NYC Taxi Pickup Hotspot Analysis using Hadoop MapReduce

Module Name:	Cloud Computing
Module Number:	EC7205
Assignment Title:	Large-Scale Data Analysis Using MapReduce
Group Number:	03

Team Members:

Abeysekara P.K. (EG/2020/3799)
Aralugaswaththa S.V.C.R.P (EG/2020/3827)
De Silva K.B.L.H. (EG/2020/3882)

June 7, 2025

Contents

1 Quick Start: Get Up and Running

This section provides the essential commands to clone, build, and run the NYC Taxi Hotspot Analysis application. For detailed explanations, refer to the subsequent sections.

Prerequisites:

- Java Development Kit (JDK) 1.8+
- Apache Maven 3.x
- Hadoop 3.3.x (HDFS and YARN services must be running)
- Git

1. Clone the Repository:

```
git clone https://github.com/PasanAbeysekara/Taxi-Pickup-Hotspot-Analysis-using-Hadoop-MapReduce
cd Taxi-Pickup-Hotspot-Analysis-using-Hadoop-MapReduce
```

2. Build the Project: Navigate into the NYCTaxiAnalysis directory first:

```
cd NYCTaxiAnalysis
mvn clean package
```

This creates target/NYCTaxiAnalysis-1.0-SNAPSHOT.jar. After building, you might want to return to the repository root: cd ..

3. Prepare Data:

a. Download Data Files: Download the following files (links also available in Section 2):

- Trip Data: yellow_tripdata_2016-01.parquet
- Lookup Data: taxi_zone_lookup.csv

b. Place Data Files: Create a directory named data at the root of the cloned repository (e.g., Taxi-Pickup-Hotspot-Analysis-using-Hadoop-MapReduce/data/) and place the downloaded files into it.

4. Upload Data to HDFS: (Ensure you are at the root of the cloned repository. Replace <your_username> with your Hadoop username.)

```
# Create HDFS directories (if they don't exist)
hdfs dfs -mkdir -p /user/<your_username>/nyctaxi_input
hdfs dfs -mkdir -p /user/<your_username>/nyctaxi_lookup

# Upload data files from your local 'data' directory
hdfs dfs -put ./data/yellow_tripdata_2016-01.parquet /user/<your_username>/nyctaxi_input/
hdfs dfs -put ./data/taxi_zone_lookup.csv /user/<your_username>/nyctaxi_lookup/
```

5. Run the MapReduce Job: (Ensure you are in the NYCTaxiAnalysis directory where the JAR file was built. Replace <your_username>.)

```
# Remove previous output directory (if any) to prevent errors
hdfs dfs -rm -r /user/<your_username>/nyctaxi_output

# Execute the job
hadoop jar target/NYCTaxiAnalysis-1.0-SNAPSHOT.jar com.nyctaxi.NYCTaxiDriver \
```

```
/user/<your_username>/nyctaxi_input/yellow_tripdata_2016-01.parquet \
/user/<your_username>/nyctaxi_output \
/user/<your_username>/nyctaxi_lookup/taxi_zone_lookup.csv
```

6. View Top N Results: After the job completes, navigate to the root of the cloned repository.

```
# Get the merged output from HDFS to a local file
hdfs dfs -getmerge /user/<your_username>/nyctaxi_output ./final_output.txt

# Run the Python script to display Top N results
python3 DataInvestigation/get_top_n.py ./final_output.txt

# Optional: Clean up the local merged file
# rm ./final_output.txt
```

---

2 Prerequisite Installation

Before proceeding with the setup and execution, ensure you have the necessary prerequisites. We provide helper scripts to guide you through the installation on Linux/macOS and Windows. These scripts should be located in the root of the repository (`install_prerequisites.sh` and `install_prerequisites.bat`).

2.1 For Linux/macOS Users:

1. Clone this repository if you haven't already.
2. Navigate to the repository root directory.
3. Make the script executable:

```
chmod +x install_prerequisites.sh

```

4. Run the script:

```
./install_prerequisites.sh

```

   This script will attempt to install Git, OpenJDK (1.8 or a newer LTS), and Maven using your system's package manager. It will also download Apache Hadoop (e.g., 3.3.6) to `$HOME/hadoop/hadoop-3.3.6` (or similar, check script output).

   IMPORTANT (Hadoop): The script only downloads and extracts Hadoop. You **MUST** configure Hadoop manually. This includes:

   - Setting the `HADOOP_HOME` environment variable.
   - Adding `$HADOOP_HOME/bin` and `$HADOOP_HOME/sbin` to your `PATH`.
   - Configuring `JAVA_HOME` within `$HADOOP_HOME/etc/hadoop/hadoop-env.sh`.
   - Setting up Hadoop configuration files (`core-site.xml`, `hdfs-site.xml`, etc.) as per the official Hadoop documentation or any specific setup scripts you might have (like the `install_hadoop.sh` mentioned elsewhere in this project).

2.2 For Windows Users:

1. Clone this repository if you haven't already.
2. Navigate to the repository root directory.
3. Run the batch script:

```
install_prerequisites.bat

```

   This script will provide guidance and links for manually installing Git, OpenJDK (1.8 or newer LTS), and Apache Maven. It will also guide you on setting up the necessary environment variables (`JAVA_HOME`, `M2_HOME/MAVEN_HOME`, `PATH`).

   IMPORTANT (Hadoop on Windows):

   - **WSL2 Recommended:** Running Hadoop natively on Windows can be complex. We **strongly recommend** using Windows Subsystem for Linux 2 (WSL2) and then following the Linux/macOS installation script (`install_prerequisites.sh`) within your WSL2 environment.

- **Native Windows (Advanced):** If you choose to install Hadoop natively on Windows, the script provides general guidance. You will need to:
  - Download the Hadoop binaries.
  - Obtain the correct `winutils.exe` and other Windows-specific Hadoop files for your Hadoop version.
  - Manually configure `HADOOP_HOME`, `PATH`, and Hadoop configuration files (`core-site.xml`, `hdfs-site.xml`, `hadoop-env.cmd`, etc.).

2.3 After Running the Scripts:

- Verify each prerequisite is installed correctly and their respective `..._HOME` variables and `PATH` are properly configured.
- You might need to open a new terminal/Command Prompt session for all environment variable changes to take effect.
- For Hadoop, proceed with the detailed configuration steps as outlined in its official documentation or specific instructions relevant to your setup (e.g., single-node cluster setup).

3 Project Objective & Task

The primary objective is to identify the busiest taxi pickup locations within New York City by processing one month of taxi trip records (January 2016). This involves:

- Counting the total number of pickups for each distinct taxi zone.
- Joining these counts with a lookup table to associate zone IDs with human-readable names (Zone and Borough).
- Presenting the Top N busiest zones to highlight areas of high taxi demand.

This task aligns with typical "aggregation" and "counting" patterns well-suited for MapReduce, similar to "Log Analysis" (extracting top IPs) or "Sales Aggregation" mentioned in the assignment brief.

