Analyze Viewership of Videos

Mid Program Project - 2

1. Business Challenge/Requirement:

A leading video on demand company envisions expanding its customer base with an aim to enhance its business revenue. Taking the first leap towards this goal mandates the understanding of its user behavior online. To do so, the company turns to Big Data to analyze and interpret a huge amount of data -- collected from the mobile app as well as the website -- pertaining to user behavior. The insights garnered from the data will help them formulate efficient strategies and tailor the business model that can bring them closer to their goals. The analysis, though, will aid them in achieving the following:

1. Keeping track of user/viewer behavior will enable them to customize the services to the needs of the viewers. Targeting the viewers with custom services will potentially stimulate more customers into opting for paid subscription.

2. The calculation process for royalty to become hassle-free for video creators. Added to this, the company also plans to restructure the incentive scheme by rewarding more to video creators who produce high-quality content. The more they are paid, the higher the quality content they will create, which, in turn, will enhance their bottom line multifold.

2. Goal of the Project:

To create an end-to-end data pipeline that analyses the viewership data and helps the company comprehend the user behavior better.

4. Tool used:

1. **Spark** - used for processing and enrichment of batch data.
2. **Spark Streaming -** used for processing and enrichment of streaming data.
3. **MySQL -** used for creating lookup tables for data enrichment in spark programs.
4. **Hive -** used for all the problem statement queries and output storage.
5. **SQOOP -** for moving lookup table in MySQL to Hive table for querying problem statements
6. **Scala -** used as language to access the Spark API.

3. Steps:

1. Copy the datasets and data generator script from the given location to the provided edge node.
2. Execute the data generator script which will create data used for streaming processing.
3. Create a lookup table in MySQL and load the data which is copied to the edge node. I have used MySQL instead of Hbase because we have only a few records in the lookup table. If we have billions of records then instead of using MySQL Hbase was the better option. Lookup tables are used in the Spark program for data enrichment.
4. Used the Spark program to process the batch data available in CSV and XML formats, where both datasets are merged, enriched and loaded into HDFS in ORC format, so that it can be easily available in the Hive table.
5. Also, the Spark-Streaming program is used to process the streaming data generated from the mobile applications in JSON format, where the JSON data is enriched and loaded into HDFS location. Location, structure, fields and data format for all the sources are the same so that it can be used in the same Hive table.
6. Created the Hive table for the video-play data and used the location of the table where all the enriched data was stored.
7. Created the lookup tables in Hive for getting solutions to problem statements.

IMPLEMENTATION DETAILS

1. Setup data from given location to edge node and HDFS as required

```
-- Create Project Folder
mkdir mid-project2
cd mid-project2

-- Copy data generator from HDFS
hadoop fs -get /bigdatapgp/common_folder/midproject2/data_generator/
cd data_generator

-- Get dataset from HDFS to file server
hadoop fs -get /bigdatapgp/common_folder/midproject2/datasets

-- Run data generator script
nohup python2 data_gen_execute.py &
cd ..

-- Create folder in HDFS
hadoop fs -mkdir mid-project2

-- Create folder in HDFS
hadoop fs -put datasets/ mid-projects/
```

2. Create lookup table in MySQL

```
USE labuser_database;
-- Create table for Channel Geocd
CREATE TABLE IF NOT EXISTS channel_geocd_788309(channel_id INT AUTO_INCREMENT
PRIMARY KEY, geo_cd  varchar(10));

-- Create table for Video creator
CREATE TABLE IF NOT EXISTS video_creator_788309(creator_id INT AUTO_INCREMENT
PRIMARY KEY,user_id varchar(50));

-- Create table for use subscriptions
CREATE TABLE IF NOT EXISTS user_subscn_788309(user_id
INT,subscription_start_date varchar(50),subscription_end_date varchar(50));

-- Create table for User creator
CREATE TABLE IF NOT EXISTS user_creator_788309(user_id INT,creator_id
varchar(255));

-- Load data for Channel Geocd
LOAD DATA LOCAL INFILE 'mid-project2/datasets/channel-geocd.csv' INTO TABLE
channel_geocd_788309 FIELDS TERMINATED BY ',';

-- Load data for Video Creator
LOAD DATA LOCAL INFILE 'mid-project2/datasets/video-creator.csv' INTO TABLE
video_creator_788309 FIELDS TERMINATED BY ',';

-- Load data for User Subscriptions
LOAD DATA LOCAL INFILE 'mid-project2/datasets/user-subscn.csv' INTO TABLE
user_subscn_788309 FIELDS TERMINATED BY ',';

-- Load data for User Creator
LOAD DATA LOCAL INFILE 'mid-project2/datasets/user-creator.csv' INTO TABLE
user_creator_788309 FIELDS TERMINATED BY ',';
```

