Big Data Processing with MapReduce on a Hadoop Cluster on AWS

Name: Pruthvi Bharatbhai Dholkia

Class: DS644-006

NJIT UCID: pd469

Email: [pd469@njit.edu](mailto:pd469@njit.edu)

1. **Dataset Introduction**

* Name: Appliances dataset
* Source: [amazon: appliances reviews dataset](https://amazon-reviews-2023.github.io/)
* Size: 886 mb
* Format: JSONL  
  1. **Reason to select this dataset**
* The dataset consists of large-scale customer reviews from Amazon, making it an ideal real-world example for analyzing consumer behavior and product trends.
* It provides structured and unstructured data, allowing us to test MapReduce’s efficiency in handling and transforming JSON-based records.
* I can perform diverse analysis like

1. Product Popularity Analysis (Based on review count and engagement)
2. Rating Distribution (Customer satisfaction levels)
3. Helpful Reviews Identification (Finding the most influential reviews)
4. Verified vs Unverified Reviews (Measuring credibility and trustworthiness)

* Many companies like Amazon, Flipkart, and Walmart rely on big data analytics for customer sentiment analysis, fraud detection, and product ranking.
* The project mimics real-world data engineering tasks in retail and e-commerce analytics.

1. **Hadoop Cluster Setup with Screenshots**  
   1. **AWS EC2 Instances Running**

The screenshot displays the AWS EC2 Management Console, showing the four-node Hadoop cluster deployed for this project.

* Master Node: **t2.micro** instance managing the Hadoop cluster.
* Slave1 & Slave2: **t2.medium** instances for better load handling.
* Slave3**: t2.micro** instance.

Before making slave 1 & 2 medium, I went with t2.micro. Since the dataset is over 500MB and JSON parsing is memory-intensive, Slave1 and Slave2 were upgraded to t2.medium for better performance and even job distribution across the cluster. This helped avoid memory bottlenecks and ensured smoother execution.

A screenshot of a computer

AI-generated content may be incorrect.EC2 Instances

* 1. **Passwordless SSH from Master to Slave**

This screenshot shows a successful passwordless SSH login from the Master node (ubuntu@ip-172-31-17-204) to Slave1 using the command ssh slave1, ssh slave2, ssh slave3.

A screenshot of a computer

AI-generated content may be incorrect.Passwordless SSH is essential for **Hadoop daemons** to communicate seamlessly during distributed processing.  
 slave 1 slave2 A screenshot of a computer

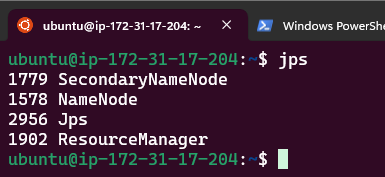
AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Slave3

* 1. **JPS output from Master and Slaves**

A black background with white text

AI-generated content may be incorrect.These screenshots below display the output of the jps command run on both the Master and Slave nodes, showing active Hadoop cluster.

Master Node slave1 Node

A close-up of a computer code

AI-generated content may be incorrect.A computer screen with numbers and a code

AI-generated content may be incorrect. Slave2 Node Slave3 Node

1. **MapReduce Code**

In this project, a total of **five** different MapReduce jobs were performed on the Amazon Appliances Review dataset to uncover various aspects of customer behavior and product insights, each focusing on a different analytical goal.

1. **Unique Products Count** – To determine how many different appliances were  
    reviewed.
2. **Rating Distribution Analysis** – To analyze overall customer satisfaction levels (1–5  
    stars).
3. **Top Helpful Reviews** – To identify products with the most helpful feedback.
4. **Verified vs. Unverified Analysis** – To assess the credibility of customer reviews.
5. **Product Popularity & Engagement** – To find the most discussed and highly-rated  
    appliances.

**Project Directory Structure Overview**

Before diving into the code, it's important to understand the organized folder structure used in this project. The Hadoop jobs, datasets, logs, and configuration files were carefully structured under the ~/big\_data\_project directory.

Each MapReduce job is stored in its own dedicated folder (e.g., job1\_unique\_products, job2\_rating\_distribution), containing the .java, .class, and .jar files.