4 Dataset Chosen & Appropriateness

We selected a **publicly available dataset** as encouraged by the assignment guidelines. The dataset consists of two main parts:

a. NYC Yellow Taxi Trip Data:

- **Source:** NYC Taxi & Limousine Commission (TLC) Trip Record Data (a well-known public dataset) - Official Link
- **File Used:** `yellow_tripdata_2016-01.parquet` (Data for January 2016)
- **Format:** Apache Parquet (a columnar format suitable for large-scale data processing)
- **Size:** Approximately 10.9 million records, significantly exceeding the 100,000-row minimum requirement.
- **Complexity & Realism:** This is real-world data, inherently complex with numerous fields, varied data types, and potential for missing/dirty data, making it a realistic and challenging dataset for MapReduce.
- **Relevant Column for Task:** `PULocationID` (Pickup Location ID).
- **Download Link:** yellow_tripdata_2016-01.parquet

b. Taxi Zone Lookup Table:

- **Source:** NYC TLC - Taxi Zones
- **File Used:** `taxi_zone_lookup.csv`
- **Format:** CSV
- **Relevant Columns:** `LocationID`, `Borough`, `Zone`
- **Purpose:** Essential for converting numeric `PULocationID`s into meaningful, interpretable results (Zone and Borough names). This join operation adds to the complexity and real-world applicability of the task.
- **Download Link:** taxi_zone_lookup.csv

The chosen dataset is public, real-world, and large-scale (10.9M records). Its structure (Parquet format, multiple columns) and the nature of the task (aggregation, join) are well-suited for a MapReduce solution. The task addresses a real-world problem of identifying high-demand areas, demonstrating the appropriateness of the dataset for the assignment.

4.1 Data Investigation

To understand the dataset structure, run the provided analysis script. Ensure the Parquet and CSV files are downloaded (e.g., into a parent directory `../` relative to `DataInvestigation/` or adjust path in script).

```
python3 DataInvestigation/analysis.py
```

Output of `analysis.py` (summary):

```
--- Exploring Parquet File: ../yellow_tripdata_2016-01.parquet ---

1. Basic Information:
Shape (rows, columns): (10905067, 19)

First 5 rows:
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  ...  total_amount  congestion_surcharge  airport_fee
0         1  2016-01-01 00:12:22   2016-01-01 00:29:14  ...         18.36                  None         None
1         1  2016-01-01 00:41:31   2016-01-01 00:55:10  ...         10.80                  None         None
2         1  2016-01-01 00:53:37   2016-01-01 00:59:57  ...          7.30                  None         None
3         1  2016-01-01 00:13:28   2016-01-01 00:18:07  ...          6.30                  None         None
4         1  2016-01-01 00:33:04   2016-01-01 00:47:14  ...         12.30                  None         None

[5 rows x 19 columns]

Column Data Types and Non-Null Counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10905067 entries, 0 to 10905066
Data columns (total 19 columns):
 #   Column                 Non-Null Count     Dtype
---  ------                 --------------     -----
 0   VendorID               10905067 non-null  int64
 1   tpep_pickup_datetime   10905067 non-null  datetime64[us]
 2   tpep_dropoff_datetime  10905067 non-null  datetime64[us]
 3   passenger_count        10905067 non-null  int64
 4   trip_distance          10905067 non-null  float64
 5   RatecodeID             10905067 non-null  int64
 6   store_and_fwd_flag     10905067 non-null  object
 7   PULocationID           10905067 non-null  int64
 8   DOLocationID           10905067 non-null  int64
 9   payment_type           10905067 non-null  int64
 10  fare_amount            10905067 non-null  float64
 11  extra                  10905067 non-null  float64
 12  mta_tax                10905067 non-null  float64
 13  tip_amount             10905067 non-null  float64
 14  tolls_amount           10905067 non-null  float64
 15  improvement_surcharge  10905067 non-null  float64
 16  total_amount           10905067 non-null  float64
 17  congestion_surcharge   0 non-null         object
 18  airport_fee            0 non-null         object
dtypes: datetime64[us](2), float64(8), int64(6), object(3)
memory usage: 1.5+ GB

2. Null Value Analysis:
                      Null Count  Null Percentage (%)
congestion_surcharge    10905067                100.0
airport_fee             10905067                100.0

3. Descriptive Statistics (Numerical Columns):
           VendorID        tpep_pickup_datetime  ... improvement_surcharge  total_amount
count  1.090507e+07                    10905067  ...          1.090507e+07  1.090507e+07
mean   1.535005e+00  2016-01-16 13:45:41.820952  ...          2.997352e-01  1.564197e+01
min    1.000000e+00         2016-01-01 00:00:00  ...         -3.000000e-01 -9.584000e+02
25%    1.000000e+00         2016-01-09 00:03:34  ...          3.000000e-01  8.300000e+00
50%    2.000000e+00         2016-01-16 03:54:24  ...          3.000000e-01  1.162000e+01
75%    2.000000e+00  2016-01-23 13:08:56.500000  ...          3.000000e-01  1.716000e+01
max    2.000000e+00         2016-01-31 23:59:59  ...          3.000000e-01  1.112716e+05
std    4.987731e-01                         NaN  ...          1.218746e-02  3.637961e+01

[8 rows x 16 columns]

4. Descriptive Statistics (Object/Categorical Columns):
              tpep_pickup_datetime       tpep_dropoff_datetime store_and_fwd_flag congestion_surcharge airport_fee
count                     10905067                    10905067           10905067                    0           0
unique                         NaN                         NaN                  2                    0           0
top                            NaN                         NaN                  N                  NaN         NaN
freq                           NaN                         NaN           10841883                  NaN         NaN
mean    2016-01-16 13:45:41.820952  2016-01-16 14:00:57.898052                NaN                  NaN         NaN
min            2016-01-01 00:00:00         2016-01-01 00:00:00                NaN                  NaN         NaN
25%            2016-01-09 00:03:34         2016-01-09 00:17:51                NaN                  NaN         NaN
50%            2016-01-16 03:54:24         2016-01-16 04:10:38                NaN                  NaN         NaN
75%     2016-01-23 13:08:56.500000         2016-01-23 13:31:23                NaN                  NaN         NaN
max            2016-01-31 23:59:59         2016-03-28 12:54:26                NaN                  NaN         NaN

5. Specific Columns of Interest for MapReduce (PULocationID):

--- PULocationID Analysis ---
Is PULocationID unique? False
Number of unique PULocationIDs: 261
Nulls in PULocationID: 0
Min PULocationID: 1
Max PULocationID: 265

Top 10 most frequent PULocationIDs:
PULocationID
237    411853
161    392997
236    390744
230    366641
234    365252
162    363725
79     361127
186    356337
170    346698
48     337406
Name: count, dtype: int64
Count of PULocationIDs <= 0: 0

6. Datetime Column Range (tpep_pickup_datetime):
```

```
--- tpep_pickup_datetime Analysis ---
Min pickup datetime: 2016-01-01 00:00:00
Max pickup datetime: 2016-01-31 23:59:59

7. Check 'congestion_surcharge' and 'airport_fee' (problematic object types):

Value counts for 'congestion_surcharge':
congestion_surcharge
None    10905067
Name: count, dtype: int64

Value counts for 'airport_fee':
airport_fee
None    10905067
Name: count, dtype: int64

--- Exploring CSV Lookup File: ../taxi_zone_lookup.csv ---

1. Basic Information:
Shape (rows, columns): (265, 4)

First 5 rows:
   LocationID        Borough                     Zone service_zone
0           1            EWR           Newark Airport          EWR
1           2         Queens              Jamaica Bay    Boro Zone
2           3          Bronx  Allerton/Pelham Gardens    Boro Zone
3           4      Manhattan            Alphabet City  Yellow Zone
4           5  Staten Island            Arden Heights    Boro Zone

Column Data Types and Non-Null Counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 265 entries, 0 to 264
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   LocationID    265 non-null    int64
 1   Borough       264 non-null    object
 2   Zone          264 non-null    object
 3   service_zone  263 non-null    object
dtypes: int64(1), object(3)
memory usage: 8.4+ KB

2. Null Value Analysis:
              Null Count  Null Percentage (%)
service_zone           2             0.754717
Borough                1             0.377358
Zone                   1             0.377358

3. Specific Columns of Interest (LocationID, Borough, Zone):

--- LocationID Analysis (Lookup Table) ---
Is LocationID unique? True
Number of unique LocationIDs: 265
Nulls in LocationID: 0
Min LocationID: 1
Max LocationID: 265

Borough value counts:
Borough
Queens           69
Manhattan        69
Brooklyn         61
Bronx            43
Staten Island    20
EWR               1
Unknown           1
NaN               1
Name: count, dtype: int64

Zone value counts (Top 10):
Zone
Governor's Island/Ellis Island/Liberty Island    3
Corona                                           2
Newark Airport                                   1
Ocean Hill                                       1
Parkchester                                      1
Park Slope                                       1
Ozone Park                                       1
Old Astoria                                      1
Ocean Parkway South                              1
Oakwood                                          1
Name: count, dtype: int64
Number of unique Zones: 261

--- Comparing PULocationIDs from Trip Data with LocationIDs in Lookup Table ---
Number of unique PULocationIDs in trip data: 261
Number of unique LocationIDs in lookup table: 265

All PULocationIDs from trip data are present in the lookup table's LocationIDs (based on unique values).

INFO: 4 LocationIDs found in lookup table but NOT as PULocationIDs in this trip data sample.
Examples: [104, 204, 110, 103]

--- Exploration Complete ---
```

5 MapReduce Job Implementation & Logic

The analysis is performed using a single Hadoop MapReduce job written in **Java**.

