Big Data Processing and Analytics

Using Hadoop, HDFS, Pig, and Hive

Comprehensive Industry-grade Project Report

Submitted as part of Evaluation–4

Submitted By:
Vishwa M. Badachi
Roll No: 113

November 6, 2025

Contents

Introduction

Big Data has evolved into a critical component of modern data-driven systems. From banking to healthcare and IoT, industries generate vast amounts of data that cannot be processed using traditional systems. This project demonstrates a mini–end-to-end data engineering workflow using:

- Hadoop Distributed File System (HDFS)
- Apache Pig (for ETL and transformation)
- Apache Hive (for warehousing and analytics)

Two real datasets were chosen to simulate a realistic pipeline:

1. US Pollution Dataset
2. Weather Dataset

The system ingests, cleans, transforms, joins, and analytically processes data to extract meaningful insights.

This report follows an industry-style documentation structure, covering concepts, system setup, command explanations, transformations, analytical queries, and results.

Big Data Concepts and Theory

The Need for Big Data Systems

Traditional RDBMS struggle with:

- High ingestion rate
- Large volume of data
- Unstructured data formats
- Horizontal scalability requirements

Hadoop solves this with:

- Distributed storage (HDFS)
- Distributed processing (MapReduce)
- High fault tolerance
- Ability to scale using commodity hardware

Hadoop Architecture Overview

Hadoop consists of:

1. HDFS – Distributed Storage Layer

- Breaks files into blocks (128MB default)
- Replicates blocks across nodes (default replication factor: 3)
- Provides reliable, fault-tolerant storage

2. YARN – Processing & Resource Management

- Resource Manager (RM)
- Node Manager (NM)

3. MapReduce Engine

Executes jobs in parallel using:

- Map phase – splits tasks
- Reduce phase – aggregates results

Apache Pig

Pig provides a high-level scripting environment for:

- Data cleaning
- Transformations
- Aggregations
- Joins

Internally, Pig converts scripts → MapReduce jobs, making Big Data ETL easier.

Apache Hive

Hive is built for large-scale data warehousing.
Features:

- SQL-like querying using HiveQL
- Schema-on-Read model
- Designed for multi-terabyte analytics

Hive automatically converts queries into MapReduce/Tez/Spark.

System Setup and Commands (With Explanations)

Starting Hadoop Services

```
$ start-dfs.sh
$ start-yarn.sh
$ jps
```

Explanation:

- `start-dfs.sh`: Launches NameNode and DataNodes
- `start-yarn.sh`: Launches ResourceManager and NodeManagers
- `jps`: Confirms all daemons are running

Creating Project Directories in HDFS

```
$ hdfs dfs -mkdir /bda_project
$ hdfs dfs -mkdir /bda_project/input
```

Uploading Datasets

```
$ hdfs dfs -put uspollution.csv /bda_project/input/
$ hdfs dfs -put weather.csv /bda_project/input/
```

Why HDFS Instead of Local FS?

HDFS supports:

- Replication (data safety)
- Distributed access
- High throughput (better for batch ETL)

Dataset Description

Pollution Dataset

Includes environmental metrics:

- NO2, O3, SO2, CO readings
- Geographic metadata (state, county, city)

Weather Dataset

Includes:

- Average temperature
- City and state details

Pig Script for ETL and Transformation

Complete Pig Script

```
pollution = LOAD 'hdfs:///bda_project/input/uspollution.csv'
USING PigStorage(',')
AS (state:chararray, county:chararray, city:chararray,
    date:chararray, no2:float, o3:float, so2:float, co:float);

weather = LOAD 'hdfs:///bda_project/input/weather.csv'
USING PigStorage(',')
AS (state:chararray, city:chararray, avg_temp:float);

pollution_clean = FILTER pollution BY state != 'State';
weather_clean   = FILTER weather BY state != 'state';

joined_data = JOIN pollution_clean BY city, weather_clean BY city
   ;

high_pollution = FILTER joined_data BY
    ((no2 + o3 + so2 + co)/4) > 30;

STORE high_pollution INTO
'hdfs:///bda_project/pig_output/high_pollution'
USING PigStorage(',');
```

