ELEN 6889
Large Scale Data Stream Processing

Final Project Presentation
Topic: Data Analysis in Sport Events Streaming

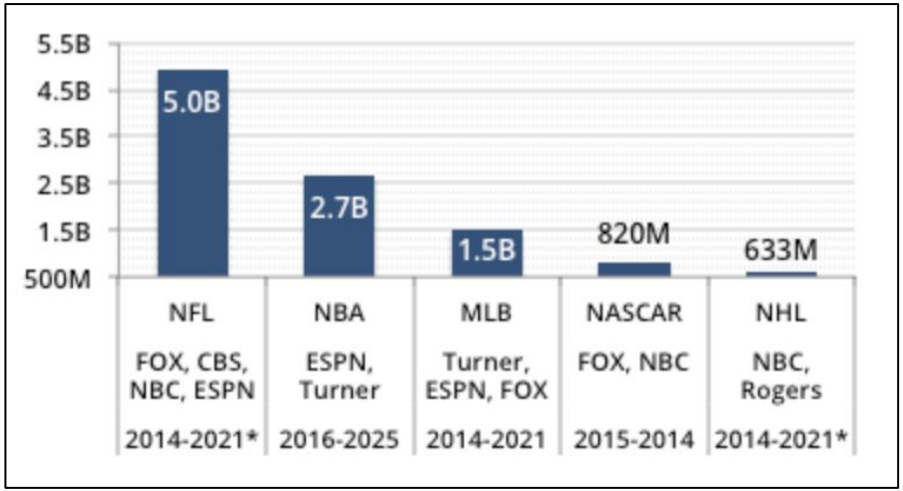
Group members: Weirui Peng, Brian Yao, Jingtian Zhang, Yifeng Wang