Why Hadoop MapReduce is Essential for This Task: The primary dataset (`yellow_tripdata_2016-0` contains approximately **10.9 million records**. Processing this volume of data on a single machine using traditional methods would be inefficient and potentially infeasible due to:

- **Processing Time:** Sequentially reading, parsing, aggregating, and joining millions of records would be very slow.
- **Memory Constraints:** Holding all data or large intermediate aggregations in a single machine's memory could lead to `OutOfMemoryError` exceptions.
- **Lack of Scalability:** Such an approach would not scale to handle larger datasets (e.g., multiple years of data).

Hadoop MapReduce is well-suited for this large-scale data analysis because it provides:

1. **Distributed Storage (HDFS):** Manages the large Parquet file across a cluster (or a single-node setup emulating a cluster).
2. **Distributed & Parallel Processing:** The MapReduce framework automatically splits the input data and distributes "Map" tasks to process these splits concurrently across available nodes or cores. Reducer tasks also benefit from parallelism. This significantly reduces overall processing time.
3. **Fault Tolerance:** The framework can handle task failures by automatically rescheduling them, ensuring job completion even with hardware issues in a larger cluster.
4. **Scalability:** Hadoop's architecture allows for horizontal scaling by adding more nodes to increase processing capacity for even larger datasets.

For our task of counting millions of taxi pickups and joining this data with zone information, Hadoop provides the necessary robust and scalable infrastructure.

5.1 MapReduce Workflow Details & Example:

The core logic involves counting pickups for each `PULocationID` and then translating this ID to a human-readable zone name using a lookup table.

Conceptual Data Snippets for Illustration:

- **Trip Data (from Parquet - relevant field `PULocationID`):**
  - Record 1: `PULocationID = 161`
  - Record 2: `PULocationID = 48`
  - Record 3: `PULocationID = 161`
  - Record 4: `PULocationID = 230`
  - Record 5: `PULocationID = 161`
  - *(Imagine 10.9 million such records)*
- **Zone Lookup Data (`taxi_zone_lookup.csv` - relevant fields):**

LocationID	Borough	Zone
48	Manhattan	Clinton East
161	Manhattan	Midtown Center
230	Manhattan	Times Sq/Theatre

1. **Mapper (`PickupLocationMapper.java`):**
   - **Input:** Each row from the `yellow_tripdata_2016-01.parquet` file, represented as a Parquet `Group` object (which allows access to individual fields within a record).
   - **Process:**
     - For each input `Group` (trip record), the mapper extracts the integer value of the `PULocationID` field.
     - It includes robust error handling for scenarios where the Parquet reader might pass a `null Group` object (e.g., due to an empty or malformed split). Such records are skipped, and a Hadoop counter (`NullGroupValueEncountered`) is incremented for monitoring.
   - **Output (Intermediate Key-Value Pairs):** Emits the extracted `PULocationID` as an `IntWritable` key and the integer `1` as an `IntWritable` value. Each `(PULocationID, 1)` pair signifies one observed pickup from that location.
     - Example emissions for the conceptual data:
       * `(IntWritable(161), IntWritable(1))`
       * `(IntWritable(48), IntWritable(1))`
       * `(IntWritable(161), IntWritable(1))`
       * `(IntWritable(230), IntWritable(1))`
       * `(IntWritable(161), IntWritable(1))`

2. **Shuffle and Sort (Hadoop Framework Phase):**
   - This crucial phase, managed entirely by Hadoop, occurs after all Mappers complete.
   - It collects all `(key, value)` pairs emitted by all Mappers.
   - It sorts these pairs based on the key (`PULocationID`).
   - It groups all values associated with the same key, preparing them for the Combiner or Reducer.
   - **Example data after Shuffle & Sort (input to Combiners/Reducers):**
     - `IntWritable(48): [IntWritable(1)]`
     - `IntWritable(161): [IntWritable(1), IntWritable(1), IntWritable(1)]`
     - `IntWritable(230): [IntWritable(1)]`

3. **Combiner (`PickupLocationCombiner.java`):**
   - **Purpose:** This optional but highly recommended phase acts as a "mini-reducer" that runs on the same node where map tasks finished. Its primary goal is to reduce the amount of data transferred over the network to the Reducer nodes, significantly optimizing job performance.
   - **Input:** Receives the sorted and grouped output from the mappers that ran on its local node. For example, if three map outputs for `PULocationID=161` were processed on one node, the Combiner would receive `(IntWritable(161), Iterable<IntWritable> containing three '1's)`.
   - **Process:** Performs a local aggregation by summing the `IntWritable` values (the '1's) for each distinct `PULocationID` it processes.
   - **Output:** Emits aggregated key-value pairs, where the value is now a partial sum.

   - Example emission for key 161 from one combiner instance: `(IntWritable(161), IntWritable(3))`

4. **Reducer** (`PickupLocationReducer.java`):
   - `setup()` **Phase (DistributedCache Join Preparation):**
     - This method is called once per Reducer task before any `reduce()` calls.
     - It loads the `taxi_zone_lookup.csv` file. This file was previously added to the Hadoop DistributedCache by the `NYCTaxiDriver`. The DistributedCache ensures the file is available locally on each node running a Reducer task.
     - The Reducer reads this local CSV file and populates an in-memory `HashMap<Integer, String[]>`, mapping each `LocationID` to an array containing its {`Borough`, `Zone`}.
     - The loading process includes robust CSV parsing logic: it skips the header row, trims whitespace from parsed string values, and handles cases where borough or zone names might be empty or missing by assigning default "Unknown" string values. Hadoop counters are used to track any parsing issues or the number of entries loaded (e.g., `ZoneLookupEntriesLoaded`).
   - `reduce()` **Phase (Final Aggregation & Join):**
     - **Input:** Receives data that has been shuffled and sorted from all Mappers (and Combiners, if used). The input is grouped by `PULocationID` (the key), with an `Iterable` of `IntWritable` values representing the partial sums from Combiners (or individual '1's if no Combiner ran or if keys were unique post-map).
     - **Process:**
       (a) For each `PULocationID` key, iterates through the list of `IntWritable` values and sums them to calculate the `total_pickup_count` for that zone.
       (b) Uses the `PULocationID` (integer value of the key) to look up the corresponding Borough and Zone from the in-memory `zoneLookup` HashMap built in the `setup()` phase.
       (c) If a `PULocationID` from the trip data is not found in the lookup table, it formats the output string to indicate an unknown zone and increments an error counter (`IDNotFoundInCache`).
     - **Output:** Emits the final key-value pairs to HDFS. The key is a `Text` object containing the formatted "Zone Name (Borough)", and the value is an `IntWritable` representing the `total_pickup_count`.

