24 20:11:13 INFO OffsetSeqLog: BatchIds found from listing:
23/04/24 20:11:13 INFO MicroBatchExecution: Starting new streaming query.
23/04/24 20:11:13 INFO MicroBatchExecution: Stream started from {}
23/04/24 20:11:15 INFO AdminClientConfig: AdminClientConfig values:
        bootstrap.servers = [localhost:9092]
        client.dns.lookup = use_all_dns_ips
        client.id =
        connections.max.idle.ms = 300000
        default.api.timeout.ms = 60000
        metadata.max.age.ms = 300000
        metric.reporters = []
        metrics.num.samples = 2
        metrics.recording.level = INFO
        metrics.sample.window.ms = 30000
        receive.buffer.bytes = 65536
        reconnect.backoff.max.ms = 1000
        reconnect.backoff.ms = 50
        request.timeout.ms = 30000
        retries = 2147483647
        retry.backoff.ms = 100
        sasl.client.callback.handler.class = null
        sasl.jaas.config = null
        sasl.kerberos.kinit.cmd = /usr/bin/kinit
        sasl.kerberos.min.time.before.relogin = 60000
        sasl.kerberos.service.name = null
```

```
WELCOME TO SPARK BATCH CONSUMER
Enter the topic you want to choose :
 1. top_tweets
 2. top_hashtags
 3. tweets
 4. hashtags
 5. random topic

The chosen topic is :  1
23/04/24 20:10:56 INFO SparkContext: Running Spark version 3.4.0
23/04/24 20:10:56 INFO ResourceUtils: ==============================================================
23/04/24 20:10:56 INFO ResourceUtils: No custom resources configured for spark.driver.
23/04/24 20:10:56 INFO ResourceUtils: ==============================================================
23/04/24 20:10:56 INFO SparkContext: Submitted application: TwitterCount
23/04/24 20:10:56 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 1, script: , ve
ndor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Ma
p(cpus -> name: cpus, amount: 1.0)
23/04/24 20:10:56 INFO ResourceProfile: Limiting resource is cpu
23/04/24 20:10:56 INFO ResourceProfileManager: Added ResourceProfile id: 0
23/04/24 20:10:57 INFO SecurityManager: Changing view acls to: smsraj
23/04/24 20:10:57 INFO SecurityManager: Changing modify acls to: smsraj
```

```
only showing top 20 rows

23/04/24 20:12:38 INFO WriteToDataSourceV2Exec: Data source write support MicroBatchWrite[epoch: 0, writer: ConsoleWriter[numRows=20, trun
=false]] committed.
23/04/24 20:12:38 INFO WatermarkTracker: Updating event-time watermark from 0 to 1682346973815 ms
23/04/24 20:12:38 INFO CheckpointFileManager: Writing atomically to file:/tmp/temporary-250cdd5c-7d44-47ae-84da-3fdba0b11d41/commits/0 usi
emp file file:/tmp/temporary-250cdd5c-7d44-47ae-84da-3fdba0b11d41/commits/.0.0ae082cc-324e-4693-a0ed-60e2aa3e8b22.tmp
23/04/24 20:12:39 INFO CheckpointFileManager: Renamed temp file file:/tmp/temporary-250cdd5c-7d44-47ae-84da-3fdba0b11d41/commits/.0.0ae082
24e-4693-a0ed-60e2aa3e8b22.tmp to file:/tmp/temporary-250cdd5c-7d44-47ae-84da-3fdba0b11d41/commits/0
23/04/24 20:12:39 INFO MicroBatchExecution: Streaming query made progress: {
  "id" : "eb8b7a76-d542-49cf-96d3-d79cb50eadc2",
  "runId" : "cebb8346-6b85-4a1d-b334-7f58be64a4b9",
  "name" : null,
  "timestamp" : "2023-04-24T14:41:13.705Z",
  "batchId" : 0,
  "numInputRows" : 13950,
  "inputRowsPerSecond" : 0.0,
  "processedRowsPerSecond" : 162.79992531042853,
  "durationMs" : {
    "addBatch" : 76023,
    "commitOffsets" : 515,
    "getBatch" : 175,
    "latestOffset" : 5508,
    "queryPlanning" : 2801,
    "triggerExecution" : 85684,
    "walCommit" : 527
  },
  "eventTime" : {
    "avg" : "2023-04-24T13:12:34.823Z",
```