3. Code for batch ingestion of Video Play data written in Spark-Scala

```
import org.apache.spark.sql.functions.{col, when}
import org.apache.spark.sql.{DataFrame, SparkSession}

import java.util.Properties

object VideoBatchIngestion {
  def main(args: Array[String]):Unit = {

    // Video Play Data files
    val videoPlayFilePath =
"hdfs://nameservice1/user/edureka_788309/mid-project2/datasets/"
    val companyWebsiteVideoPlayFile = videoPlayFilePath + "video_plays.xml"
    val otherWebsiteVideoPlayFile = videoPlayFilePath + "video_plays.csv"

    // Create spark Session
    val spark = SparkSession.builder
      .appName("Batch Ingestion of Video Play Website Data ")
      .master("yarn")
      .getOrCreate()

    val sc = spark.sparkContext
    val conf = sc.hadoopConfiguration
    val fs = org.apache.hadoop.fs.FileSystem.get(conf)
    val xmlFile = fs.exists(new
org.apache.hadoop.fs.Path(companyWebsiteVideoPlayFile))

    // Validate the existence of XML File
    if (!xmlFile) {
      println("VIDEO PLAY FILE NOT FOUND OF COMPANY WEBSITE")
      return
    }

    // Validate the existence of CSV File
    val csvFile = fs.exists(new
org.apache.hadoop.fs.Path(companyWebsiteVideoPlayFile))
    if (!csvFile) {
      println("VIDEO PLAY FILE NOT FOUND OF OTHER WEBSITE")
      return
    }

    // Create dataframe from XML file
    val companyWebsiteDf: DataFrame =
spark.read.format("com.databricks.spark.xml")
      .option("rowTag", "record")
      .option("dateFormat", "dd/MM/yyy H:m:s")
      .option("mode", "DROPMALFORMED")
      .load(companyWebsiteVideoPlayFile)
```

```
    // Create dataframe from CSV file
    val otherWebsiteDfTemp: DataFrame = spark.read.format("csv")
      .option("header", "true")
      .option("inferSchema", "true")
      .option("dateFormat", "dd/MM/yyy H:m:s")
      .option("mode", "DROPMALFORMED")
      .option("delimiter", ",")
      .load(otherWebsiteVideoPlayFile)
    val otherWebsiteDf =
otherWebsiteDfTemp.select(col("channel_id").cast("long"),
col("creator_id").cast("long"), col("disliked").cast("boolean"),
col("geo_cd").cast("string"), col("liked").cast("boolean"),
col("minutes_played").cast("long"), col("timestamp").cast("string"),
col("user_id").cast("long"), col("video_end_type").cast("long"),
col("video_id").cast("long"))

    // Combine the data of XML and CSV
    val videoPlayDf: DataFrame = companyWebsiteDf.union(otherWebsiteDf)

    val channelGeoCdDf = spark.read.format("jdbc")
      .option("url", "jdbc:mysql://dbserver.edu.cloudlab.com")
      .option("dbtable", "labuser_database.channel_geocd_788309")
      .option("user", "edu_labuser").option("password", "edureka")
      .load()

    def getGeoCdFromChannelId(channelId: String) =
channelGeoCdDf.filter(col("channel_id") ===
channelId).select(col("geo_cd")).collect().map(_.getString(0)).mkString("")

    // Clean the records
    val cleanVideoPlayDf = videoPlayDf
      .withColumn("liked", when(col("liked") === null,
false).otherwise(col("liked")))
      .withColumn("disliked", when(col("disliked") === null,
false).otherwise(col("disliked")))
      .withColumn("video_end_type", when(col("video_end_type") =!= 0 &&
col("video_end_type") =!= 1 && col("video_end_type") =!= 2 &&
col("video_end_type") =!= 3, 3).otherwise(col("video_end_type")))
      .withColumn("geo_cd", when(col("geo_cd") === null,
getGeoCdFromChannelId(col("channel_id").toString())).otherwise(col("geo_cd")))
      .withColumn("geo_cd", when(col("geo_cd") === "",
getGeoCdFromChannelId(col("channel_id").toString())).otherwise(col("geo_cd")))

    // Date enrichment
    val enrichedData = cleanVideoPlayDf.filter("(user_id IS NOT NULL AND
user_id != 0) AND (video_id IS NOT NULL AND video_id != 0) AND (creator_id IS
NOT NULL AND creator_id != 0) AND (timestamp IS NOT NULL AND timestamp != '')
AND minutes_played IS NOT NULL AND (geo_cd IS NOT NULL AND geo_cd != '') AND
```

```
(channel_id IS NOT NULL AND channel_id != 0) AND video_end_type IS NOT NULL AND
liked IS NOT NULL AND disLiked IS NOT NULL")

    // Save Enriched data in HDFS
    new Properties()
    enrichedData.select("user_id", "video_id", "creator_id", "timestamp",
"minutes_played", "geo_cd", "channel_id", "video_end_type", "liked",
"disLiked")
      .write

.orc("hdfs://nameservice1/user/edureka_788309/mid-project2/datasets/enriched/")
    spark.close()
  }
}
```

