NACKADEMIN

Datafångst, migrering och förädling (ETL/ELT)

NYC Taxi Trip ETL Pipeline with Apache Spark and MongoDB

Nora Ayaz

December 7, 2024

Contents

1 Introduction

This report outlines the implementation of a classic ETL (Extract, Transform, Load) pipeline for processing NYC taxi trip data. The pipeline leverages Apache Spark running locally with 4 workers for data processing, and MongoDB as the final data store.

The goals of this project include:

- Automating the pipeline to process new data without manual intervention.
- Demonstrating the functionality of the ETL pipeline with evidence of all workers contributing to the tasks.
- Scheduling the pipeline using `cron` for periodic execution.

2 Environment Setup

2.1 2.1 Apache Spark Configuration

Apache Spark 3.5.3 was installed and configured on a local machine. A master node and 4 worker nodes were launched to process the data in parallel. Each worker node was configured with 2 cores and 4GB of memory to optimize resource allocation. Below are the steps followed for the setup:

- **Starting the Spark Master Node:**

```
./sbin/start-master.sh
```

- **Starting Worker Nodes with Separate Working Directories:**

```
./sbin/start-worker.sh -d /usr/local/spark/work1 spark://localhost
   :7077
./sbin/start-worker.sh -d /usr/local/spark/work2 spark://localhost
   :7077
./sbin/start-worker.sh -d /usr/local/spark/work3 spark://localhost
   :7077
./sbin/start-worker.sh -d /usr/local/spark/work4 spark://localhost
   :7077
```

- **Specifying Resource Allocations:**

```
./sbin/start-worker.sh -c 2 -m 4G -d /usr/local/spark/work1 spark://
   localhost:7077
./sbin/start-worker.sh -c 2 -m 4G -d /usr/local/spark/work2 spark://
   localhost:7077
./sbin/start-worker.sh -c 2 -m 4G -d /usr/local/spark/work3 spark://
   localhost:7077
./sbin/start-worker.sh -c 2 -m 4G -d /usr/local/spark/work4 spark://
   localhost:7077
```

- **Verifying Active Nodes:**

```
jps
```

Below are the screenshots of the setup:

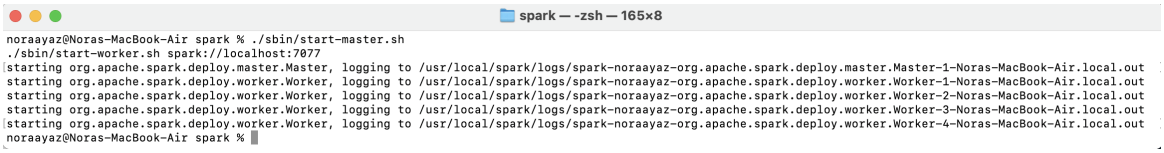

Figure 1: Spark master and worker nodes setup.

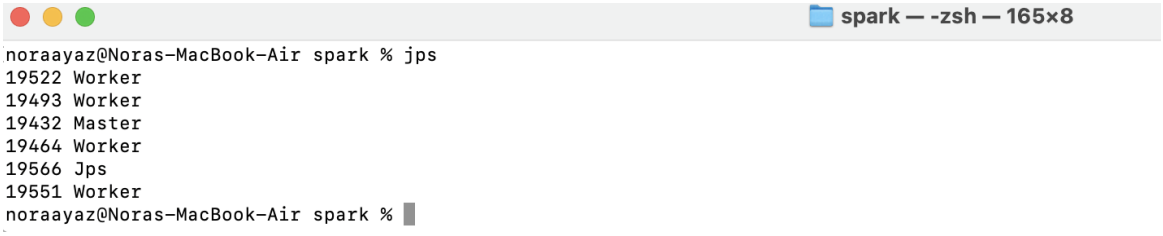

Figure 2: Verifying active Spark master and worker nodes using JPS.

2.2 2.2 MongoDB Configuration

MongoDB 6.0 was installed on the local machine using Homebrew. The installation and configuration were completed with the following steps:

- **Install MongoDB:**

```
brew tap mongodb/brew
brew install mongodb-community@6.0
```

- **Start MongoDB Service:**

```
brew services start mongodb/brew/mongodb-community
```

- **Access MongoDB Shell:**

```
mongosh
```

After the installation, the database was configured to store the processed trip data, and the configuration was validated as explained in the "Validation and Results" section.

