SPARK -TWITTER PIPELINE

BY : IBRAHIM ELSAYED ELSADEK

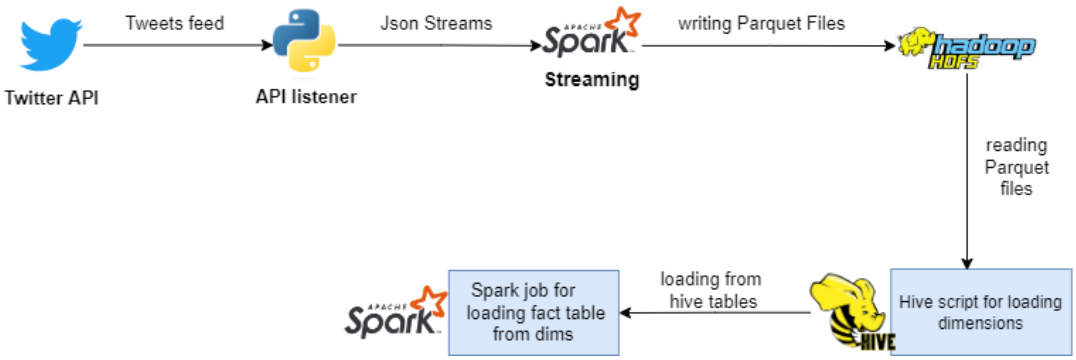
ITI intake 43

Introduction:

Apache Spark is a popular open-source big data processing framework that provides high-performance, scalable, and fault-tolerant processing of large data sets. Spark Streaming is a real-time processing extension to Spark that enables the processing of live data streams. These technologies have become the backbone of many modern big data applications, including real-time analytics, machine learning, and data processing.

In this report, we present a project that utilizes Spark and Spark Streaming to process data from the Twitter API. The goal of this project is to extract meaningful insights and analyze real-time data streams from Twitter. The project uses Spark's distributed processing capabilities and Spark Streaming's real-time processing capabilities to analyze large volumes of tweets in real-time.

The report provides an overview of the project's architecture, design, and implementation details.



Business case: "Layoffs"

Over the past few months, the global job market has been hit hard by a wave of layoffs across companies of all sizes. At our organization, we recognized the importance of exploring this topic in-depth and understanding how individuals are responding to these workforce changes.

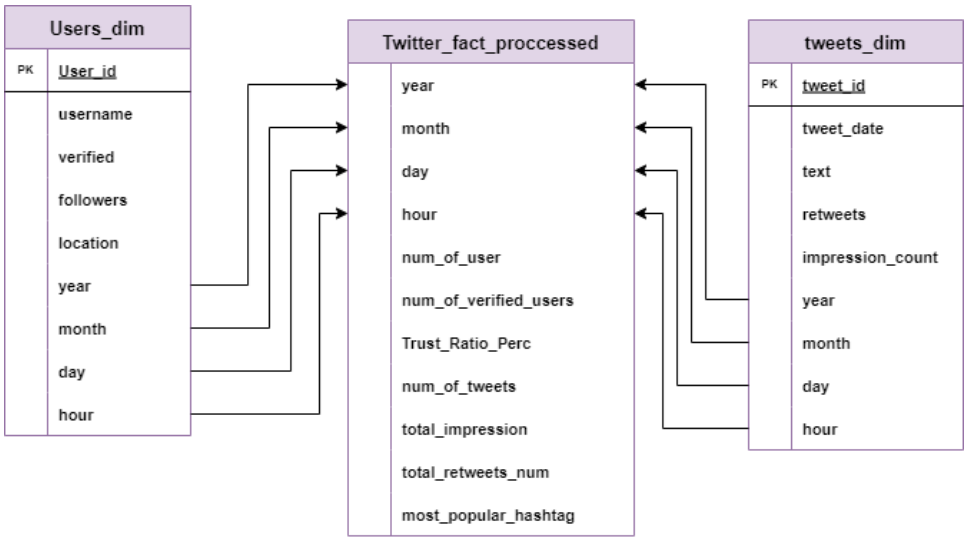
To gather insights and analyze trends, we developed a custom script that leverages Twitter's Search API to collect real-time tweets related to the keyword "layoffs". We carefully crafted our search criteria to include a range of variations on the term, such as "laid off," "job loss," and "unemployment," in order to capture a comprehensive view of the conversation.