4. Code for Real-time ingestion of Video Play data written in SparkStreaming-Scala

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{col, when}
import org.apache.spark.sql.types.{DataTypes, StructField, StructType}

object VideoStreamIngestion {
  def main(args: Array[String]): Unit = {

    // Read only today's data
    val currentDate = java.time.LocalDate.now
    val filePath =
"/mnt/bigdatapgp/edureka_788309/mid-project2/data_generator/out/generate_data/"
+currentDate+"/json"

    // Check File Existence
    if( !new java.io.File(filePath).exists) {
      println("Error: File Not exits for " + currentDate)
      return
    }

    // Define Json file Schema
    val jsonSchema = StructType(
        List(
          StructField("liked", DataTypes.BooleanType, nullable=true),
          StructField("user_id", DataTypes.LongType, nullable=true),
          StructField("video_end_type", DataTypes.StringType, nullable=true),
          StructField("minutes_played", DataTypes.LongType, nullable=true),
          StructField("video_id", DataTypes.LongType, nullable=true),
          StructField("geo_cd", DataTypes.StringType, nullable=true),
          StructField("channel_id", DataTypes.LongType, nullable=true),
          StructField("creator_id", DataTypes.LongType, nullable=true),
          StructField("timestamp", DataTypes.StringType, nullable=true),
          StructField("disliked", DataTypes.BooleanType, nullable=true)
        )
    )

    // Create spark session
    val spark = SparkSession.builder().appName("Stream Ingestion of Video Play
Website Data").master("local[2]").getOrCreate()

    // Load Json file
    val videoMobileDf =
spark.readStream.format("json").schema(jsonSchema).load("file://" + filePath)

    // Get Lookup table
    val channelGeoCdDf = spark.read.format("jdbc")
      .option("url", "jdbc:mysql://dbserver.edu.cloudlab.com")
      .option("dbtable", "labuser_database.channel_geocd_788309")
      .option("user", "edu_labuser")
```

```
      .option("password", "edureka")
      .load()

    def getGeoCdFromChannelId(channelId: String) =
channelGeoCdDf.filter(col("channel_id") ===
channelId).select(col("geo_cd")).collect().map(_.getString(0)).mkString("")