Date: May 2, 2023

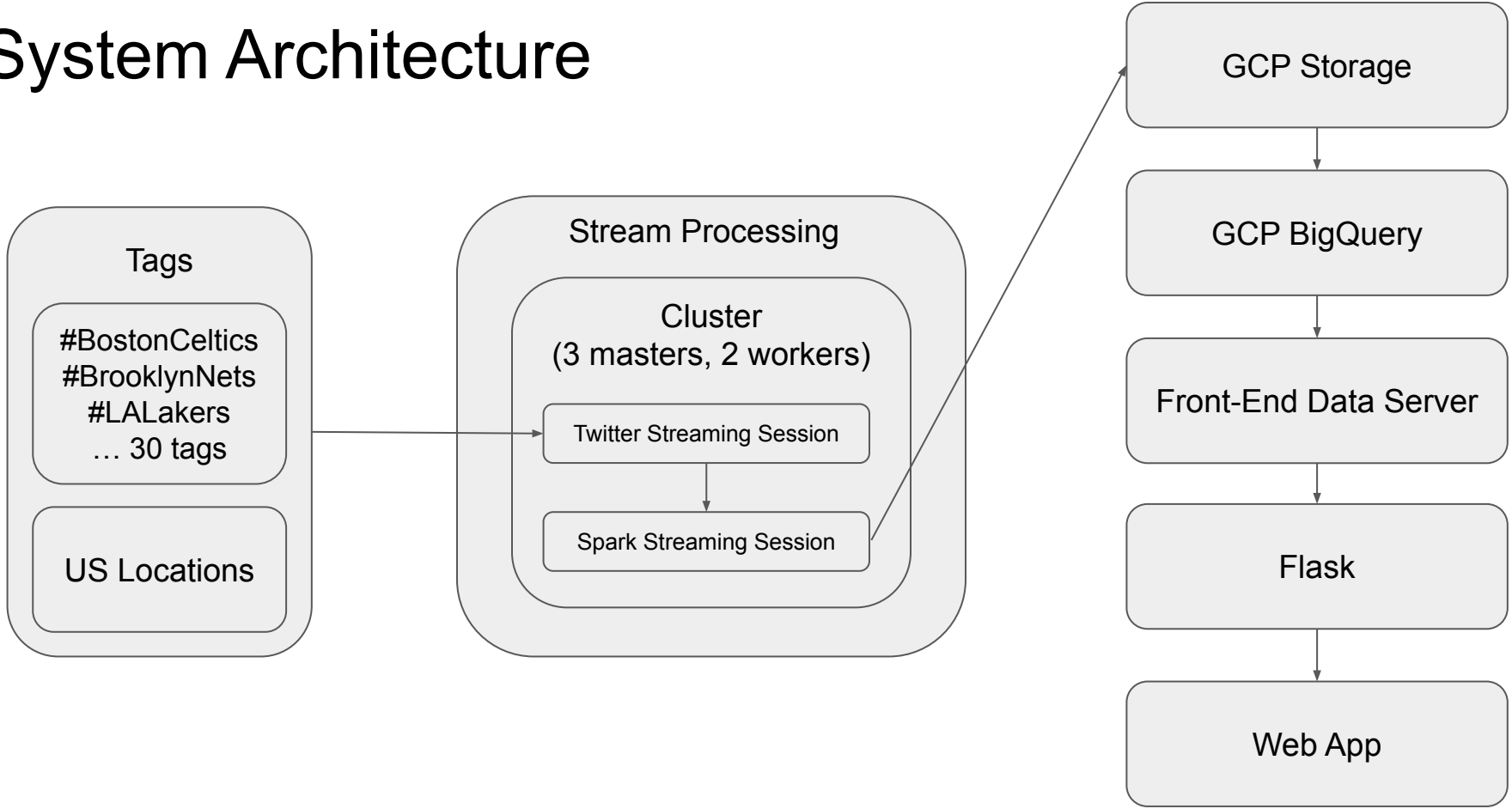
Contents

Introduction

- Sport leagues rely on viewership and ratings for their respective TV deals
- Social media produces data constantly and allows us to look in the lens of a fan to understand how events are perceived
- Real-time streaming allows for real-time strategy adjustment



System Architecture



Data Collection

- Set up necessary API keys and access tokens to connect to the Twitter API through Tweepy
- Define search parameters: hashtags, date & time windows, and quantity of tweets to be pulled per hashtag
- Prepare NLTK (Natural Language Toolkit) to be used for sentiment analysis
- For each hashtag, create a CSV file to save the tweet data

Data Collection

- Parse tweets for query matches. For each tweet:
  a. Retrieve creation time, text, location (if available), and favorite count
  b. Analyze sentiment using the NLTK (Natural Language Toolkit)
  c. Save hashtag, time, text, location, favorite count, and sentiment in a CSV file

- The resulting dataset consists of several CSV files, each containing the tweet data for a single hashtag, including the hashtag, time, text, location, favorite count, and sentiment of each tweet

Data Collection

- Difficulties

  a. Twitter API level limit

  - Apply an ELEVATED level twitter developer access
  - Access data in recent 7 days
  - So only collect data during 4/13/2023 - 4/21/2023

  b. Twitter API tweet limit

  - At most 15,000 tweets within 15 minutes
  - So the program will automatically wait for 15 minutes when facing “429 error”