Separate folders are maintained for:

* Hadoop installation (hadoop-2.6.5/)
* HDFS data (hadoop\_data/)
* Hadoop logs (hadoop\_logs/)

A screen shot of a computer program

AI-generated content may be incorrect.This structure not only keeps the environment clean and modular but also simplifies job execution commands and output management.

**A screenshot of a computer program

AI-generated content may be incorrect.** HDFS Structure

* 1. **Job 1: Unique Products Count**

Code Explanation: Map, Reduce, and Driver classes.

1. Mapper Class (ProductMapper)

Input: Each line from the JSONL file.  
Logic: Parses each line as JSON, extracts the asin (product ID).  
Output: (asin, 1) — emits each product ID with a count of 1.

1. Reducer Class (ProductReducer)

Input: All values associated with a single asin key.

Logic: Adds each asin key to a HashSet to ensure uniqueness.

Cleanup: After all keys are processed, it emits: ("Total unique products", totalCount)

where totalCount is the size of the HashSet.

1. Driver (main method)

Sets up the Hadoop job configuration.

Specifies the input/output paths and class types.

Runs the job using job.waitForCompletion(true).

A screenshot of a computer program

AI-generated content may be incorrect.Code snippet of job1 unique product count

**Compilation & Execution Commands**

**Step 1:** Upload the Java file to its job directory on the master node  
scp -i "C:\Users\pruth\Downloads\644midProject.pem" "C:\Users\pruth\Desktop\DS644\mid-project\jobs\UniqueProductsCount.java" ubuntu@54.167.79.219:/home/ubuntu/big\_data\_project/jobs/job1\_unique\_products/

**Step 2:** Navigate to the job directory  
cd ~/big\_data\_project/jobs/job1\_unique\_products/

**Step 3:** Compile the Java code  
javac -classpath `hadoop classpath` -d . UniqueProductsCount.java

**Step 4:** Create a JAR file  
jar -cvf UniqueProductsCount.jar UniqueProductsCount\*.class

**Step 5:** Verify the contents of the JAR  
jar -tf UniqueProductsCount.jar

**Step 6:** Remove the previous output directory in HDFS (if it exists)  
hdfs dfs -rm -r /user/ubuntu/output/job1\_unique\_products  
 **Step 7:** Run the MapReduce job on appliances dataset  
hadoop jar ~/big\_data\_project/jobs/job1\_unique\_products/UniqueProductsCount.jar UniqueProductsCount /user/ubuntu/input/Appliances.jsonl /user/ubuntu/output/job1\_unique\_products  
  
**Step 8:** View the job output  
hdfs dfs -cat /user/ubuntu/output/job1\_unique\_products/part-r-00000

* 1. **Job 2: Rating Distribution Analysis**

Code Explanation: Map, Reduce, and Driver classes.

1. Mapper Class (RatingMapper):

Parses each JSON line, extracts the rating (if between 1 and 5), and emits (rating, 1) for every valid review.

1. Reducer (RatingReducer)

Receives all values grouped by rating, sums them, and outputs the total count for each star rating.

1. Driver (main method)

A screen shot of a computer program

AI-generated content may be incorrect.Configures and runs the MapReduce job by setting classes, key-value types, and input/output paths.

Code of job 2 Rating Distribution Analysis

**Compilation & Execution Commands**

**Step1:** below command to upload java file to its job directory in master nodescp -i "C:\Users\pruth\Downloads\644midProject.pem" "C:\Users\pruth\Desktop\DS644\mid-project\jobs\RatingDistributionAnalysis.java" ubuntu@54.167.79.219:/home/ubuntu/big\_data\_project/jobs/job2\_rating\_distribution/

**Step 2:** Navigate to the job directory  
cd ~/big\_data\_project/jobs/job2\_rating\_distribution/

**Step 3:** Compile the Java code  
javac -classpath `hadoop classpath` -d . RatingDistributionAnalysis.java

**Step 4:** Create a JAR file  
jar -cvf RatingDistributionAnalysis.jar RatingDistributionAnalysis\*.class

**Step 5:** Verify the JAR contents:  
jar -tf RatingDistributionAnalysis.jar

**Step 6:** Check the output (Ensure previous output is removed if needed)  
hdfs dfs -rm -r /user/ubuntu/output/job2\_rating\_distribution