    // Clean the records
    val cleanVideoPlayDf = videoMobileDf
      .withColumn("liked", when(col("liked") === null,
false).otherwise(col("liked")))
      .withColumn("disliked", when(col("disliked") === null,
false).otherwise(col("disliked")))
      .withColumn("video_end_type", when(col("video_end_type") =!= 0 &&
col("video_end_type") =!= 1 && col("video_end_type") =!= 2 &&
col("video_end_type") =!= 3, 3).otherwise(col("video_end_type")))
      .withColumn("geo_cd", when(col("geo_cd") === null,
getGeoCdFromChannelId(col("channel_id").toString())).otherwise(col("geo_cd")))
      .withColumn("geo_cd", when(col("geo_cd") === "",
getGeoCdFromChannelId(col("channel_id").toString())).otherwise(col("geo_cd")))

    // Date enrichment
    val enrichedData = cleanVideoPlayDf
      .select(col("user_id").cast("long"), col("video_id").cast("long"),
col("creator_id").cast("long"), col("timestamp").cast("string"),
col("minutes_played").cast("long"), col("geo_cd"),
col("channel_id").cast("long"), col("video_end_type").cast("long"),
col("liked").cast("boolean"), col("disLiked").cast("boolean"))
      .filter("(user_id IS NOT NULL AND user_id != 0) AND (video_id IS NOT NULL
AND video_id != 0) AND (creator_id IS NOT NULL AND creator_id != 0) AND
(timestamp IS NOT NULL AND timestamp != '') AND minutes_played IS NOT NULL AND
(geo_cd IS NOT NULL AND geo_cd != '') AND (channel_id IS NOT NULL AND
channel_id != 0) AND video_end_type IS NOT NULL AND liked IS NOT NULL AND
disLiked IS NOT NULL")

    // Save Enriched Streamed data
    val data = enrichedData
      .writeStream
      .format("orc")
      .option("path",
"hdfs://nameservice1/user/edureka_788309/mid-project2/datasets/enriched/")
      .option("checkpointLocation",
"hdfs://nameservice1/user/edureka_788309/mid-project2/datasets/checkPointStream
/")
      .start()

    data.awaitTermination()
    spark.close() }}
```

5. Execute the Spark code from edge node

```
-- Command to execute the Batch Ingestion of Video Play data
spark2-submit --class VideoBatchIngestion --packages
com.databricks:spark-xml_2.11:0.5.0,mysql:mysql-connector-java:5.1.38
--driver-class-path Jars/mysql-connector-java-5.1.38.jar
Jars/videobatchingestion_2.11-0.1.jar

-- Command to execute the Real-time Streaming of Video Play data
spark2-submit --class VideoStreamIngestion --packages
mysql:mysql-connector-java:5.1.38 --driver-class-path
Jars/mysql-connector-java-5.1.38.jar --master local
Jars/videostreamingestion_2.11-0.1.jar
```

6. Create a Hive table for solving the problem statements.

```
-- Create Hive database
CREATE DATABASE IF NOT EXISTS videoOnDemand_788309;

USE videoOnDemand_788309;

-- Create Hive table for Video Play data enriched from spark code
CREATE EXTERNAL TABLE IF NOT EXISTS enriched_video_data
(
`user_id` BIGINT,
`video_id` BIGINT,
`creator_id` BIGINT,
`timestamp` String,
`minutes_played` BIGINT,
`geo_cd` String,
`channel_id` BIGINT,
`video_end_type` BIGINT,
`liked` Boolean,
`disLiked` Boolean
) STORED AS ORC
LOCATION '/user/edureka_788309/mid-project2/datasets/enriched/';

-- Copy Channel Geocd data from MySQL to Hive
sqoop import \
--connect jdbc:mysql://dbserver.edu.cloudlab.com:3306/labuser_database \
--username edu_labuser \
--password edureka \
--split-by channel_id \
--query 'select * FROM channel_geocd_788309 WHERE $CONDITIONS' \
--hcatalog-database videoOnDemand_788309 \
--hcatalog-table channel_geocd \
--create-hcatalog-table \
--hcatalog-storage-stanza "stored as orcfile"