3 ETL Pipeline Process

3.1 Pipeline Integration in Spark

The ETL pipeline was integrated into Spark, which required initializing a Spark session. The session was configured to optimize resources, including memory allocation and the number of cores per worker. The configuration details are as follows:

- **Master URL:** `spark://localhost:7077`
- **Executor Memory:** 2GB per worker.
- **Executor Cores:** 2 cores per worker.
- **Maximum Cores:** 8 cores across all workers.
- **Shuffle Partitions:** 16 partitions for efficient data shuffling.

Below is a screenshot of the Python code initializing the Spark session:

```
# Initialize Spark session
spark = SparkSession.builder \
    .master("spark://localhost:7077") \
    .appName("Yellow Taxi ETL Pipeline") \
    .config("spark.executor.memory", "2g") \
    .config("spark.executor.cores", "2") \
    .config("spark.cores.max", "8") \
    .config("spark.sql.shuffle.partitions", "16") \
    .getOrCreate()
```

Figure 3: Initialization of the Spark session for the ETL pipeline.

3.2 Extract - Data Retrieval

The data extraction process involves dynamically downloading files from the source URL. We used the following logic to download and manage files:

- The base URL for the Yellow Taxi dataset: `https://d37ci6vzurychx.cloudfront.net/trip-data/`.
- File patterns were defined as `yellow_tripdata_{year}-{month:02d}.parquet`.
- Metadata files `downloaded_files.json` and `processed_files.json` were used to track the status of downloads and processing.

```
# Base URL and file patterns
base_url = "https://d37ci6vzurychx.cloudfront.net/trip-data/"
file_pattern = "yellow_tripdata_{year}-{month:02d}.parquet"

# Directories for files
local_folder = "/usr/local/spark/data/yellow_taxi/"
processed_folder = "/usr/local/spark/data/yellow_taxi_fixed/"
metadata_downloaded = os.path.join(local_folder, "downloaded_files.json")
metadata_processed = os.path.join(processed_folder, "processed_files.json")

# Create directories if they don't exist
os.makedirs(local_folder, exist_ok=True)
os.makedirs(processed_folder, exist_ok=True)
```

Figure 4: Configuration of directories and file patterns for data extraction.

```python
def download_new_files():
    """
    Download Yellow Taxi data files that are not already downloaded.
    """
    downloaded_files = load_metadata(metadata_downloaded)

    for year in years:
        for month in range(1, 13):
            file_name = file_pattern.format(year=year, month=month)
            file_url = base_url + file_name
            local_file_path = os.path.join(local_folder, file_name)

            if file_name in downloaded_files or os.path.exists(local_file_path):
                print(f"File already exists, skipping: {file_name}")
                continue

            try:
                print(f"Downloading {file_url}...")
                response = requests.get(file_url)
                response.raise_for_status()
                with open(local_file_path, "wb") as f:
                    f.write(response.content)
                print(f"Successfully downloaded: {file_name}")
                downloaded_files.append(file_name)
                save_metadata(metadata_downloaded, downloaded_files)
            except requests.exceptions.RequestException as e:
                print(f"Failed to download {file_name}: {e}")
```

Figure 5: Python function for downloading new data files.

3.3 Transform - Data Cleaning and Feature Engineering

```python
    spark_df = spark_df.filter((col("trip_distance") > 0) & (col("total_amount") > 0))
    spark_df = spark_df.filter(unix_timestamp("tpep_dropoff_datetime") > unix_timestamp("tpep_pickup_datetime"))
    spark_df = spark_df.select(
        "VendorID", "tpep_pickup_datetime", "tpep_dropoff_datetime",
        "trip_distance", "PULocationID", "DOLocationID", "tip_amount", "total_amount"
    ).withColumn(
        "trip_duration_minutes",
        (unix_timestamp("tpep_dropoff_datetime") - unix_timestamp("tpep_pickup_datetime")) / 60
    ).withColumn(
        "tip_ratio", col("tip_amount") / col("total_amount")
    ).withColumn(
        "distance_segment",
        when(col("trip_distance") <= 2, "short")
        .when(col("trip_distance") <= 10, "medium")
        .otherwise("long")
    )

    transform_time = time() - start_time
    print(f"Transformations completed in {transform_time:.2f} seconds.")

    # Save transformed data to MongoDB
    start_time = time()
    save_to_mongodb(spark_df)

    mongo_time = time() - start_time
    print(f"Saved to MongoDB in {mongo_time:.2f} seconds.")

    # Mark file as processed
    processed_files.append(file_name)
    save_metadata(metadata_processed, processed_files)
except Exception as e:
    print(f"Error processing file {file_name}: {e}")
```

Figure 6: Data transformation and feature engineering in PySpark.