This comprehensive MapReduce pipeline efficiently transforms raw trip data into an aggregated summary of pickup hotspots, enriched with human-readable location names.

5.2 Code Structure:

The project's code is designed for modularity, clarity, error resilience, and efficiency:

- **Modularity:** Code is organized into distinct classes (Driver, Mapper, Combiner, Reducer) following MapReduce best practices.

- **Clarity & Readability:** Descriptive naming for variables and methods, along with comments explaining complex logic sections, enhances code understanding.

- **Error Handling:** The Mapper handles potential null input records. The Reducer's CSV parsing is robust. Hadoop counters track processing anomalies.

- **Efficiency:** A Combiner minimizes shuffle data. The DistributedCache facilitates an efficient reduce-side join.
- **Project Organization:** A standard Maven project structure ensures straightforward building and management of dependencies.

.

```
       DataInvestigation/              # Scripts for data
exploration and post-processing
             analysis.py               # Script for initial data
analysis/exploration
             get_top_n.py              # Python script for sorting
and displaying Top N results
       NYCTaxiAnalysis/                # Core MapReduce Java project
 (Maven structure)
             dependency-reduced-pom.xml  # POM generated by Maven
Shade Plugin
             pom.xml                   # Maven project configuration
             src/
                 main/
                       java/
                           com/
                               nyctaxi/  # Java package structure
                                   NYCTaxiDriver.java
                                   PickupLocationCombiner.java
                                   PickupLocationMapper.java
                                   PickupLocationReducer.java
                 test/
                     java/
                         com/
                             nyctaxi/
                                 AppTest.java
       README.md
       images/
             1.png
             ... (other image files) ...
             image.png
       install_hadoop.sh
```

6 Setup Environment & Execution

6.1 Prerequisites:

- Java Development Kit (JDK) 1.8 or higher.
- Apache Maven 3.x.
- Hadoop 3.3.x (A single-node cluster was used for this project, installed locally on Ubuntu). HDFS and YARN services must be running.
- Git for cloning the repository.

6.2 Hadoop Installation Evidence:

Screenshots demonstrating a functional Hadoop environment:

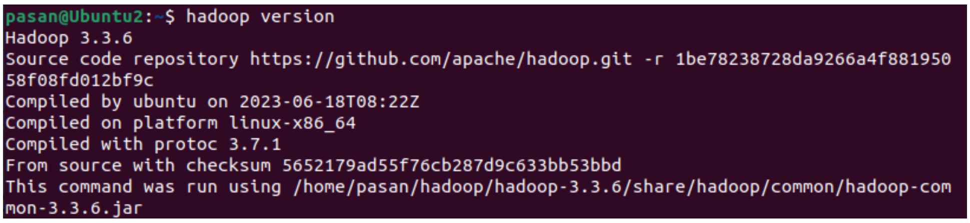


Figure 1: HDFS Status

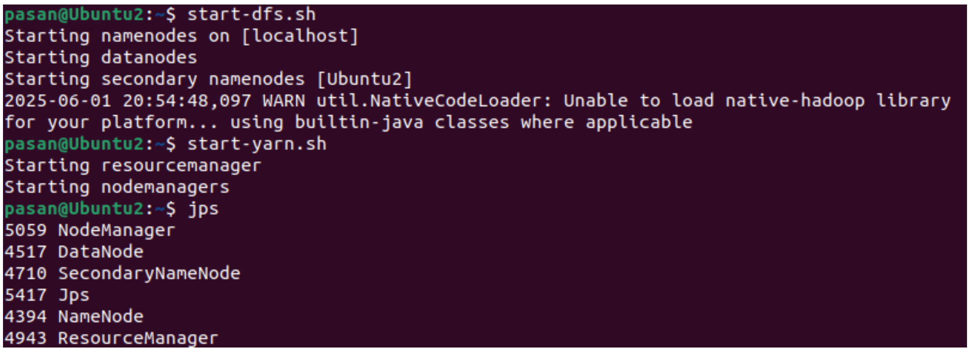


Figure 2: YARN Status

6.3 Steps to Run (Detailed):

These steps provide a comprehensive guide. For a quicker set of commands, see the Quick Start section.

1. **Clone the Repository:**

```
git clone https://github.com/PasanAbeysekara/Taxi-Pickup-Hotspot-Analysis-using-Hadoop-MapReduce
cd Taxi-Pickup-Hotspot-Analysis-using-Hadoop-MapReduce

```

2. **Download Data:** Download files as described in Section 2 and place them in a `data/` directory at the project root.

3. **Build the Project:** Navigate to `NYCTaxiAnalysis/` sub-directory:

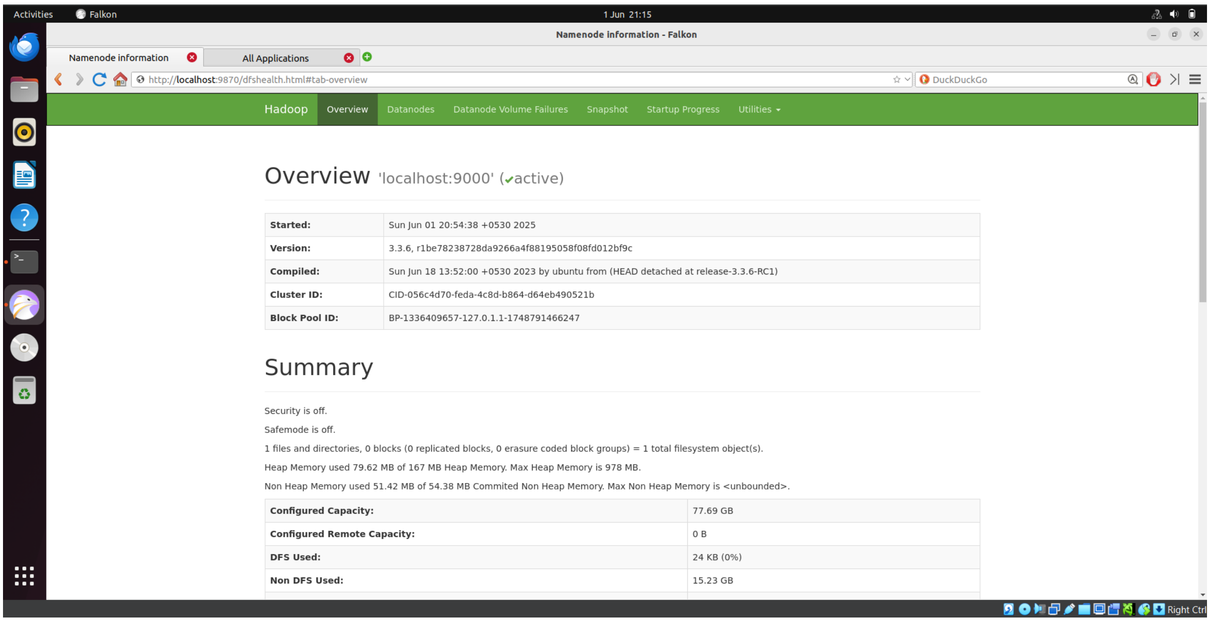


Figure 3: NameNode UI

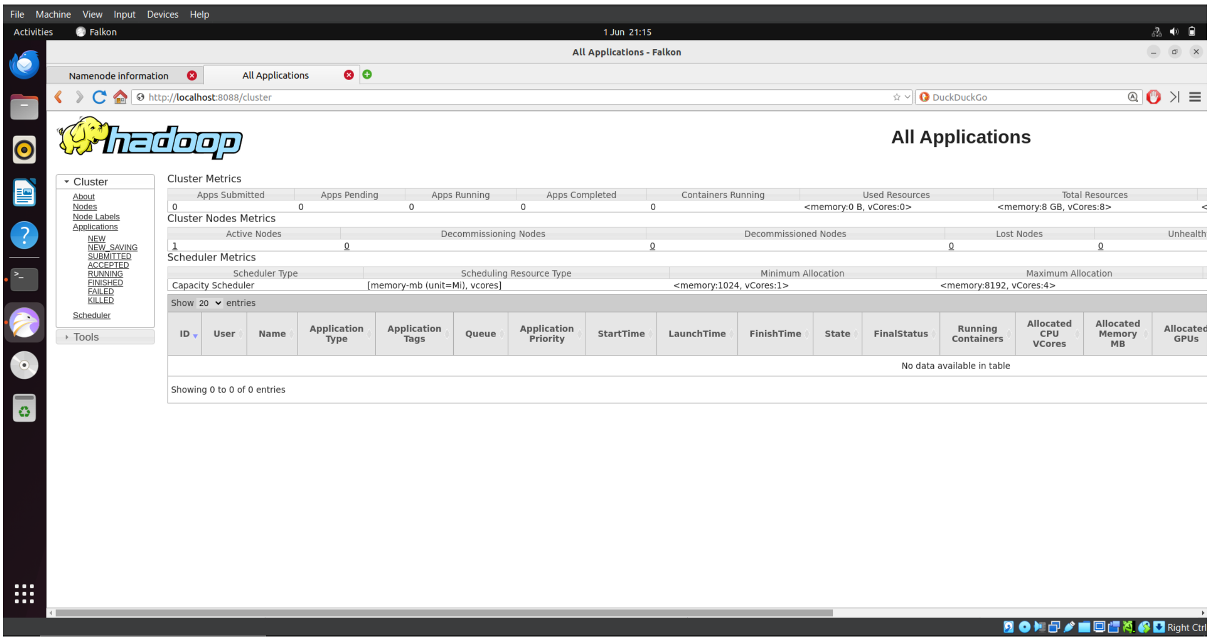


Figure 4: ResourceManager UI

```
cd NYCTaxiAnalysis
mvn clean package

```

4. **Start Hadoop Services (if not already running):**

```
start-dfs.sh
start-yarn.sh
mr-jobhistory-daemon.sh start historyserver

```

5. **Upload Data to HDFS:** (Run from the root of the repository. Replace `<your_username>`.)

```
hdfs dfs -mkdir -p /user/<your_username>/nyctaxi_input
hdfs dfs -mkdir -p /user/<your_username>/nyctaxi_lookup
hdfs dfs -put ./data/yellow_tripdata_2016-01.parquet /user/<
    your_username>/nyctaxi_input/
hdfs dfs -put ./data/taxi_zone_lookup.csv /user/<your_username>/
    nyctaxi_lookup/

```

6. **Run the MapReduce Job:** Navigate to `NYCTaxiAnalysis/`.

```
hdfs dfs -rm -r /user/<your_username>/nyctaxi_output
hadoop jar target/NYCTaxiAnalysis-1.0-SNAPSHOT.jar com.nyctaxi.
    NYCTaxiDriver \
/user/<your_username>/nyctaxi_input/yellow_tripdata_2016-01.parquet
     \
/user/<your_username>/nyctaxi_output \
/user/<your_username>/nyctaxi_lookup/taxi_zone_lookup.csv

```

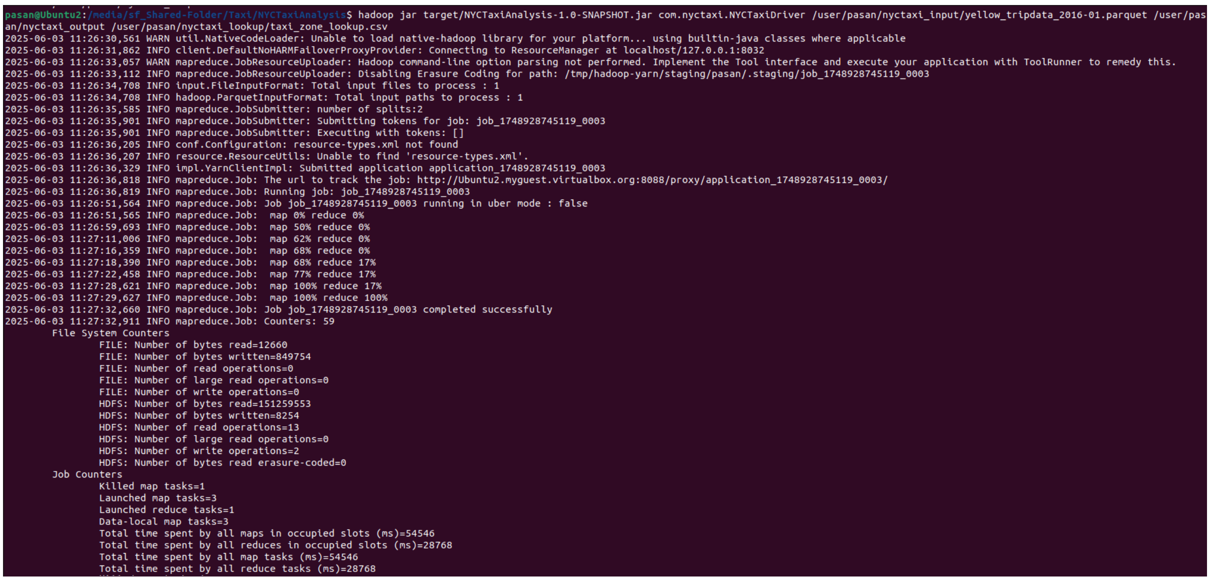


Figure 5: Job Submission & Progress 1

6.4 Execution Output Evidence:

MapReduce Job Log / YARN UI for Counters:

Output Sample (from HDFS):

```
hdfs dfs -cat /user/<your_username>/nyctaxi_output/part-r-00000 | head
    -n 10
```

```
            Total time spent by all reduce tasks (ms)=28768
            Total vcore-milliseconds taken by all map tasks=54546
            Total vcore-milliseconds taken by all reduce tasks=28768
            Total megabyte-milliseconds taken by all map tasks=55855104
            Total megabyte-milliseconds taken by all reduce tasks=29458432
    Map-Reduce Framework
            Map input records=10905067
            Map output records=10905067
            Map output bytes=87240536
            Map output materialized bytes=2622
            Input split bytes=309
            Combine input records=10906069
            Combine output records=1263
            Reduce input groups=261
            Reduce shuffle bytes=2622
            Reduce input records=261
            Reduce output records=261
            Spilled Records=1524
            Shuffled Maps =2
            Failed Shuffles=0
            Merged Map outputs=2
            GC time elapsed (ms)=1539
            CPU time spent (ms)=29170
            Physical memory (bytes) snapshot=1262039040
            Virtual memory (bytes) snapshot=8222445568
            Total committed heap usage (bytes)=959447040
            Peak Map Physical memory (bytes)=755515392
            Peak Map Virtual memory (bytes)=2746245120
            Peak Reduce Physical memory (bytes)=221835264
            Peak Reduce Virtual memory (bytes)=2740129792
    ReducerSetup
            ZoneLookupEntriesLoaded=265
    Shuffle Errors
            BAD_ID=0
            CONNECTION=0
            IO_ERROR=0
            WRONG_LENGTH=0
            WRONG_MAP=0
            WRONG_REDUCE=0
    File Input Format Counters
            Bytes Read=151259244
    File Output Format Counters
            Bytes Written=8254
    parquet
            bytesread=151238658
            bytestotal=151238658
            timeread=496
```