-- Copy Video Creator data from MySQL to Hive
```

```
sqoop import \
--connect jdbc:mysql://dbserver.edu.cloudlab.com:3306/labuser_database \
--username edu_labuser \
--password edureka \
--split-by creator_id \
--query 'select * FROM video_creator_788309 WHERE $CONDITIONS' \
--hcatalog-database videoOnDemand_788309 \
--hcatalog-table video_creator \
--create-hcatalog-table \
--hcatalog-storage-stanza "stored as orcfile"
```

```
-- Copy User Subscription data from MySQL to Hive
sqoop import \
--connect jdbc:mysql://dbserver.edu.cloudlab.com:3306/labuser_database \
--username edu_labuser \
--password edureka \
--split-by user_id \
--query 'select * FROM user_subscn_788309 WHERE $CONDITIONS' \
--hcatalog-database videoOnDemand_788309 \
--hcatalog-table user_subscn \
--create-hcatalog-table \
--hcatalog-storage-stanza "stored as orcfile"
```

```
-- Create User Creator data from HDFS
CREATE TABLE IF NOT EXISTS user_creator
(
      user_id STRING,
      creators_array ARRAY<STRING>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '&';


LOAD DATA INPATH
'hdfs://nameservice1/user/edureka_788309/mid-project2/datasets/user-creator.csv
'
OVERWRITE INTO TABLE user_creator;
```

7. Problem Statements

```
USE videoOnDemand_788309;

-- Enable the dynamic partition
set hive.exec.dynamic.partition.mode=nonstrict;
```

1. Fetch the most popular channels by the criteria of a maximum number of videos played, also liked by unique users.
Output: It should include the columns – channel id, total distinct videos played, count of distinct users

```
CREATE TABLE IF NOT EXISTS popular_channels (
      channel_id STRING,
      total_distinct_videos_played INT,
      distinct_user_count INT
) PARTITIONED BY (geo_cd String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE popular_channels PARTITION(geo_cd)
SELECT
      channel_id,COUNT(DISTINCT user_id) AS distinct_user_count,COUNT(DISTINCT
video_id) AS total_distinct_videos_played,geo_cd
FROM
      enriched_video_data
WHERE
      liked=True
GROUP BY
      channel_id
ORDER BY
      total_distinct_videos_played DESC
LIMIT 10;
```

channel_id	distinct_user_count	total_distinct_videos_played
4051	588	588
1399	588	588
561	587	587
5842	587	587
3202	586	586
5104	586	586
5755	586	586
1311	585	585
3588	585	585

3196	585	585

2. Determine the total duration of videos played by each type of user, where the type of user can be 'subscribed' or 'unsubscribed.' An unsubscribed user is the one whose video is either not present in the lookup table created from the dataset - user-subscn.txt or has subscription_end_date earlier than the timestamp of the video played by him.
Output: It should include the columns – user type, duration

```
CREATE TABLE IF NOT EXISTS users_durations (
      user_type STRING,
      duration INT
)
PARTITIONED BY (geo_cd String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