```
23/04/24 20:12:38 INFO WriteToDataSourceV2Exec: Data source write support MicroBatchWrite[epoch: 0, writer: ConsoleWriter[numRows=20, truncate
=false]] is committing.
-------------------------------------------
Batch: 0
-------------------------------------------
23/04/24 20:12:38 INFO CodeGenerator: Code generated in 5.828592 ms
23/04/24 20:12:38 INFO CodeGenerator: Code generated in 35.99906 ms
+------------------------------------------+--------+-----------+
|window                                    |language|tweet_count|
+------------------------------------------+--------+-----------+
|{2023-04-24 19:20:00, 2023-04-24 19:25:00}|null    |425        |
|{2023-04-24 17:30:00, 2023-04-24 17:35:00}|null    |350        |
|{2023-04-24 19:05:00, 2023-04-24 19:10:00}|null    |450        |
|{2023-04-24 17:10:00, 2023-04-24 17:15:00}|null    |300        |
|{2023-04-24 19:15:00, 2023-04-24 19:20:00}|null    |450        |
|{2023-04-24 17:15:00, 2023-04-24 17:20:00}|null    |300        |
|{2023-04-24 18:00:00, 2023-04-24 18:05:00}|null    |350        |
|{2023-04-24 19:50:00, 2023-04-24 19:55:00}|null    |350        |
|{2023-04-24 18:15:00, 2023-04-24 18:20:00}|null    |450        |
|{2023-04-24 17:40:00, 2023-04-24 17:45:00}|null    |375        |
|{2023-04-24 20:00:00, 2023-04-24 20:05:00}|null    |400        |
|{2023-04-24 20:10:00, 2023-04-24 20:15:00}|null    |100        |
|{2023-04-24 17:20:00, 2023-04-24 17:25:00}|null    |400        |
|{2023-04-24 19:55:00, 2023-04-24 20:00:00}|null    |325        |
|{2023-04-24 17:05:00, 2023-04-24 17:10:00}|null    |150        |
|{2023-04-24 17:45:00, 2023-04-24 17:50:00}|null    |350        |
|{2023-04-24 18:05:00, 2023-04-24 18:10:00}|null    |375        |
|{2023-04-24 19:25:00, 2023-04-24 19:30:00}|null    |375        |
|{2023-04-24 18:25:00, 2023-04-24 18:30:00}|null    |425        |
|{2023-04-24 18:10:00, 2023-04-24 18:15:00}|null    |425        |
```

```
},
"eventTime" : {
  "avg" : "2023-04-24T13:12:34.823Z",
  "max" : "2023-04-24T14:41:13.815Z",
  "min" : "2023-04-24T11:38:16.387Z",
  "watermark" : "1970-01-01T00:00:00.000Z"
},
"stateOperators" : [ {
  "operatorName" : "stateStoreSave",
  "numRowsTotal" : 38,
  "numRowsUpdated" : 38,
  "allUpdatesTimeMs" : 7840,
  "numRowsRemoved" : 0,
  "allRemovalsTimeMs" : 0,
  "commitTimeMs" : 83236,
  "memoryUsedBytes" : 55584,
  "numRowsDroppedByWatermark" : 0,
  "numShufflePartitions" : 200,
  "numStateStoreInstances" : 200,
  "customMetrics" : {
    "loadedMapCacheHitCount" : 0,
    "loadedMapCacheMissCount" : 0,
    "stateOnCurrentVersionSizeBytes" : 26784
  }
} ],
"sources" : [ {
  "description" : "KafkaV2[Subscribe[tweets]]",
  "startOffset" : null,
  "endOffset" : {
    "tweets" : {
```


```
    }
  } ],
  "sources" : [ {
    "description" : "KafkaV2[Subscribe[tweets]]",
    "startOffset" : null,
    "endOffset" : {
      "tweets" : {
        "0" : 13950
      }
    },
    "latestOffset" : {
      "tweets" : {
        "0" : 13950
      }
    },
    "numInputRows" : 13950,
    "inputRowsPerSecond" : 0.0,
    "processedRowsPerSecond" : 162.79992531042853,
    "metrics" : {
      "avgOffsetsBehindLatest" : "0.0",
      "maxOffsetsBehindLatest" : "0",
      "minOffsetsBehindLatest" : "0"
    }
  } ],
  "sink" : {
    "description" : "org.apache.spark.sql.execution.streaming.ConsoleTable$@34ee93ed",
    "numOutputRows" : 38
  }
}
```

8) **Comparison of Streaming and Batch modes**

Results and Discussion

In this case, we have used both streaming and batch processing to subscribe to only Topic-1, which is "top_tweets". We have collected statistics on computation, aggregation, and counting by window, among others. We have observed that the processed row rate in seconds is close to zero in streaming, indicating that the processing takes place in real-time. In contrast, in batch processing, the number is in the few hundreds, indicating that we are processing the data after some time has elapsed. The number of rows input to the process models also follows a similar pattern.

Additionally, the memory usage and commit time are lower in streaming processing compared to batch processing, which is due to the large amount of data processed in batch processing that is not applicable to streaming processing. These are some of the available statistics that we can use to compare the results of both processes. However, there are many other parameters that we can use for evaluation.

9) Conclusion

In summary, both streaming and batch processing play crucial roles in data processing, but their applications differ based on specific use cases.

Streaming is ideal for processing data in real-time or near real-time, while batch processing is more suitable for handling large volumes of data at once.

In the context of the code example provided, batch processing was employed to process a significant number of tweets obtained from a Kafka topic within a designated time frame. The data was grouped by language and timestamp window and computed to provide the total count of tweets, which was then output to the console in the "complete" output mode.

Ultimately, choosing between streaming and batch processing is determined by the unique requirements of the data being processed and the specific use case. It is up to the data engineer or scientist to decide which processing mode is most suitable for their needs. The comparison metrics discussed above help in assessing the differences between these two modes.

10) Bibliography

https://medium.com/analytics-vidhya/apache-spark-structured-streaming-with-pyspark-b4a054a7947d

https://towardsdatascience.com/how-to-build-a-simple-kafka-producer-and-consumer-with-python-a967769c4742

https://linuxhint.com/install-apache-kafka-ubuntu-22-04/

https://www.btelligent.com/en/blog/how-to-csv-to-kafka-with-python/

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