Data Collection

```python
auth = tweepy.OAuth1UserHandler(api_key, api_secret, access_token, access_secret)
api = tweepy.API(auth)

until_date = '2023-04-22'
until_time = '23:59:59'

max_tweets = 200
query_date_time = datetime.strptime(until_date + ' ' + until_time, '%Y-%m-%d %H:%M:%S')
query_date_time_formatted = query_date_time.strftime('%Y-%m-%d_%H:%M:%S')

query_hashtags = []
# 'ClevelandCavaliers', 'NewYorkKnicks', 'Philadelphia76ers', 'BostonCeltics', 'AtlantaHawks','MilwaukeeBucks', 'MiamiHeat', '
# '76ers', 'Nets', 'Celtics','Bucks', 'Cavaliers', 'Knicks', 'Heat', 'Hawks'
# 'Nuggets', 'Grizzles', 'Kings', 'Suns', 'Clippers','Warriors', 'Lakers', 'Timberwolves'
# 'PhoenixSuns', 'DenverNuggets', 'MemphisGrizzlies', 'SacramentoKings', 'LAClippers', 'GoldenStateWarriors', 'LosAngelesLaker
```

```python
for hashtag in query_hashtags:
    query = '#' + hashtag + ' until:' + query_date_time_formatted
    path = hashtag + '.csv'
    with open(path, 'a', encoding='utf-8', newline='') as f:
        csv_writer = csv.writer(f)
```

```python
    try:
        print('Now print: ' + hashtag)
        for tweet in tweepy.Cursor(api.search_tweets, q=query, count=max_tweets).items():
            time = tweet.created_at.strftime('%Y-%m-%d %H:%M:%S')
            text = tweet.text
            if tweet.place is not None:
                location = tweet.place.full_name + ', ' + tweet.place.country
            else:
                location = 'No location available.'
            favorite_count = tweet.favorite_count

            # 分析推文的情感
            sentiment_scores = sia.polarity_scores(text)
            sentiment = 0  # 中性情感
            if sentiment_scores['pos'] > sentiment_scores['neg']:
                sentiment = 1  # 积极情感
            elif sentiment_scores['pos'] < sentiment_scores['neg']:
                sentiment = -1  # 消极情感

            # 将推文数据作为一行写入 CSV 文件
            csv_writer.writerow([hashtag, time, text, location, favorite_count, sentiment])
            sum += 1

        print(sum)
    except Exception as e:
        print(e)
        if '429' in str(e):
            print('Rate limit exceeded. Waiting for 15 minutes...')
            t.sleep(15 * 60)  # 等待 15 分钟
            continue
```

Analysis Process

- Read the streaming data from a Google Cloud Storage bucket using spark.readStream.csv(), returning a DataFrame that represents the data stream

- Filter the streaming data based on non-null values for location, likes, and sentiment, converting -1 values to 0 for column ‘likes’, as well as filtering out foreign language text; Use Watermark function on the filtered data helps the system handle late data by defining a 1-hour threshold

```
filtered_data = (
    streaming_data.filter(
        "isnotnull(location) AND isnotnull(likes) AND isnotnull(sentiment)")
    .withColumn("likes", when(col("likes") == -1, 0).otherwise(col("likes")))
    .filter(is_english_udf(col("text")))
    .withWatermark("timestamp", "1 hour")
)
```

Analysis Process

- Perform windowed aggregation on the filtered data using the groupBy() and agg() methods.
  - The data is grouped by a 30-minute window and slide duration, as well as the location
  - This allows for real-time analysis of the data within each window
  - Select the necessary columns and rename the window start and end columns to provide a more comprehensive output

```
# Aggregate data
aggregated_data = (
    filtered_data.groupBy(
        window("timestamp", "30 minutes", "30 minutes"),
        "location"
    )
    .agg(
        sum("likes").alias("total_likes"),
        sum(when(col("sentiment") == 1, 1).otherwise(0)).alias("count_sentiment_1"),
        sum(when(col("sentiment") == 0, 1).otherwise(0)).alias("count_sentiment_0"),
        sum(when(col("sentiment") == -1, 1).otherwise(0)).alias("count_sentiment_-1"),
        avg("sentiment").alias("average_sentiment")
    )
    .select(
        "location",
        "total_likes",
        "count_sentiment_1",
        "count_sentiment_0",
        "count_sentiment_-1",
        "average_sentiment",
        col("window").getField("start").alias("window_start"),
        col("window").getField("end").alias("window_end")
    )
)
```