**Step 7:** run job  
hadoop jar ~/big\_data\_project/jobs/job2\_rating\_distribution/RatingDistributionAnalysis.jar RatingDistributionAnalysis /user/ubuntu/input/Appliances.jsonl /user/ubuntu/output/job2\_rating\_distribution

**step 8:** Check the Results After Execution  
hdfs dfs -cat /user/ubuntu/output/job2\_rating\_distribution/part-r-00000

* 1. **Job 3: Top Products by Helpful Reviews**

Code Explanation: Map, Reduce, and Driver classes.

1. Mapper Class (HelpfulVotesMapper):

Parses each JSON line, extracts the asin and helpful\_vote fields, and emits (asin, helpful\_vote) for reviews with more than 0 helpful votes.

1. Reducer (HelpfulVotesReducer):

Sums all helpful votes for each product and stores them in a TreeMap to identify and output the top 20 most helpful-reviewed products.

1. Driver (main method)

Sets up the MapReduce job by specifying the Mapper, Reducer, key/value types, and input-output paths.

* 1. **Job 4: Verified vs Unverified Reviews**

Code Explanation: Map, Reduce, and Driver classes.

1. Mapper Class (VerifiedMapper):

Parses each JSON review, checks the verified\_purchase field, and emits ("Verified", 1) or ("Unverified", 1) accordingly.

1. Reducer (VerifiedReducer):

Aggregates the counts for both keys ("Verified" and "Unverified") to compute how many reviews fall into each category.

1. Driver (main method):

Sets up the job with the appropriate classes, key-value types, and input-output paths, then executes it on the cluster.

Example Output Interpretation: If the output is ("Verified", 110000) and ("Unverified",  
 19500), it shows that out of all reviews, 110,000 are verified purchases (trusted), and  
 19,500 are unverified (potentially less reliable) offering a clear insight into review  
 credibility.

* 1. **Job 5: Product Popularity and Engagement**

Code Explanation: Map, Reduce, and Driver classes.

1. Mapper Class (PopularityMapper):

Parses each JSON review and emits (asin, [rating, helpful\_vote, 1]) for every product to track review count, helpful votes, and rating.

1. Reducer (PopularityReducer):

Aggregates data per product to compute total reviews, average rating, and total helpful votes, then selects the top 20 most engaging products using a priority queue.

1. Driver (main method):

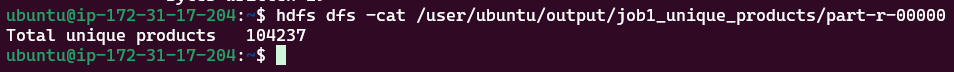
Sets the Mapper, Reducer, key-value types, input/output paths, and initiates the job execution.

Example Output Interpretation:An output like B01N0TQ0OH,4000,4.6,3500 indicates the product received 4,000 reviews, had an average rating of 4.6, and got 3,500 helpful votes, showing strong engagement and popularity.

1. **Code Execution & Output Interpretation**
   1. A screenshot of a computer program

      AI-generated content may be incorrect.**Job 1: Unique Products Count**

Running Code Command for Job 1 Unique Products Count

Output of Job 1 Unique Products Count

**Output Interpretation**

* The MapReduce job completed successfully, processing over **2.1 million review records** as indicated by Map input records=**2128605**.
* This result means there are **104,237 distinct products** (ASINs) reviewed in the dataset. This metric reflects the breadth and variety of appliances covered in the Amazon reviews and confirms the dataset's diversity.
  1. A screenshot of a computer

     AI-generated content may be incorrect.**A screenshot of a computer program

     AI-generated content may be incorrect.Job 2: Rating Distribution Analysis**

Running Code Command for Job 2 Rating Distribution Analysis

Output of JobA screen shot of a computer

AI-generated content may be incorrect. 2 Rating Distribution Analysis

**Output Interpretation**

* This output shows the number of reviews for each star rating (from 1 to 5), extracted and counted using MapReduce.
* Each line represents: <rating> <total\_reviews>
* 5-star reviews dominate the dataset with over 1.48 million entries, showing generally positive feedback.
* 1-star reviews are also relatively high (250K+), indicating a noticeable amount of dissatisfaction.