INSERT OVERWRITE TABLE users_durations PARTITION(geo_cd)
SELECT
      CASE
            WHEN (su.user_id IS NULL OR evd.timestamp IS NULL OR
CAST(unix_timestamp(evd.timestamp,'dd/MM/yyyy hh:mm:ss') AS DECIMAL(20,0)) >
CAST(su.subscription_end_date AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED'
            WHEN (su.user_id IS NOT NULL AND evd.timestamp IS NOT NULL AND
CAST(unix_timestamp(evd.timestamp,'dd/MM/yyyy hh:mm:ss') AS DECIMAL(20,0)) <=
CAST(su.subscription_end_date AS DECIMAL(20,0))) THEN 'SUBSCRIBED'
      END AS user_type,
      sum(evd.minutes_played),
      geo_cd
FROM
      enriched_video_data evd
      LEFT OUTER JOIN user_subscn su ON evd.user_id = su.user_id
GROUP BY
      CASE
            WHEN (su.user_id IS NULL OR evd.timestamp IS NULL OR
CAST(unix_timestamp(evd.timestamp,'dd/MM/yyyy hh:mm:ss') AS DECIMAL(20,0)) >
CAST(su.subscription_end_date AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED'
            WHEN (su.user_id IS NOT NULL AND evd.timestamp IS NOT NULL AND
CAST(unix_timestamp(evd.timestamp,'dd/MM/yyyy hh:mm:ss') AS DECIMAL(20,0)) <=
CAST(su.subscription_end_date AS DECIMAL(20,0))) THEN 'SUBSCRIBED'
      END, geo_cd;
```

user_type	duration	geo_cd
UNSUBSCRIBED	160428443	AA
UNSUBSCRIBED	160295186	AP
UNSUBSCRIBED	160368403	AU

UNSUBSCRIBED	160669531	EU
UNSUBSCRIBED	160478672	JP

Note: Dataset seems to be very old or data is not correct

3. Determine the list of connected creators. Connected creators are those whose videos are most watched by the unique users who follow them.
Output: It should include the columns – creator id, count of user

```
CREATE TABLE IF NOT EXISTS connected_creators (
      creator _id STRING,
      user_count INT
) PARTITIONED BY(geo_cd String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE connected_creators PARTITION(geo_cd)
SELECT
      ua.creator_id,COUNT(DISTINCT ua.user_id) AS user_count,geo_cd
FROM
(
      SELECT
            user_id, creator_id
      FROM
            user_creator LATERAL VIEW explode(creators_array) creators AS
creator_id
) ua
INNER JOIN
(
      SELECT
            creator_id, video_id, user_id,geo_cd
      FROM
            enriched_video_data
) ed
ON ua.creator_id = ed.creator_id AND ua.user_id = ed.user_id
GROUP BY
      ua.creator_id,ed.geo_cd
ORDER BY
      user_count DESC
LIMIT 10;
```

4. Determine which videos and creators are generating maximum revenue. Royalty applies to a video only if it was liked, completed successfully, or both.
Output: It should include the columns – video id, duration

```
CREATE TABLE IF NOT EXISTS top_royalty_videos (
      video_id STRING,
      duration INT)
      PARTITIONED BY (geo_cd String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE top_royalty_videos PARTITION(geo_cd)
SELECT
      video_id,SUM(minutes_played) AS duration,geo_cd
FROM
      enriched_video_data
WHERE
      liked = true OR video_end_type=0
GROUP BY
      video_id,geo_cd
ORDER BY
      duration DESC
LIMIT 10;
```

video_id	duration	geo_cd
19199	8089	AA
2818	7736	AA
6024	8621	AP
446	8298	AP
41785	7903	AP
25912	7730	AU
43885	7956	EU
1231	7842	EU
20023	7748	EU
5629	8062	JP

5. Determine the unsubscribed users who watched the videos for the longest duration.
Output: It should include the columns – user id, duration

```
CREATE TABLE IF NOT EXISTS top_unsubscribed_users (
      user_id STRING,
      duration INT
) PARTITIONED BY(geo_cd String)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

INSERT OVERWRITE TABLE top_unsubscribed_users PARTITION(geo_cd)
SELECT
      ed.user_id, SUM(ed.minutes_played) AS duration,ed.geo_cd
FROM
      enriched_video_data ed
      LEFT OUTER JOIN user_subscn su ON ed.user_id=su.user_id
WHERE
      (su.user_id IS NULL OR (CAST(unix_timestamp(ed.timestamp, 'dd/MM/yyyy
hh:mm:ss') AS DECIMAL(20,0)) > CAST(su.subscription_end_date AS
DECIMAL(20,0))))
GROUP BY
      ed.user_id,ed.geo_cd
ORDER BY
    duration DESC
LIMIT 10;
```

user_id	duration	geo_cd
6024	8621	AP
446	8298	AP
19199	8089	AA
5629	8062	JP
43885	7956	EU
41785	7903	AP
1231	7842	EU
20023	7748	EU
2818	7736	AA
25912	7730	AU

----------xxxxxxxxxx----------