Analysis Process

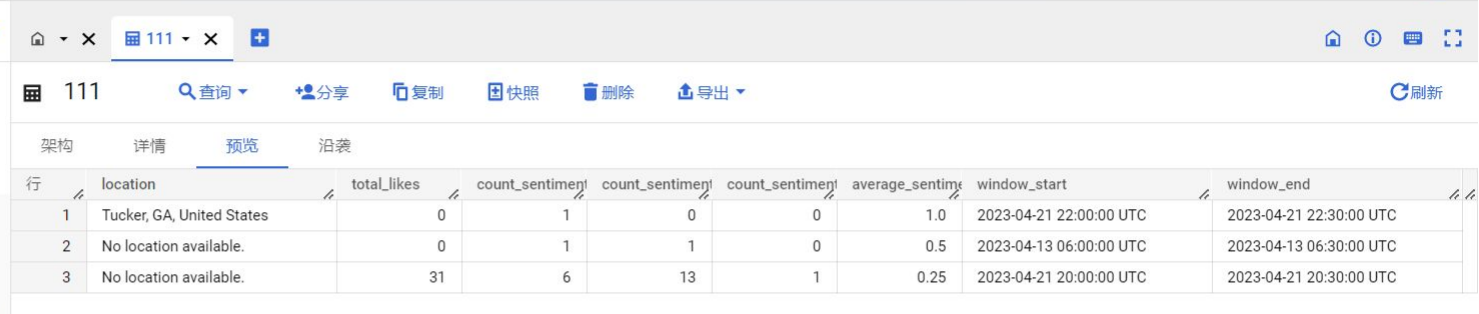
- The aggregated data is written to a checkpoint folder and new data will be appended to the existing output data

- The final output data is read into a DataFrame and sorted based on the window_start column in ascending order

```
# Write aggregated data to temporary folder
query = (
    aggregated_data.writeStream
    .outputMode("append")
    .format("parquet")
    .option("path", "gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/")
    .option("checkpointLocation", "gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/checkpoint/")
    .start()
)


query.awaitTermination()

# Read the output from the temporary folder
output_data = spark.read.parquet("gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/")

# Sort the output data and write the final result
sorted_output = output_data.sort(asc("window_start"))

sorted_output.write.csv("gs://dataproc-staging-us-central1-300388429198-ocbfs3ky/outputdata/", mode="overwrite")
```