Figure 6: Job Submission & Progress 2

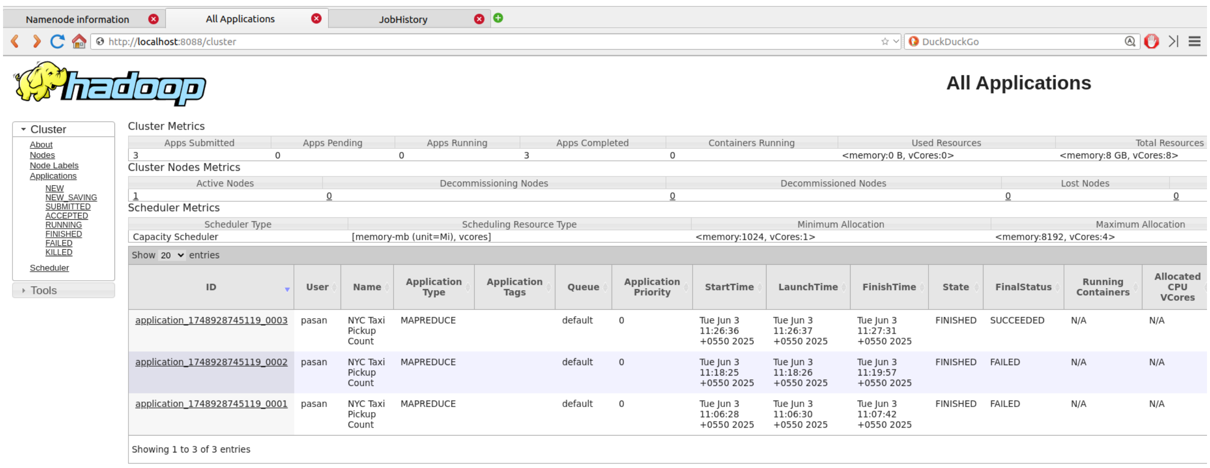


Figure 7: YARN App Running/Completed 1

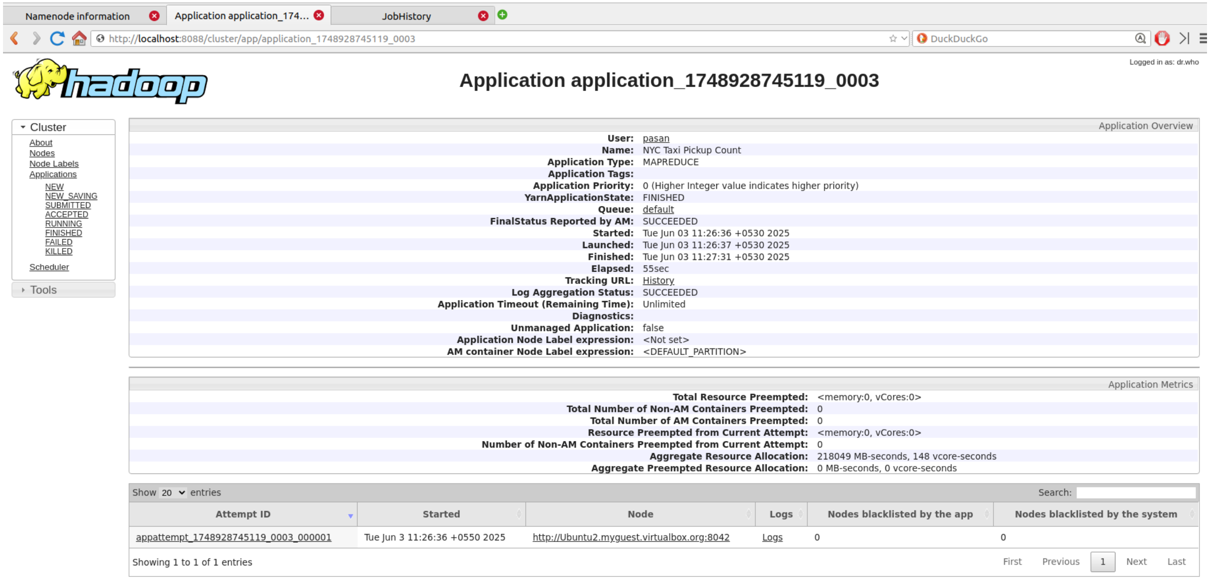


Figure 8: YARN App Running/Completed 2

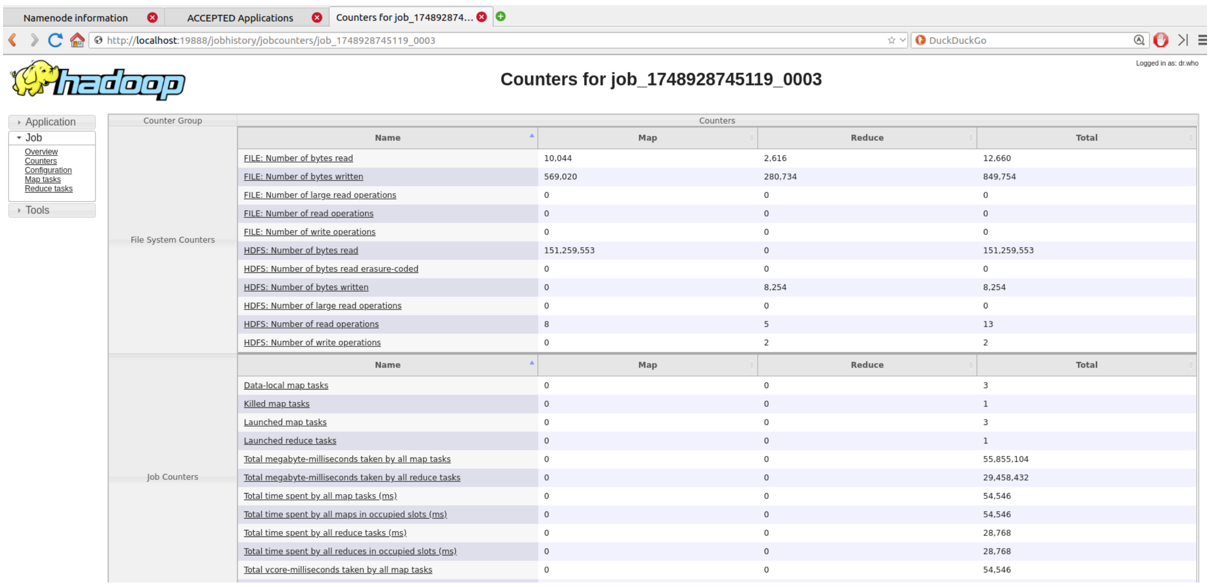
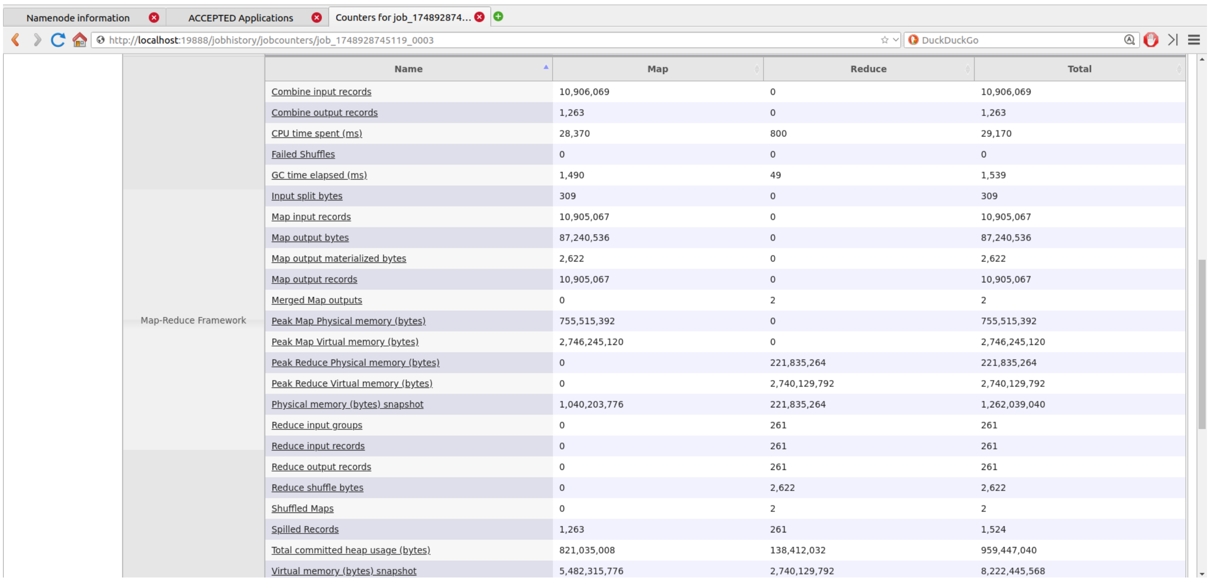
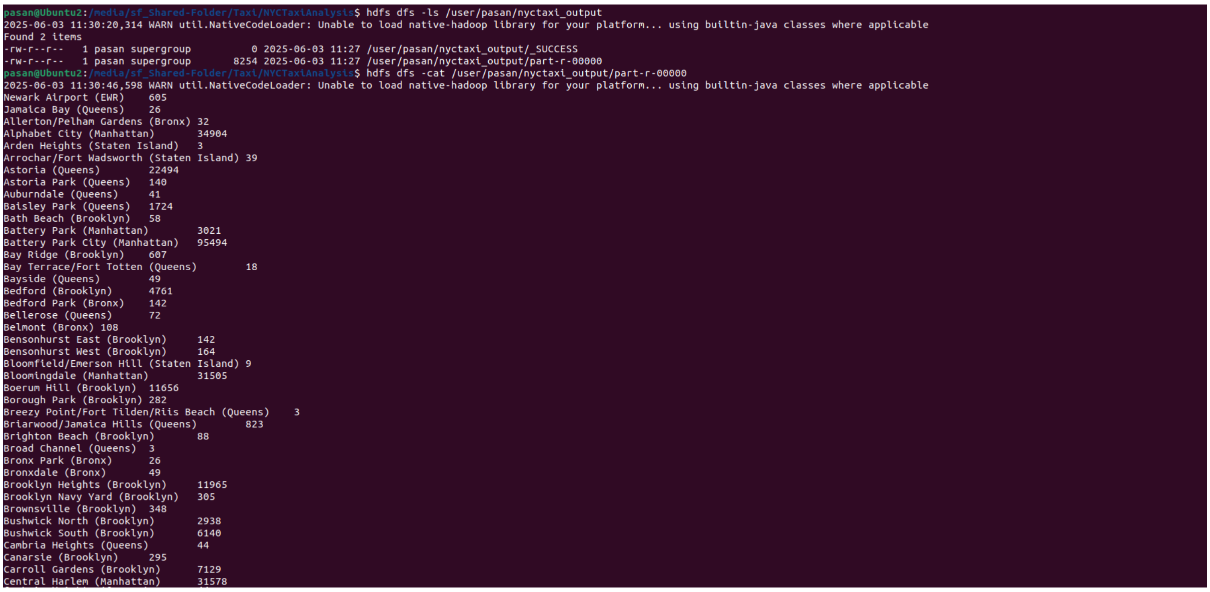