3.4 Load - Saving Processed Data to MongoDB

The processed data was saved into a MongoDB collection named `processed_trips`. The following Python function was used for saving data:

```python
# Step 3: Save data to MongoDB
def save_to_mongodb(df):
    """
    Save the processed DataFrame to MongoDB.
    """
    try:
        client = MongoClient("mongodb://localhost:27017/")
        db = client["yellow_taxi_db"]
        collection = db["processed_trips"]

        pandas_df = df.toPandas()
        collection.insert_many(pandas_df.to_dict("records"))
        print("Data successfully loaded into MongoDB!")
    except Exception as e:
        print(f"Error during MongoDB insertion: {e}")
```

Figure 7: Python function for saving transformed data to MongoDB.

3.5 File Processing and Metadata Tracking

To ensure idempotency and track the status of files, metadata management was integrated. The following logic was used to process files and maintain metadata:

```python
def process_files():
    """
    Process downloaded files and save transformed data to MongoDB.
    """
    downloaded_files = load_metadata(metadata_downloaded)
    processed_files = load_metadata(metadata_processed)
    to_process_files = [file for file in downloaded_files if file not in processed_files]

    if not to_process_files:
        print("No new files to process.")
        return

    print(f"Processing files: {to_process_files}")
```

Figure 8: File processing logic with metadata tracking.

3.6 Spark Execution in Stages

The pipeline execution was monitored in the Spark UI, showcasing how the data was processed across multiple stages. Below are the screenshots:

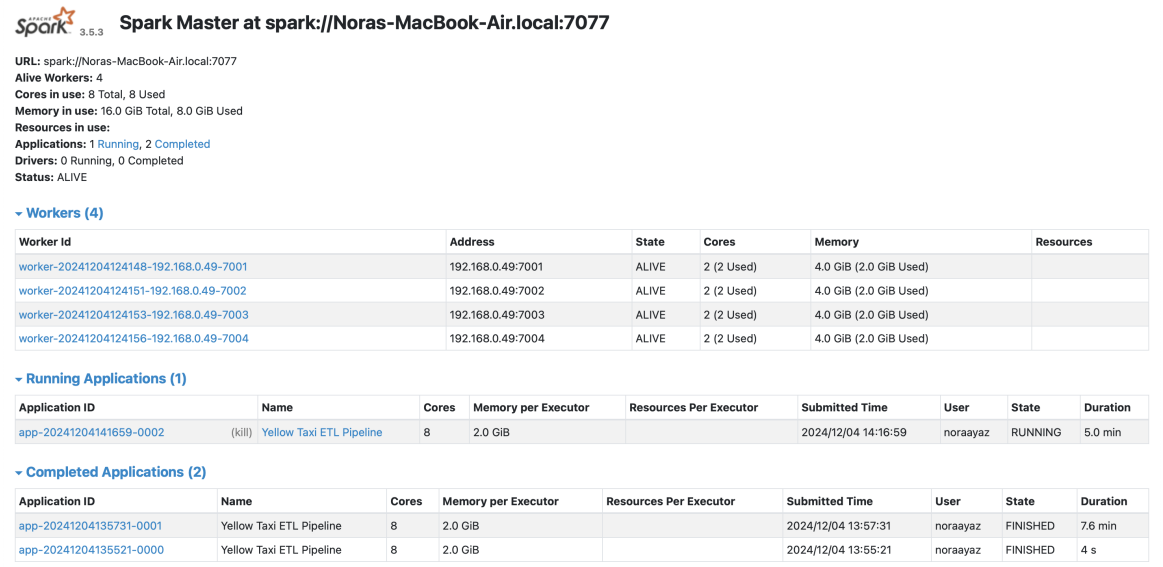

Figure 9: Overview of Spark application in the UI.