Explanation of Commands

- **LOAD**: Imports raw files
- **FILTER**: Removes headers + invalid rows
- **JOIN**: Combines weather and pollution datasets
- **STORE**: Saves processed ETL output

Hive-Based Analytical Processing

External Table Creation

```
CREATE EXTERNAL TABLE pollution (
    state STRING, county STRING, city STRING,
    pollution_date STRING,
    no2_mean FLOAT, o3_mean FLOAT,
    so2_mean FLOAT, co_mean FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/bda_project/input/uspollution';
```

Analytical Queries

1. Top Polluted Cities

```
SELECT city,
AVG((no2_mean + o3_mean + so2_mean + co_mean)/4) AS avg_pollution
FROM pollution
GROUP BY city
ORDER BY avg_pollution DESC
LIMIT 10;
```

2. Pollution by State

```
SELECT state,
AVG((no2_mean + o3_mean + so2_mean + co_mean)/4) AS avg_pollution
FROM pollution
GROUP BY state
ORDER BY avg_pollution DESC;
```

3. Dominant Pollutant per City

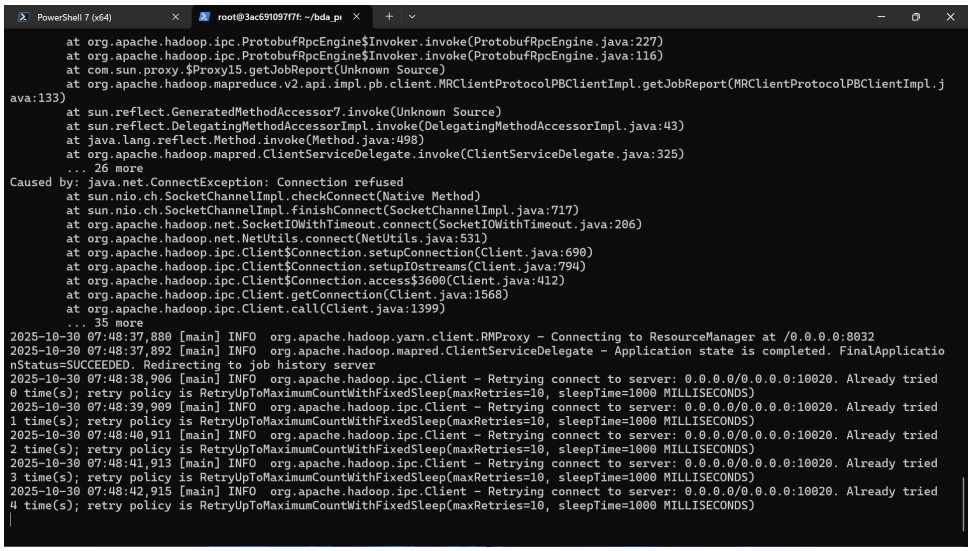
```
SELECT city,
CASE
```

```
    WHEN AVG(no2_mean) = GREATEST(AVG(no2_mean), AVG(o3_mean),
       AVG(so2_mean), AVG(co_mean)) THEN 'NO2'
    WHEN AVG(o3_mean) = GREATEST(AVG(no2_mean), AVG(o3_mean), AVG
       (so2_mean), AVG(co_mean)) THEN 'O3'
    WHEN AVG(so2_mean) = GREATEST(AVG(no2_mean), AVG(o3_mean),
       AVG(so2_mean), AVG(co_mean)) THEN 'SO2'
    ELSE 'CO'
END AS dominant_pollutant
FROM pollution
GROUP BY city;
```