Figure 9: Job Counters 1



	Name	Map	Reduce	Total
Map-Reduce Framework	Combine input records	10,906,069	0	10,906,069
	Combine output records	1,263	0	1,263
	CPU time spent (ms)	28,370	800	29,170
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	1,490	49	1,539
	Input split bytes	309	0	309
	Map input records	10,905,067	0	10,905,067
	Map output bytes	87,240,536	0	87,240,536
	Map output materialized bytes	2,622	0	2,622
	Map output records	10,905,067	0	10,905,067
	Merged Map outputs	0	2	2
	Peak Map Physical memory (bytes)	755,515,392	0	755,515,392
	Peak Map Virtual memory (bytes)	2,746,245,120	0	2,746,245,120
	Peak Reduce Physical memory (bytes)	0	221,835,264	221,835,264
	Peak Reduce Virtual memory (bytes)	0	2,740,129,792	2,740,129,792
	Physical memory (bytes) snapshot	1,040,203,776	221,835,264	1,262,039,040
	Reduce input groups	0	261	261
	Reduce input records	0	261	261
	Reduce output records	0	261	261
	Reduce shuffle bytes	0	2,622	2,622
	Shuffled Maps	0	2	2
	Spilled Records	1,263	261	1,524
	Total committed heap usage (bytes)	821,035,008	138,412,032	959,447,040
	Virtual memory (bytes) snapshot	5,482,315,776	2,740,129,792	8,222,445,568

Figure 10: Job Counters 2



```
pasan@Ubuntu2:/media/sf_Shared-Folder/Taxi/NYCTaxiAnalysis$ hdfs dfs -ls /user/pasan/nyctaxi_output
2025-06-03 11:30:20,314 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   1 pasan supergroup          0 2025-06-03 11:27 /user/pasan/nyctaxi_output/_SUCCESS
-rw-r--r--   1 pasan supergroup       8254 2025-06-03 11:27 /user/pasan/nyctaxi_output/part-r-00000
pasan@Ubuntu2:/media/sf_Shared-Folder/Taxi/NYCTaxiAnalysis$ hdfs dfs -cat /user/pasan/nyctaxi_output/part-r-00000
2025-06-03 11:30:46,598 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Newark Airport (EWR)    605
Jamaica Bay (Queens)    26
Allerton/Pelham Gardens (Bronx) 32
Alphabet City (Manhattan)       34904
Arden Heights (Staten Island)   3
Arrochar/Fort Wadsworth (Staten Island) 39
Astoria (Queens)        22494
Astoria Park (Queens)   140
Auburndale (Queens)     41
Baisley Park (Queens)   1724
Bath Beach (Brooklyn)   58
Battery Park (Manhattan)        3021
Battery Park City (Manhattan)   95494
Bay Ridge (Brooklyn)    607
Bay Terrace/Fort Totten (Queens)        18
Bayside (Queens)        49
Bedford (Brooklyn)      4761
Bedford Park (Bronx)    142
Bellerose (Queens)      72
Belmont (Bronx) 108
Bensonhurst East (Brooklyn)     142
Bensonhurst West (Brooklyn)     164
Bloomfield/Emerson Hill (Staten Island) 9
Bloomingdale (Manhattan)        31505
Boerum Hill (Brooklyn)  11656
Borough Park (Brooklyn) 282
Breezy Point/Fort Tilden/Riis Beach (Queens)    3
Briarwood/Jamaica Hills (Queens)        823
Brighton Beach (Brooklyn)       88
Broad Channel (Queens)  3
Bronx Park (Bronx)      26
Bronxdale (Bronx)       49
Brooklyn Heights (Brooklyn)     11965
Brooklyn Navy Yard (Brooklyn)   305
Brownsville (Brooklyn)  348
Bushwick North (Brooklyn)       2938
Bushwick South (Brooklyn)       6140
Cambria Heights (Queens)        44
Canarsie (Brooklyn)     295
Carroll Gardens (Brooklyn)      7129
Central Harlem (Manhattan)      31578
```