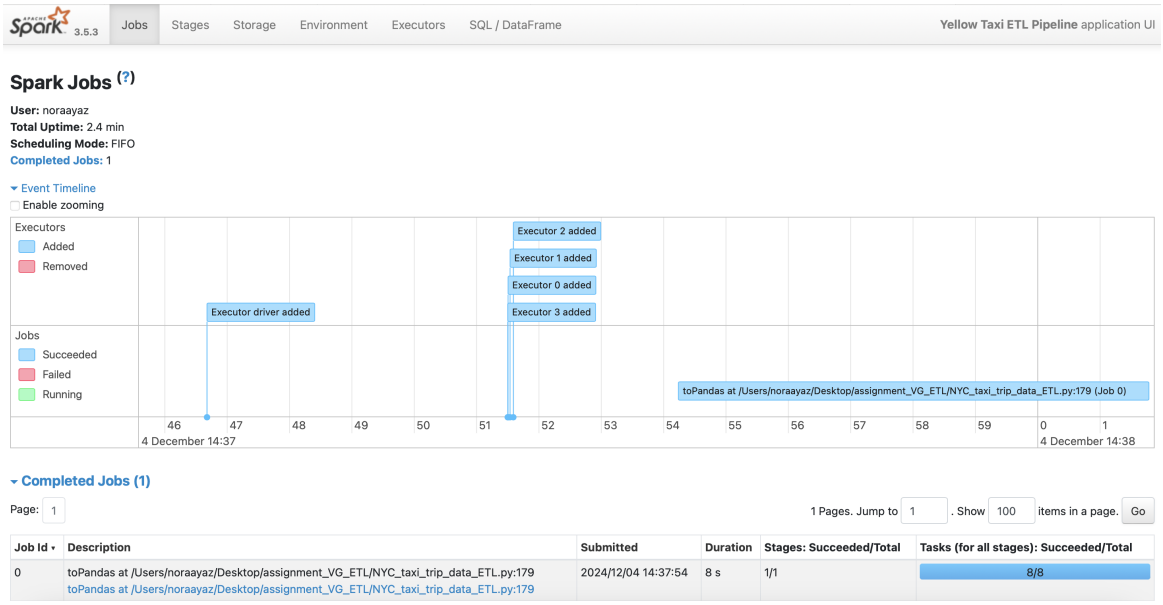

Figure 10: Intermediate stage of data processing in Spark.

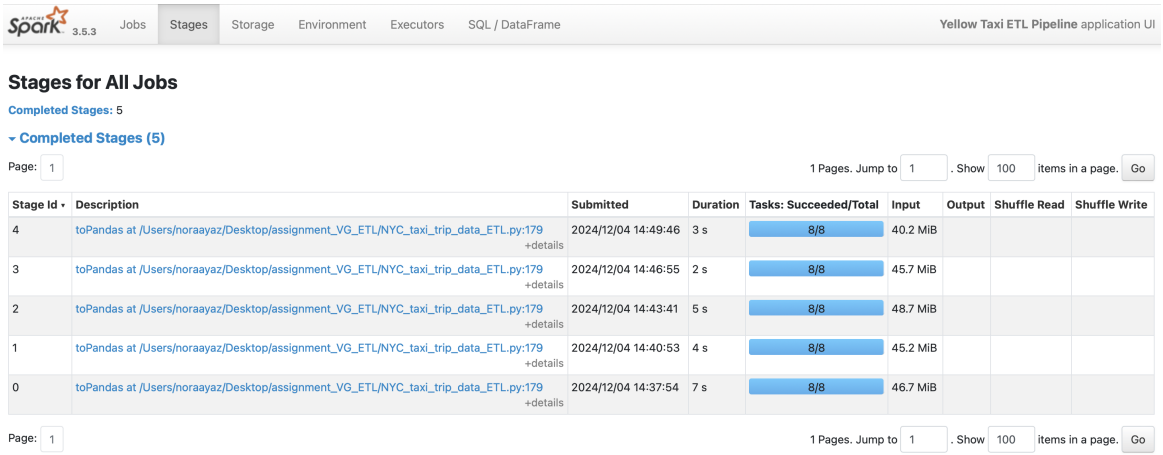

Figure 11: Details of completed stages in Spark.

4 Automation with Cron

To ensure the pipeline runs periodically without manual intervention, a `cron` job was configured. Below is the configuration used:

```
# Cron job for running the ETL pipeline on the 1st of every month at
    midnight
0 0 1 * * /Users/noraayaz/Desktop/assignment_VG_ETL/NYC_taxi_trip_data_ETL.
    py
```

The `crontab` configuration was validated, as shown below:

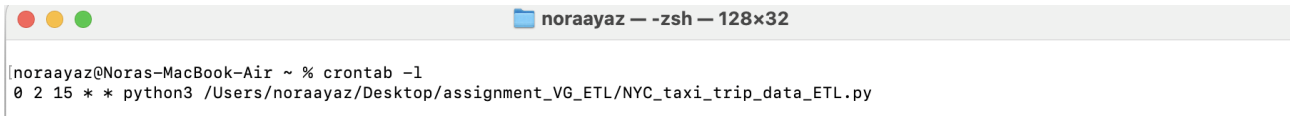

Figure 12: Cron job setup for automatic execution of the ETL pipeline.