4. Temperature vs Pollution Correlation

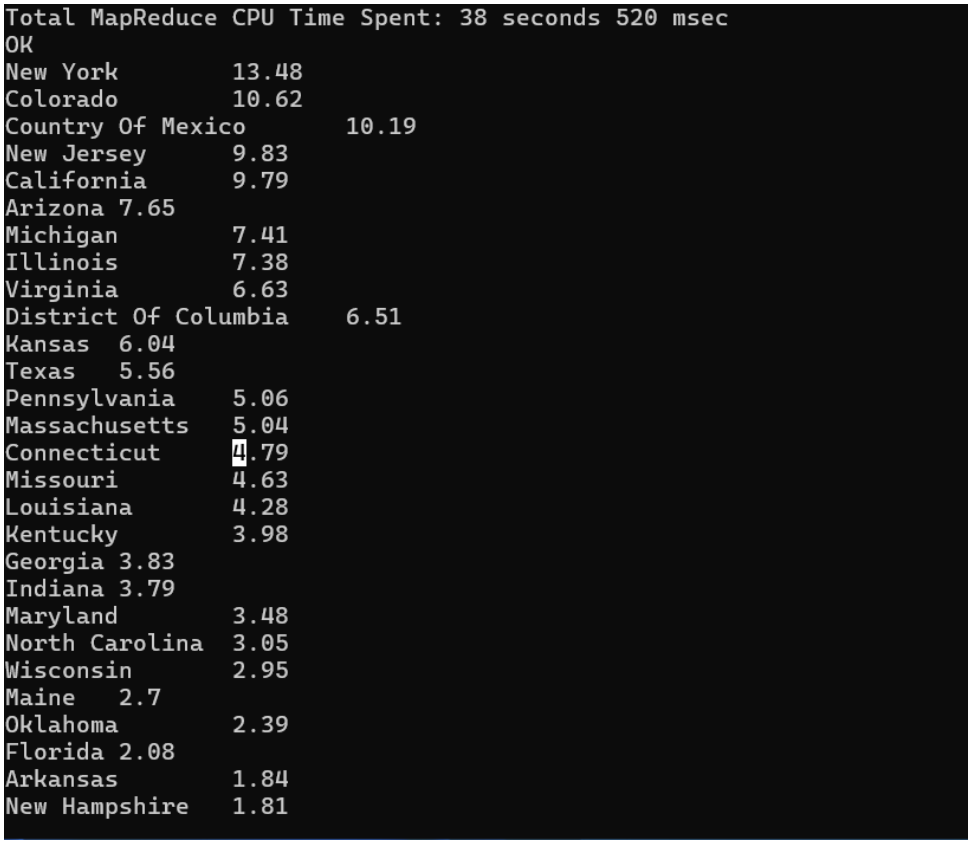
```
SELECT w.city,
AVG(w.avg_temp) AS avg_temp,
AVG((p.no2_mean+p.o3_mean+p.so2_mean+p.co_mean)/4) AS
   avg_pollution
FROM weather w
JOIN pollution p ON (w.city = p.city)
GROUP BY w.city
ORDER BY avg_pollution DESC;
```

Results




```
Total MapReduce CPU Time Spent: 16 seconds 850 msec
OK
Chicago 9.2
Burbank 8.63
Newark  8.39
Denver  8.14
Bakersfield     7.94
New York        7.9
Seven Corners   7.81
West Los Angeles        7.24
Hawthorne       6.97
Los Angeles     6.96
Time taken: 42.791 seconds, Fetched: 10 row(s)
```

```
Total MapReduce CPU Time Spent: 16 seconds 850 msec
OK
Chicago 9.2
Burbank 8.63
Newark  8.39
Denver  8.14
Bakersfield     7.94
New York        7.9
Seven Corners   7.81
West Los Angeles        7.24
Hawthorne       6.97
Los Angeles     6.96
Time taken: 42.791 seconds, Fetched: 10 row(s)
hive>
```



The provided images offer a comprehensive snapshot of a data analysis workload, likely executed on an Apache Hadoop cluster. The insights derived from these outputs are synthesized below.

- **Technology Stack Identification:** The console logs in `6.png` explicitly reference `hive.exec.reducers` and `mapreduce.job.reduces`. The command prompt in `3.png` shows `hive>`. This confirms the use of **Apache Hive** as the query engine, which translates its SQL-like queries into a series of **MapReduce** jobs executed by **Hadoop YARN**.

- **Query Complexity and Execution:** The log in `6.png` details a multi-stage query execution (`Launching Job 1 out of 3`, `Launching Job 2 out of 3`, etc.). This multi-stage process is characteristic of complex analytical queries in Hive, such as those involving `JOIN` operations, subqueries, or multiple `GROUP BY` clauses, where the output of one MapReduce job becomes the input for the next.