Figure 11: HDFS Output Sample 1

```
Springfield Gardens South (Queens)      566
Spuyten Duyvil/Kingsbridge (Bronx)      298
Stapleton (Staten Island)       15
Starrett City (Brooklyn)        42
Steinway (Queens)       8494
Stuy Town/Peter Cooper Village (Manhattan)      35809
Stuyvesant Heights (Brooklyn)   2451
Sunnyside (Queens)      19493
Sunset Park East (Brooklyn)     200
Sunset Park West (Brooklyn)     1497
Sutton Place/Turtle Bay North (Manhattan)       211517
Times Sq/Theatre District (Manhattan)   366641
TriBeCa/Civic Center (Manhattan)        205379
Two Bridges/Seward Park (Manhattan)     22549
UN/Turtle Bay South (Manhattan) 142269
Union Sq (Manhattan)    365252
University Heights/Morris Heights (Bronx)       262
Upper East Side North (Manhattan)       390744
Upper East Side South (Manhattan)       411853
Upper West Side North (Manhattan)       200094
Upper West Side South (Manhattan)       271939
Van Cortlandt Park (Bronx)      40
Van Cortlandt Village (Bronx)   133
Van Nest/Morris Park (Bronx)    119
Washington Heights North (Manhattan)    2350
Washington Heights South (Manhattan)    9611
West Brighton (Staten Island)   4
West Chelsea/Hudson Yards (Manhattan)   153641
West Concourse (Bronx)  1590
West Farms/Bronx River (Bronx)  77
West Village (Manhattan)        240500
Westchester Village/Unionport (Bronx)   166
Westerleigh (Staten Island)     4
Whitestone (Queens)     99
Willets Point (Queens)  18
Williamsbridge/Olinville (Bronx)        59
Williamsburg (North Side) (Brooklyn)    17956
Williamsburg (South Side) (Brooklyn)    17535
Windsor Terrace (Brooklyn)      615
Woodhaven (Queens)      228
Woodlawn/Wakefield (Bronx)      142
Woodside (Queens)       7184
World Trade Center (Manhattan)  47595
Yorkville East (Manhattan)      133692
Yorkville West (Manhattan)      198801
N/A (Unknown)   169845
Outside of NYC (N/A)    6723
```

Figure 12: HDFS Output Sample 2

7 Results Interpretation & Insights

The MapReduce job successfully processed all 10,905,067 records from the January 2016 taxi trip dataset. The output provides a count of taxi pickups for 261 distinct taxi zones, which were then mapped to their respective names and boroughs.

7.1 Summary of Results:

The primary output is a list of taxi zones ranked by their total pickup counts. To view the ranked list, use the `get_top_n.py` script as described in the Quick Start (Step 6) or Section 4.c.

The Top 20 busiest pickup locations for January 2016 are:

1. Upper East Side South (Manhattan): 411,853 pickups
2. Midtown Center (Manhattan): 392,997 pickups
3. Upper East Side North (Manhattan): 390,744 pickups
4. Times Sq/Theatre District (Manhattan): 366,641 pickups
5. Union Sq (Manhattan): 365,252 pickups
6. Midtown East (Manhattan): 363,725 pickups
7. East Village (Manhattan): 361,127 pickups
8. Penn Station/Madison Sq West (Manhattan): 356,337 pickups
9. Murray Hill (Manhattan): 346,698 pickups
10. Clinton East (Manhattan): 337,406 pickups
11. Lincoln Square East (Manhattan): 304,218 pickups
12. Midtown North (Manhattan): 291,486 pickups
13. Gramercy (Manhattan): 279,824 pickups
14. Upper West Side South (Manhattan): 271,939 pickups
15. Midtown South (Manhattan): 263,255 pickups
16. Lenox Hill West (Manhattan): 263,139 pickups
17. LaGuardia Airport (Queens): 262,277 pickups
18. East Chelsea (Manhattan): 261,688 pickups
19. JFK Airport (Queens): 247,243 pickups
20. West Village (Manhattan): 240,500 pickups

Screenshot of the terminal output from the `get_top_n.py` script:

```
pasan@Ubuntu2:/media/sf_Shared-Folder/Taxi/DataInvestigation$ python3 get_top_n.py local_output.txt
Top 20 Busiest Pickup Locations:
1. Upper East Side South (Manhattan): 411853
2. Midtown Center (Manhattan): 392997
3. Upper East Side North (Manhattan): 390744
4. Times Sq/Theatre District (Manhattan): 366641
5. Union Sq (Manhattan): 365252
6. Midtown East (Manhattan): 363725
7. East Village (Manhattan): 361127
8. Penn Station/Madison Sq West (Manhattan): 356337
9. Murray Hill (Manhattan): 346698
10. Clinton East (Manhattan): 337406
11. Lincoln Square East (Manhattan): 304218
12. Midtown North (Manhattan): 291486
13. Gramercy (Manhattan): 279824
14. Upper West Side South (Manhattan): 271939
15. Midtown South (Manhattan): 263255
16. Lenox Hill West (Manhattan): 263139
17. LaGuardia Airport (Queens): 262277
18. East Chelsea (Manhattan): 261688
19. JFK Airport (Queens): 247243
20. West Village (Manhattan): 240500
```

Figure 13: Top 20 Results

7.2 Patterns and Insights Discovered:

- **Manhattan Dominance:** A significant majority of the busiest pickup locations are situated in Manhattan. This underscores Manhattan's role as the central business, entertainment, and residential hub of NYC, generating high taxi demand.
- **Key Hubs:** Areas like Midtown (Center, East, North, South), Upper East/West Sides, Times Square/Theatre District, and financial/transportation hubs like Penn Station consistently appear at the top. This is expected due to high population density, tourist activity, and commuter traffic.
- **Airport Traffic:** Both LaGuardia Airport and JFK Airport are prominent in the top 20, reflecting their importance as major transit points.
- **Skewed Distribution:** The pickup counts are heavily skewed. A relatively small number of zones account for a disproportionately large share of the total pickups, while many other zones have significantly lower activity.

7.3 Performance and Accuracy Observations:

- **Performance:**
  - The job efficiently processed ˜10.9 million records. The strategic use of a **Combiner** was vital for performance, significantly reducing data shuffled to reducers.
  - Reading from Parquet (a columnar format) is efficient for queries accessing a limited subset of columns.
  - The DistributedCache mechanism for the lookup table join is an efficient method for handling small auxiliary datasets in MapReduce.
- **Accuracy:**
  - The core MapReduce logic (map-combine-reduce for counting) is a standard and accurate approach for this aggregation task.
  - The join logic's accuracy relies on the `taxi_zone_lookup.csv`. The Reducer's robust CSV parsing ensures correct mapping of IDs to names.

- Hadoop counters such as `ReducerSetup -> ZoneLookupEntriesLoaded` (265) and `Reduce output records` (261) align with expectations for the dataset, indicating correct processing.
- The final successful run showed no significant error counts for critical operations, indicating high data integrity and correct processing.

8 Troubleshooting/Challenges Faced

Several challenges were encountered and overcome during the development of this project:

- **Reading Parquet in Java MapReduce:** This required careful management of Parquet-related dependencies in the `pom.xml` file and correct configuration of `ParquetInputFormat` with `GroupReadSupport` in the Hadoop job driver.
- **NullPointerExceptions in Mapper:** Early iterations faced NPEs when mappers attempted to access fields from Parquet `Group` objects. This was resolved by implementing robust `null` checks for the `Group` object itself at the beginning of the `map` method.
- **DistributedCache File Handling:** Ensuring the `taxi_zone_lookup.csv` was correctly added to the DistributedCache and then accessed properly within the Reducer's `setup()` method. The key was to use the local file name rather than its HDFS path.
- **CSV Parsing Robustness:** The initial CSV parsing logic for the lookup table was improved to be more robust against common issues, such as inconsistent quoting, leading/trailing whitespace, and empty fields.
- **Hadoop Environment Configuration:** Standard troubleshooting of a local Hadoop single-node setup, ensuring all necessary daemons were running correctly and that HDFS paths were accessible.

This project provides a practical demonstration of applying Hadoop MapReduce to analyze a significant volume of real-world data, successfully navigating common challenges in Big Data processing to extract meaningful and actionable insights.