5 Validation and Results

5.1 Worker Contributions

The contribution of all 4 workers was validated using the Spark UI. Each worker utilized 2 cores and contributed equally to the ETL tasks. Below is a screenshot demonstrating worker contributions:

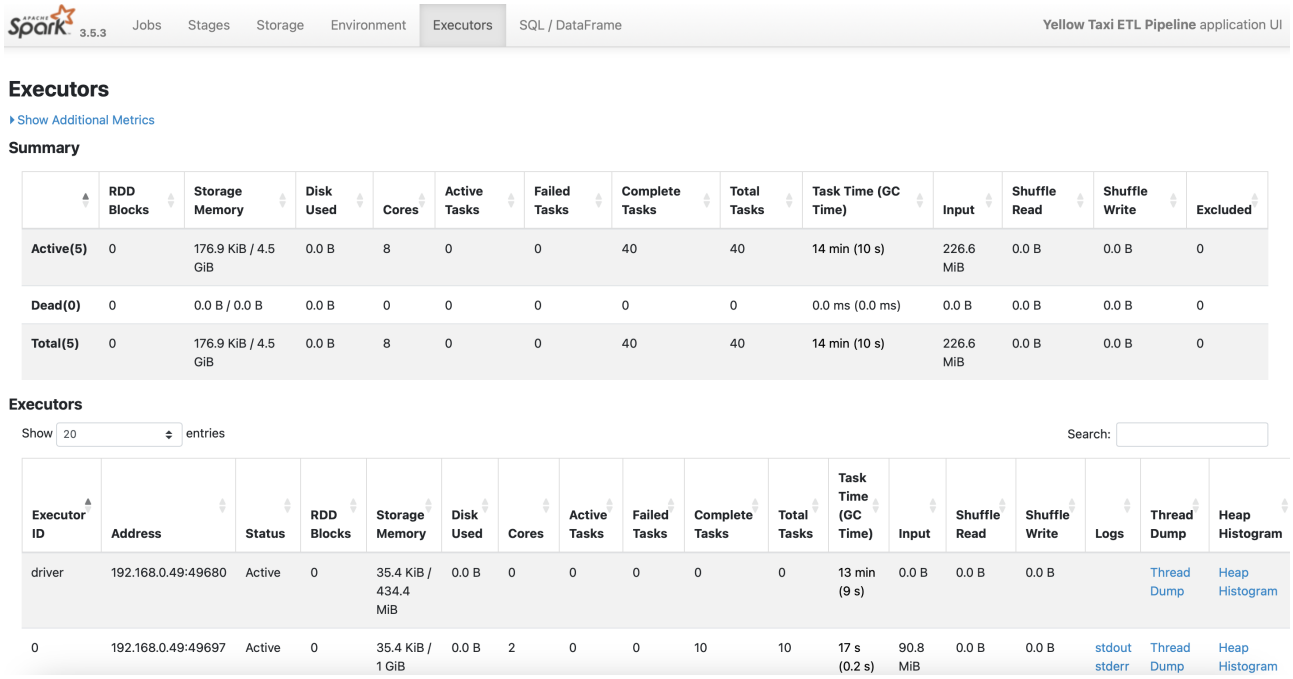

Figure 13: Spark UI showing worker contributions.

5.2 MongoDB Validation

The processed data was successfully loaded into MongoDB. Validation steps included:

- Verifying data insertion into the `processed_trips` collection.
- Ensuring accurate calculation of metrics such as trip distance and tip ratio.
- Checking total storage size and document count.

The following screenshots provide evidence of successful data storage and validation:

```
test> use yellow_taxi_db
switched to db yellow_taxi_db
yellow_taxi_db> show collections
processed_trips
test_collection
trips
yellow_taxi_db> db.processed_trips.find().count()
(node:21688) [MONGODB DRIVER] Warning: cursor.count is deprecated and will be removed in the next major version, plea
se use `collection.estimatedDocumentCount` or `collection.countDocuments` instead
(Use `node --trace-warnings ...` to show where the warning was created)
85534589
yellow_taxi_db> db.processed_trips.find().limit(5)
[
  {
    _id: ObjectId('675036978bc12a3b0cd7edf2'),
    VendorID: 1,
    tpep_pickup_datetime: ISODate('2023-02-01T01:32:53.000Z'),
    tpep_dropoff_datetime: ISODate('2023-02-01T01:34:34.000Z'),
    trip_distance: 0.3,
    PULocationID: 142,
    DOLocationID: 163,
    tip_amount: 0,
    total_amount: 9.4,
    trip_duration_minutes: 1.6833333333333333,
    tip_ratio: 0,
    distance_segment: 'short'
  },
```

Figure 14: MongoDB collection preview showing processed trips.

```
test_mongodb.py > ...
1   from pymongo import MongoClient
2
3   client = MongoClient("mongodb://localhost:27017/")
4   db = client["yellow_taxi_db"]
5   collection = db["processed_trips"]
6
7   total_records = collection.count_documents({})
8   print(f"Total records: {total_records}")
9
10  for doc in collection.find().limit(5):
11      print(doc)
12
```