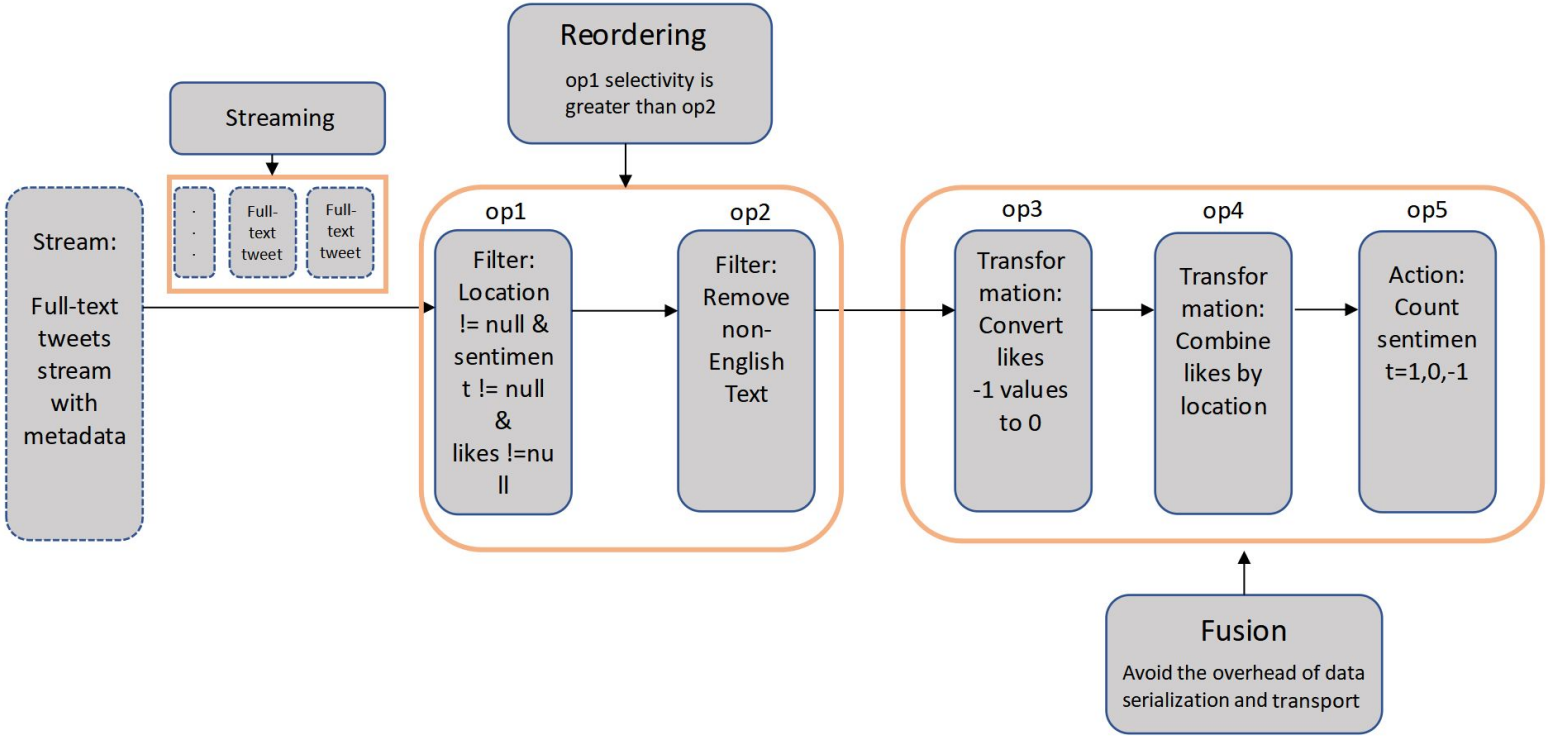
Analysis Process

- The sorted output data is written as CSV files to a Google Cloud Storage bucket, overwriting any existing data



行	location	total_likes	count_sentiment	count_sentiment	count_sentiment	average_sentime	window_start	window_end
1	Tucker, GA, United States	0	1	0	0	1.0	2023-04-21 22:00:00 UTC	2023-04-21 22:30:00 UTC
2	No location available.	0	1	1	0	0.5	2023-04-13 06:00:00 UTC	2023-04-13 06:30:00 UTC
3	No location available.	31	6	13	1	0.25	2023-04-21 20:00:00 UTC	2023-04-21 20:30:00 UTC

Optimization



Results & Demo

- Main results:
  a. Visualization
  b. Query powered by BigQuery

- Our website is deployed on GCP:
  http://34.66.247.178:8111/

Results & Demo

- Our plot function is extendable by adding new queries

```python
def plot_bigquery(start_time, end_time):
    # create BigQuery client object
    client = bigquery.Client()

    # define SQL query to retrieve the required data
    query = f"""
            SELECT string_field_0 AS location,
                   int64_field_1 AS count_likes,
                   int64_field_2 AS count_sentiment1,
                   int64_field_3 AS count_sentiment0,
                   int64_field_4 AS count_sentiment_1,
                   double_field_5 AS avg_sentiment,
                   timestamp_field_6 AS start_time,
                   timestamp_field_7 AS end_time
            FROM `NBAdataset.result`
            WHERE timestamp_field_6 >= TIMESTAMP("{start_time.strftime('%Y-%m-%d %H:%M:%S')} UTC") AND timestamp_field_6 <= TIMESTAMP("{end_time.strftime('%Y-%m-%d %H:%M:%S')}
            """

    # execute the query and convert the result to a pandas DataFrame
    df = client.query(query).to_dataframe()

    path1 = plot_1(df)
    path2 = plot_2(df)
    path3 = plot_3(df)

    return path1, path2, path3
```

Thank you for your attention