* 1. **Job 3: Top Products by Helpful Reviews** A screenshot of a computer screen

     AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Running Code Command for Job 3 Top Products by Helpful Reviews

A computer screen with white text

AI-generated content may be incorrect.

Output of Job 3 Top Products by Helpful Reviews

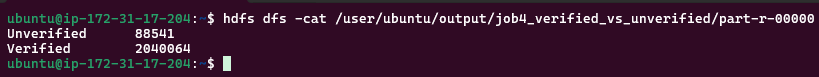
**Output Interpretation**

* This output lists the top 20 products based on the total number of helpful votes received on their reviews.
* Products like B01ALBMIEI and B07YF9SGBW stand out with 22,911 and 19,872 helpful votes, respectively, indicating high customer engagement.
* These results highlight which products had the most influential reviews, useful for identifying trusted and well-discussed items.
  1. A screenshot of a computer program

     AI-generated content may be incorrect.A screenshot of a computer

     AI-generated content may be incorrect.**Job 4: Verified vs Unverified Reviews**

Running Code Command for Job 4 Verified vs Unverified Reviews

Output of Job 4 Verified vs Unverified Reviews

**Output Interpretation**

* This MapReduce job analyzes and compares the number of Verified vs Unverified entries, likely from product reviews or user data.

Verified: 2,040,604 entries

Unverified: 88,541 entries

* This indicates that the majority of data points are verified.
* Insight: The data suggests high reliability, as most reviews or users are verified, enhancing the trustworthiness of the dataset.  
  1. A screenshot of a computer

     AI-generated content may be incorrect.**Job 5: Product Popularity and Engagement**

**A screenshot of a computer program

AI-generated content may be incorrect.**

Running Code Command for Job 5 Product Popularity and Engagement

A screenshot of a computer

AI-generated content may be incorrect.

Output of Job 5 Product Popularity and Engagement

**Output Interpretation**

* Job 5 evaluates product popularity and engagement by analyzing the number of reviews, average ratings, and total votes per product.
* Each line follows the structure:

<ProductID>, <NumberOfReviews>, <AverageRating>, <TotalVotes>

B01ALBM1EI, 4251, 4.0931, 22911

Product ID: B01ALBM1EI  
Number of Reviews: 4,251  
Average Rating: 4.09  
Total Votes: 22,911 (typically helpful/unhelpful votes)

* The job helps identify products with strong user engagement and positive reception. High review counts and vote totals reflect both popularity and active consumer interaction.

1. **Challenges & Troubleshooting**

**Challenges Faced**

* **Complex Error Tracing Across Slaves:**

During job execution, several errors occurred across different slave nodes, making it difficult to pinpoint issues. Without centralized logging initially, each node had to be manually checked for logs, increasing debugging time and complexity.

* **Initial Bottlenecks with t2.micro Instances:**

At first, all three slave nodes were configured as t2.micro, which lacked sufficient CPU and memory to process the nearly 1 GB JSON dataset. This caused severe bottlenecks, with YARN unable to allocate containers properly, leading to task failures, retries, and long job execution times.

* **Excess Hardware and Underutilization After Upgrading:**

To resolve the bottleneck, two slaves were upgraded to t2.medium while one remained t2.micro. However, this introduced a new issue: only the two medium-powered nodes were used by YARN for task execution, while the t2.micro instance remained mostly idle. The hardware was over-provisioned yet underutilized, defeating the purpose of having a third node.

* **Imbalanced Task Distribution by YARN:**

The observed task imbalance was due to YARN not distributing jobs fairly across all nodes. This was resolved by modifying YARN’s configuration, including enabling the fair scheduler, adjusting resource allocation limits, and tuning container memory settings to ensure equitable load distribution across all three nodes.

* **Malformed JSON Records Causing Mapper Failures:**

The dataset contained malformed JSON entries which triggered exceptions during parsing in the mapper class. These caused silent failures or skipped records until a proper try-catch mechanism and null checks were implemented to safely ignore bad input lines.