Figure 15: Python script to retrieve total records and sample data from MongoDB.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

venvnoraayaz@Noras-MacBook-Air assignment_VG_ETL % python test_mongodb.py
Total records: 85534589
{'_id': ObjectId('675036978bc12a3b0cd7edf2'), 'VendorID': 1, 'tpep_pickup_datetime': datetime.datetime(2023, 2, 1, 1, 32, 53), 'tpep_dropoff_datetime': datetime.datetime(2023, 2, 1,
 1, 34, 34), 'trip_distance': 0.3, 'PULocationID': 142, 'DOLocationID': 163, 'tip_amount': 0.0, 'total_amount': 9.4, 'trip_duration_minutes': 1.6833333333333333, 'tip_ratio': 0.0, '
distance_segment': 'short'}
{'_id': ObjectId('675036978bc12a3b0cd7edf3'), 'VendorID': 1, 'tpep_pickup_datetime': datetime.datetime(2023, 2, 1, 1, 29, 33), 'tpep_dropoff_datetime': datetime.datetime(2023, 2, 1,
 2, 1, 38), 'trip_distance': 18.8, 'PULocationID': 132, 'DOLocationID': 26, 'tip_amount': 0.0, 'total_amount': 74.65, 'trip_duration_minutes': 32.083333333333336, 'tip_ratio': 0.0,
'distance_segment': 'long'}
{'_id': ObjectId('675036978bc12a3b0cd7edf4'), 'VendorID': 2, 'tpep_pickup_datetime': datetime.datetime(2023, 2, 1, 1, 12, 28), 'tpep_dropoff_datetime': datetime.datetime(2023, 2, 1,
 1, 25, 46), 'trip_distance': 3.22, 'PULocationID': 161, 'DOLocationID': 145, 'tip_amount': 3.3, 'total_amount': 25.3, 'trip_duration_minutes': 13.3, 'tip_ratio': 0.13043478260869565, 'distance_segment': 'medium'}
{'_id': ObjectId('675036978bc12a3b0cd7edf5'), 'VendorID': 1, 'tpep_pickup_datetime': datetime.datetime(2023, 2, 1, 1, 52, 40), 'tpep_dropoff_datetime': datetime.datetime(2023, 2, 1,
 2, 7, 18), 'trip_distance': 5.1, 'PULocationID': 148, 'DOLocationID': 236, 'tip_amount': 5.35, 'total_amount': 32.25, 'trip_duration_minutes': 14.633333333333333, 'tip_ratio': 0.16589147286821704, 'distance_segment': 'medium'}
{'_id': ObjectId('675036978bc12a3b0cd7edf6'), 'VendorID': 1, 'tpep_pickup_datetime': datetime.datetime(2023, 2, 1, 1, 12, 39), 'tpep_dropoff_datetime': datetime.datetime(2023, 2, 1,
 1, 40, 36), 'trip_distance': 8.9, 'PULocationID': 137, 'DOLocationID': 244, 'tip_amount': 3.5, 'total_amount': 50.0, 'trip_duration_minutes': 27.95, 'tip_ratio': 0.07, 'distance_se
gment': 'medium'}
venvnoraayaz@Noras-MacBook-Air assignment_VG_ETL %
```

Figure 16: Validation of processed data using Python script.

5.3 Storage Statistics

The MongoDB storage statistics confirmed that over 85 million records were processed, occupying a total storage size of approximately 5.68 GB:

Figure 17: MongoDB storage statistics obtained via Mongo shell.

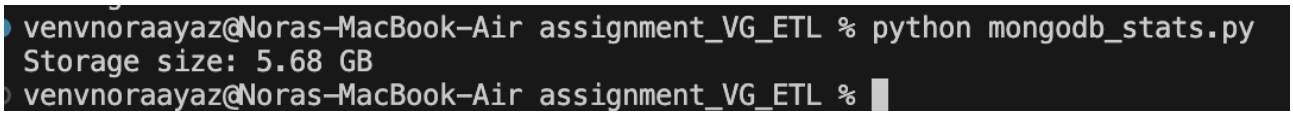

Figure 18: Python script validating MongoDB storage size.

5.4 Processing Results

The following additional logs showcase the full cycle of data extraction, transformation, and loading:

- **Processing all files:** The pipeline successfully processed available files, including 2023 and the first 9 months of 2024.
- **Validation of no pending files:** The pipeline confirmed that no new files were left to process after completion.

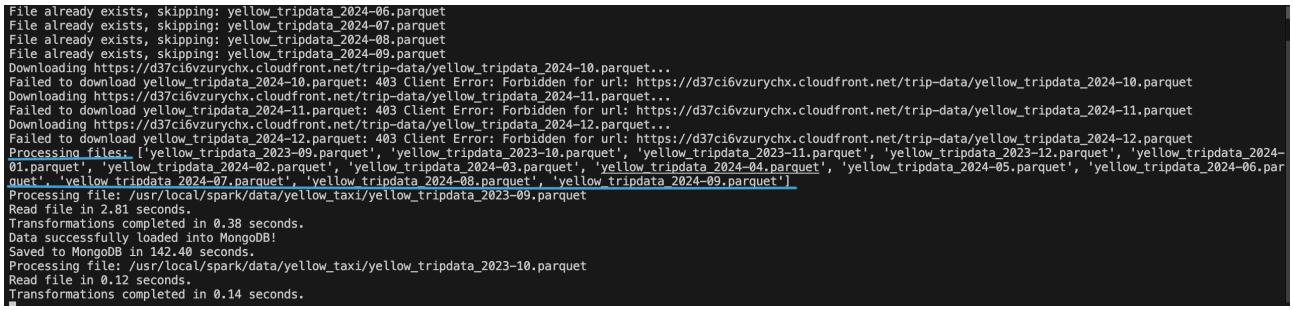

Figure 19: Processing log showing the pipeline handling available files.

Figure 20: Validation log showing no new files to process.