Our ultimate goal with this project is to build a robust data warehouse that will store all of the information we collect from Twitter. Through careful data transformation and analysis, we aim to create a real-time dashboard that visualizes key performance indicators and metrics related to this topic. By doing so, we can gain valuable insights into how people are engaging with the issue of layoffs and understand the broader social and economic implications of this trend.

Point of time:

For Pipeline: As we decided to analyze tweets after 24 hours from tweet creation data so, extraction time will be **(day -1)**

Data warehouse schema:



Fact table description:

Grain: hour per day per month per year

Measures:

- ***Num_of_users:*** count distinct number of users who tweet tweets per hour
- ***Num_of_verified_users***: count distinct number of verified users who tweet tweets per hour
- ***Trust_ratio_perc***: to measure percentage of verified users to all users who post tweets in a specific hour to know how much keywords are followed by trusted users. By dividing (Num_of_verified_users / Num_of_users) we got this measure
- ***Num_of_tweets***: count number of tweets per hour
- ***Total_impression***: it's sum of all tweets impressions per hour
- ***Total_retweets_num***: it's sum of all tweets retweets per hour
- ***Most_popular_hashtag***: to know what is the most repeated hashtag per hour

Code details:

1- Twitter Listener: `twitter_listener.py`

- **Modifications on code:**
  - Script extract data every 5 minutes
  - Start and End data are (day-1 with period 5 minutes between start and end)
  - Data is being sent to spark streaming as a JSON file for every batch.
  - Script is running all time the machine is running
- **Source :** twitter API
- **Target :** socket stream to spark streaming

2- Spark streaming script: `spark_streaming.py`

- **Code specs:**
  - User defined schema that applied on the received JSON file
  - Data received is being written on HDFS as Parquet file with partitioned by (year , month , day , hour)
  - Script is running all time the machine is running
- **Source** : socket stream from listener
- **Target :** HDFS: /twitter-landing-data

3- Hive dimensions Script: `hive_script.sql`

- **Code specs:**
  - Create statement for three tables (twitter_landing_table , users_raw , tweets_raw)
  - *Slowly Changing Dimension* in Users_raw Table
  - To achieve a Slowly Changing Dimension (SCD) in the users_raw table, a work-around approach was adopted. The concept involves creating a temporary table to hold the data from the already existing users_raw table and new users from the twitter_landing_table who are not in users_raw.
  - The temporary table serves as a merge table where the new data is added and the existing data is updated based on the user_id column. Once the merge table is updated, all data in the users_raw table is replaced with the data from the merge table.
- **Source for twitter_landing_table :** HDFS: /twitter-landing-data
- **Source for dimensions :** table(twitter_landing_table)
- **Target :** 2 Tables (tweets_raw , users_raw) located at HDFS: /twitter-raw-data

tweet_id	tweet_date	text	retweets	impression_count	username	user_id	verified	followers	location	year	month	day	hour
1654600078353649665	2023-05-05T21:32:...	RT @JBswg_Diary: ...	5400	0	thnnn074397351	1645238148300169216	false	50	unknown	2023	5	5	21
1654598842619396097	2023-05-05T21:27:...	@HillaryClinton W...	0	126	Rogue_slate	1328440184971509760	false	99	LakeShow	2023	5	5	21
1654601061057830918	2023-05-05T21:36:...	@BenPiggot @JHWei...	0	543	BLJ19641	1335030630221025285	false	67	Massachusetts, USA	2023	5	5	21
1654600448618414081	2023-05-05T21:34:...	Plaion reportedly...	0	34	channel969	1461484119121031169	false	297	unknown	2023	5	5	21
1654599239006298112	2023-05-05T21:29:...	Want to land a jo...	0	8	tmj_OHD_secure	120481130	false	192	Dayton, OH	2023	5	5	21
1654601041084657664	2023-05-05T21:36:...	Are any banks out...	0	13	SwampPump	861729577818664961	false	1035	Leading The Way	2023	5	5	21