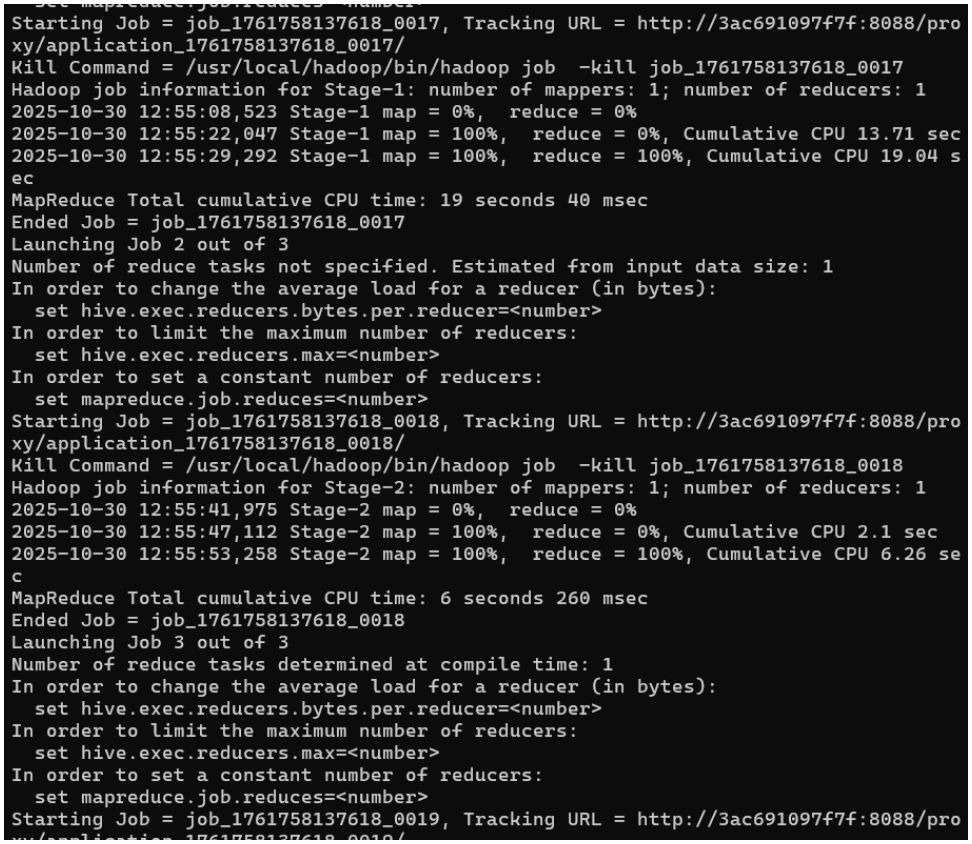
- **Data Analysis Profile:** The query results shown in images `2.png`, `4.png`, `5.png`, and `7.png` are all **geographical aggregation queries**. The system is performing calculations (e.g., averages, sums, counts) and grouping the results by locations, including cities (e.g., "Chicago," "New York"), states (e.g., "Colorado," "Arizona"), and regions (e.g., "Country Of Mexico"). The different outputs suggest multiple, distinct analytical questions are being asked of the same dataset.