```
Transformations completed in 0.12 seconds.
Data successfully loaded into MongoDB!
Saved to MongoDB in 145.27 seconds.
Processing file: /usr/local/spark/data/yellow_taxi/yellow_tripdata_2024-08.parquet
Read file in 0.12 seconds.
Transformations completed in 0.07 seconds.
Data successfully loaded into MongoDB!
Saved to MongoDB in 143.41 seconds.
Processing file: /usr/local/spark/data/yellow_taxi/yellow_tripdata_2024-09.parquet
Read file in 0.09 seconds.
Transformations completed in 0.07 seconds.
Data successfully loaded into MongoDB!
Saved to MongoDB in 189.65 seconds.
venvnoraayaz@Noras-MacBook-Air assignment_VG_ETL %
```

Figure 21: Final pipeline completion log showing successful processing of all files.

6 Conclusion

The NYC taxi trip ETL pipeline was successfully implemented and automated. Key achievements include:

- Setting up a local Spark cluster with 4 workers to process data in parallel.
- Transforming raw data into meaningful insights and storing it in MongoDB.
- Automating the pipeline with `cron` for seamless operation.

Appendices

Python Script

Below is the complete Python script for the ETL pipeline (`NYC_taxi_trip_data_ETL.py`):

```python
import os
import json
import requests
from time import time
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, unix_timestamp
from pyspark.sql.types import StructType, StructField, IntegerType,
    DoubleType, LongType, StringType, TimestampType
from pymongo import MongoClient
import pyarrow.parquet as pq
import pyarrow as pa

# Initialize Spark session
spark = SparkSession.builder \
    .master("spark://localhost:7077") \
    .appName("Yellow Taxi ETL Pipeline") \
    .config("spark.executor.memory", "2g") \
    .config("spark.executor.cores", "2") \
    .config("spark.cores.max", "8") \
    .config("spark.sql.shuffle.partitions", "16") \
    .getOrCreate()

# Base URL and file patterns
base_url = "https://d37ci6vzurychx.cloudfront.net/trip-data/"
file_pattern = "yellow_tripdata_{year}-{month:02d}.parquet"
```

```
# Directories for files
local_folder = "/usr/local/spark/data/yellow_taxi/"
processed_folder = "/usr/local/spark/data/yellow_taxi_fixed/"
metadata_downloaded = os.path.join(local_folder, "downloaded_files.json")
metadata_processed = os.path.join(processed_folder, "processed_files.json")

# Create directories if they don't exist
os.makedirs(local_folder, exist_ok=True)
os.makedirs(processed_folder, exist_ok=True)

# List of years to process
years = [2023, 2024]

# Helper functions for metadata management
def load_metadata(path):
    try:
        with open(path, "r") as f:
            return json.load(f)
    except FileNotFoundError:
        return []

def save_metadata(path, data):
    with open(path, "w") as f:
        json.dump(data, f)