* **Hadoop Cluster Setup Difficulties:**Several issues were encountered during initial Hadoop setup:

1. Passwordless SSH setup failed due to missing key authorization and incorrect file permissions (chmod 600 ~/.ssh/authorized\_keys).
2. jps didn’t show expected daemons (e.g., DataNode, NodeManager) on some nodes due to misconfigured JAVA\_HOME or Hadoop paths.
3. Namenode startup failed until the HDFS directory paths were correctly specified and the namenode was formatted.
4. Hostname resolution problems occurred until /etc/hosts was properly updated across all nodes to support internal communication.

**Troubleshooting & Fixes**

* Implemented proper SSH configuration and verified using passwordless login before cluster start-up.
* Enabled Fair Scheduler:

To ensure balanced task distribution across all nodes, the FairScheduler was enabled in YARN. Initially, the job ran only on the two medium-powered nodes (Slave1 and Slave2), as they had sufficient resources—leaving Slave3 (t2.micro) idle. By enabling fair scheduling and adjusting resource allocation limits, YARN was forced to utilize all slaves more evenly, improving parallelism.

A screenshot of a computer program

AI-generated content may be incorrect.Changes made in yarn-site.xml (added these lines)

* JSON Parsing Failures & Skipping:

Performance was also impacted by malformed JSON entries that caused mapper crashes. This was mitigated by adding try-catch logic to safely skip bad records, reducing job failures and retries.

* Incorrect JAVA\_HOME and Hadoop Path Issues

On several nodes, jps failed to list Hadoop daemons due to incorrect or missing JAVA\_HOME and HADOOP\_HOME environment variables. This was fixed by updating .bashrc or .profile on all instances to export the correct paths, followed by source ~/.bashrc.

**Performance Observations**

* During early job runs, all three slave nodes were configured as t2.micro, which caused significant bottlenecks due to limited memory and CPU. As a result, jobs failed or ran very slowly, with YARN unable to allocate resources efficiently for container execution.
* Regularly used jps and YARN Web UI (port 8088) to monitor running daemons and job status across nodes.
* To simplify debugging across multiple nodes, **YARN log aggregation** was enabled to collect application logs into a centralized directory on the master node.
* add below line to do log aggregation in yarn-site.xml:

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.nodemanager.remote-app-log-dir</name>

<value>/home/ubuntu/hadoop\_logs</value>

</property>

<property>

<name>yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds</name>

<value>3600</value>

</property>

* Storage Setup for Logs:

mkdir -p /home/ubuntu/hadoop\_logs

sudo chown -R ubuntu:ubuntu /home/ubuntu/hadoop\_logs

Automatic Log Rotation Configured at /etc/logrotate.d/hadoop\_logs:

/home/ubuntu/hadoop\_logs/\* {  
 daily  
 rotate 7  
 compress  
 delaycompress  
 notifempty  
 create 640 ubuntu ubuntu  
}

* This setup helped monitor failed or underperforming jobs using:  
  logs/userlogs/  
  hadoop\_logs/ aggregated directory

1. **Summary & Key Learnings**

**Project Reflection**

This project provided hands-on experience with big data processing using Hadoop and MapReduce in a fully distributed environment. It deepened my understanding of cluster setup, job execution, debugging, and performance tuning. Working with real-world Amazon product review data taught me how to design and execute multiple MapReduce jobs to extract meaningful insights at scale.

**Real-World Application**

MapReduce is widely used in industry for processing large-scale datasets across multiple domains. Applications include log analysis, customer sentiment, fraud detection, and recommendation systems. In this project, MapReduce was applied to derive insights like product popularity, user engagement, and review credibility, which are directly relevant to e-commerce platforms and customer analytics teams.

**Future Improvements**

* Add Visualization Layer: Export output data into charts using tools like Tableau or Python Matplotlib for better insight presentation.
* Integrate Real-Time Data Streams: Extend the solution with Apache Kafka or Flume to process live review data in real time.
* Switch to Apache Spark: For faster in-memory processing and iterative computations.