Sample of twitter_landing_table output

user_id	username	verified	followers	location	year	month	day	hour
1272235459343790080	yanlisgff	false	407	black | she/they	2023	5	5	21
1482319724377788417	De4nWhite	false	3	unknown	2023	5	5	21
768909227599409152	Keith_M_Preston	false	4252	New York	2023	5	5	21
1594153521661763584	Reality77717	false	331	In a land far, fa...	2023	5	5	21
4269803603	vmsantospedro1	false	479	Truck-free Capita...	2023	5	5	21
1238521362932617216	MythrilSaber7	false	151	unknown	2023	5	5	21

Sample of users_raw dimension output

tweet_id	user_id	tweet_date	text	retweets	impression_count	year	month	day	hour
1654600078353649665	1645238148300169216	2023-05-05T21:32:...	RT @JBswg_Diary: ...	5400	0	2023	5	5	21
1654599291586101248	1649138865113030657	2023-05-05T21:29:...	RT @JBswg_Diary: ...	5399	0	2023	5	5	21
1654598842619396097	1328440184971509760	2023-05-05T21:27:...	@HillaryClinton W...	0	126	2023	5	5	21
1654599556431060994	14669951	2023-05-05T21:30:...	Content moderator...	0	5769	2023	5	5	21
1654600448618414081	1461484119121031169	2023-05-05T21:34:...	Plaion reportedly...	0	34	2023	5	5	21

Sample of tweets_raw dimension output

4- SparkSQL Fact Table Script: `fact_processing.py`

- **Code specs:**
  - Extracting data from dims using SparkSQL with hive metastore
  - New attribute (Trust_Ratio_Perc) has been generated on the fly using some SQL
  - Hashtags was extracted alone with tweet_id to make it as dim on the fly to know popular hashtags
  - Hashtags like: layoff, #layoffs #job #loss has been excluded from results because it refers to the keyword that we already searched for.
- **Source:** dimensions Tables (tweets_raw , users_raw)
- **Target** : Tables (twitter_fact_processed) located at HDFS: /twitter-processed-data

	year	month	day	hour	num_of_user	num_of_verified_users	Trust_Ratio_Perc	num_of_tweets	total_impression	total_retweets_num	most_popular_hashtag
0	2023	5	5	21	72	3	4.0	73	13067	60042	#tamojuntopodcast

Sample of twitter_fact_processed table output

5- Pipeline code:

- **We have 4 scripts, twitter_listener.py and spark_streaming.py are running along time, and hive_script.sql, fact_processing.py run every 15 minutes.**

  ** using these commands two jobs will run all time

```
python3 twitter_listener.py
```

```
nohup spark-submit
    --master yarn
    --deploy-mode cluster
    --conf spark.pyspark.python=/usr/bin/python3 spark_streaming.py
    > spark.out &
```

- **Using crontab to schedule these jobs to run every five minutes as following:**

  Firstly we need to create .sh script to add it in crontab service which include spark-submit command with configs of spark app as required

```
#!/bin/bash

export SPARK_HOME=/opt/spark2
export HADOOP_CONF_DIR=/opt/hadoop

$SPARK_HOME/bin/spark-sql \
    --conf spark.sql.shuffle.partitions=4 \
    --queue streaming \
    -f /home/itversity/hive_script.sql
```

```
#!/bin/bash

export SPARK_HOME=/opt/spark2
export HADOOP_CONF_DIR=/opt/hadoop

$SPARK_HOME/bin/spark-submit \
    --master yarn \
    --deploy-mode client \
    --queue streaming \
    --conf spark.pyspark.python=/usr/bin/python3 \
     /home/itversity/fact_processing.py
```

- **Then in crontab we added this line, so this line ensure that scripts run every 10 minutes and fact job run after hive job finish its work**

```
*/15 * * * * /home/itversity/hive_job.sh && sleep 30 && /home/itversity/fact_job_submit.sh
```