# Step 1: Download missing files
def download_new_files():
    """
    Download Yellow Taxi data files that are not already downloaded.
    """
    downloaded_files = load_metadata(metadata_downloaded)

    for year in years:
        for month in range(1, 13):
            file_name = file_pattern.format(year=year, month=month)
            file_url = base_url + file_name
            local_file_path = os.path.join(local_folder, file_name)

            if file_name in downloaded_files or os.path.exists(
               local_file_path):
                print(f"File already exists, skipping: {file_name}")
                continue

            try:
                print(f"Downloading {file_url}...")
                response = requests.get(file_url)
                response.raise_for_status()
                with open(local_file_path, "wb") as f:
                    f.write(response.content)
                print(f"Successfully downloaded: {file_name}")
                downloaded_files.append(file_name)
                save_metadata(metadata_downloaded, downloaded_files)
            except requests.exceptions.RequestException as e:
                print(f"Failed to download {file_name}: {e}")

# Step 2: Process and transform files
def process_files():
    """
    Process downloaded files and save transformed data to MongoDB.
    """
    downloaded_files = load_metadata(metadata_downloaded)
    processed_files = load_metadata(metadata_processed)
```

```
    to_process_files = [file for file in downloaded_files if file not in
        processed_files]

    if not to_process_files:
        print("No new files to process.")
        return

    print(f"Processing files: {to_process_files}")

    # Define schema for Spark
    schema_spark = StructType([
        StructField("VendorID", IntegerType(), True),
        StructField("tpep_pickup_datetime", TimestampType(), True),
        StructField("tpep_dropoff_datetime", TimestampType(), True),
        StructField("passenger_count", LongType(), True),
        StructField("trip_distance", DoubleType(), True),
        StructField("RatecodeID", LongType(), True),
        StructField("store_and_fwd_flag", StringType(), True),
        StructField("PULocationID", IntegerType(), True),
        StructField("DOLocationID", IntegerType(), True),
        StructField("payment_type", LongType(), True),
        StructField("fare_amount", DoubleType(), True),
        StructField("extra", DoubleType(), True),
        StructField("mta_tax", DoubleType(), True),
        StructField("tip_amount", DoubleType(), True),
        StructField("tolls_amount", DoubleType(), True),
        StructField("improvement_surcharge", DoubleType(), True),
        StructField("total_amount", DoubleType(), True),
        StructField("congestion_surcharge", DoubleType(), True),
        StructField("Airport_fee", DoubleType(), True)
    ])

    for file_name in to_process_files:
        try:
            original_file = os.path.join(local_folder, file_name)
            converted_file = os.path.join(processed_folder, file_name)

            print(f"Processing file: {original_file}")

            # Start timer
            start_time = time()

            # Read transformed file into Spark
            spark_df = spark.read.schema(schema_spark).parquet(original_file
                )

            read_time = time() - start_time
            print(f"Read file in {read_time:.2f} seconds.")

            # Transform data in Spark
            start_time = time()
            spark_df = spark_df.filter((col("trip_distance") > 0) & (col("
                total_amount") > 0))
            spark_df = spark_df.filter(unix_timestamp("tpep_dropoff_datetime
                ") > unix_timestamp("tpep_pickup_datetime"))
            spark_df = spark_df.select(
                "VendorID", "tpep_pickup_datetime", "tpep_dropoff_datetime",
                "trip_distance", "PULocationID", "DOLocationID", "tip_amount
                    ", "total_amount"
            ).withColumn(
                "trip_duration_minutes",
                (unix_timestamp("tpep_dropoff_datetime") - unix_timestamp("
```

```
                    tpep_pickup_datetime")) / 60
            ).withColumn(
                "tip_ratio", col("tip_amount") / col("total_amount")
            ).withColumn(
                "distance_segment",
                when(col("trip_distance") <= 2, "short")
                .when(col("trip_distance") <= 10, "medium")
                .otherwise("long")
            )

            transform_time = time() - start_time
            print(f"Transformations completed in {transform_time:.2f}
                seconds.")

            # Save transformed data to MongoDB
            start_time = time()
            save_to_mongodb(spark_df)

            mongo_time = time() - start_time
            print(f"Saved to MongoDB in {mongo_time:.2f} seconds.")

            # Mark file as processed
            processed_files.append(file_name)
            save_metadata(metadata_processed, processed_files)
        except Exception as e:
            print(f"Error processing file {file_name}: {e}")

# Step 3: Save data to MongoDB
def save_to_mongodb(df):
    """
    Save the processed DataFrame to MongoDB.
    """
    try:
        client = MongoClient("mongodb://localhost:27017/")
        db = client["yellow_taxi_db"]
        collection = db["processed_trips"]

        pandas_df = df.toPandas()
        collection.insert_many(pandas_df.to_dict("records"))
        print("Data successfully loaded into MongoDB!")
    except Exception as e:
        print(f"Error during MongoDB insertion: {e}")

# Execute ETL pipeline
download_new_files()
process_files()
```