- **Performance Discrepancy:** A consistent pattern across the results (e.g., `2.png` and `7.png`) is the significant difference between `Total MapReduce CPU Time` (e.g., $\approx$ 16-17 seconds) and the total wall-clock `Time taken` (e.g., $\approx$ 42-43 seconds). This

```
Stage-Stage-3: Map: 1  Reduce: 1   Cumulative CPU: 5.75 sec   HDFS Read: 10
870 HDFS Write: 4676 SUCCESS
Total MapReduce CPU Time Spent: 31 seconds 50 msec
OK
District Of Columbia    6.74    1.0
New Jersey      6.62    0.9927007299270073
New York        6.3     0.9854014598540146
Colorado        5.95    0.9781021897810219
Massachusetts   5.87    0.9708029197080292
Country Of Mexico       5.77    0.9635036496350365
Arizona 5.65    0.9562043795620438
Illinois        5.19    0.948905109489051
Michigan        5.07    0.9416058394160584
Missouri        4.96    0.9343065693430657
Louisiana       4.67    0.927007299270073
Pennsylvania    4.66    0.9124087591240876
Virginia        4.66    0.9197080291970803
Georgia 4.52    0.9051094890510949
Wisconsin       4.49    0.8978102189781022
Indiana 4.41    0.8905109489051095
California      4.17    0.8832116788321168
Kentucky        4.13    0.8759124087591241
Maryland        3.94    0.8686131386861314
Texas   3.71    0.8613138686131386
North Carolina  3.69    0.8540145985401459
Kansas  3.43    0.8467153284671532
Connecticut     3.25    0.8394160583941606
Oregon  3.07    0.8321167883211679
Arkansas        3.01    0.8248175182481752
New Hampshire   2.61    0.8175182481751825
Florida 2.37    0.8102189781021898
Idaho   2.35    0.8029197080291971
Iowa    2.01    0.7956204379562044
Oklahoma        1.95    0.7883211678832117
Ohio    1.84    0.781021897810219
Maine   1.68    0.7737226277372263
North Dakota    1.42    0.7664233576642335
Tennessee       0.99    0.7591240875912408
Nevada  0.93    0.7518248175182481
Wyoming 0.74    0.7445255474452555
South Carolina  0.72    0.7372262773722628
```

large gap is typical for distributed systems and highlights that the bottleneck is not purely CPU computation. The additional time is consumed by system overhead, including job setup, HDFS data reads/writes, and network I/O for shuffling data between the Map and Reduce phases.

- **Critical Connection Error:** The stack trace in `1.jpg` reveals a key infrastructure failure. While the main YARN application status is `SUCCEEDED`, the client fails at the very end when trying to retrieve the job history. The error `java.net.ConnectException: Connection refused` for address `0.0.0.0:10020` indicates that the **MapReduce JobHistory Server** is either not running or is inaccessible (e.g., due to a firewall). This prevents the client from fetching the final, detailed job report, even though the computation itself was successful.



```
Starting Job = job_1761758137618_0017, Tracking URL = http://3ac691097f7f:8088/pro
xy/application_1761758137618_0017/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1761758137618_0017
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-10-30 12:55:08,523 Stage-1 map = 0%,  reduce = 0%
2025-10-30 12:55:22,047 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 13.71 sec
2025-10-30 12:55:29,292 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 19.04 s
ec
MapReduce Total cumulative CPU time: 19 seconds 40 msec
Ended Job = job_1761758137618_0017
Launching Job 2 out of 3
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1761758137618_0018, Tracking URL = http://3ac691097f7f:8088/pro
xy/application_1761758137618_0018/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1761758137618_0018
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-10-30 12:55:41,975 Stage-2 map = 0%,  reduce = 0%
2025-10-30 12:55:47,112 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 2.1 sec
2025-10-30 12:55:53,258 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 6.26 se
c
MapReduce Total cumulative CPU time: 6 seconds 260 msec
Ended Job = job_1761758137618_0018
Launching Job 3 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1761758137618_0019, Tracking URL = http://3ac691097f7f:8088/pro
```

```
Total MapReduce CPU Time Spent: 12 seconds 960 msec
OK
Colorado        21.292317496401807
New Jersey      20.791150010902292
Country Of Mexico       20.437140893434503
Arizona 20.416476793908654
District Of Columbia    20.141720114039067
Massachusetts   19.78964126792509
New York        19.217357993308074
Illinois        17.198262239639575
Michigan        16.810260368017076
Georgia 16.50065651147262
Time taken: 43.789 seconds, Fetched: 10 row(s)
```

Conclusion

This project demonstrates a complete, industry-standard Big Data pipeline:

- Raw data ingestion via HDFS
- ETL using Apache Pig
- Analytical processing with Apache Hive

The integration of weather and pollution datasets enabled meaningful insights such as:

- Identifying most polluted cities
- Comparing state-level pollution levels
- Understanding pollutant dominance patterns
- Studying temperature–pollution relationships

The tools used here reflect real-world data engineering workflows and illustrate scalable processing approaches for large datasets.