Scalable Cloud-Based Sentiment Analysis System for Amazon Electronics Reviews

Shwe Moe Thant
National College of Ireland
Email: x24170399@student.ncirl.ie

I. Phase 1: Design and Setup

A. Problem Analysis and Use Case

The Amazon Electronics Reviews dataset, obtained from https://www.kaggle.com/datasets/zhipengluo89/electronics [1], contains millions of consumer reviews in JSON format. Each review includes attributes such as `overall` (rating, 1–5), `reviewText`, `reviewTime`, `reviewerID`, `asin`, `verified`, and others.


```json
{
  "overall": 5,
  "verified": true,
  "reviewTime": "07 17, 2002",
  "reviewerID": "A1N070NS9CJQ2I",
  "asin": "0060009810",
  "style": {"Format:": " Hardcover"},
  "reviewerName": "Teri Adams",
  "reviewText": "This was the first time I read Garcia-Aguilera... I closed the book with a feeling I had grown emotionally as well.",
  "summary": "Hit The Spot!",
  "unixReviewTime": 1026864000
}
```


Listing 1. Sample JSON review from the dataset

Given the dataset's 4 GB size and unstructured nature, tasks such as extracting sentiment polarity or identifying recurring terms (e.g., "battery" or "defective") require substantial computational resources. Sequential processing is inadequate for real-time e-commerce needs due to latency. Hence, a scalable cloud-based architecture is proposed to support parallel processing for sentiment analysis and keyword frequency detection.

Motivations for scalability:

- **Volume:** Efficiently handle millions of records using distributed computing.
- **Velocity:** Enable near real-time feedback analysis with low latency.
- **Scalability:** Dynamically scale compute resources using AWS services.
- **Resilience:** Ensure robustness and fault tolerance via distributed architecture.

This solution aligns with the core principles of cloud-native design, enabling rapid, reliable decision-making [2].

B. Data Ingestion Pipeline

To simplify processing, a 4 GB subset of the original 11 GB dataset was created using a Python script:

```python
import os

input_file = os.path.expanduser("~/Downloads/Electronics.json")
output_file = os.path.expanduser("~/Downloads/Electronics_4GB.json")
target_size = 4 * 1024 * 1024 * 1024  # 4 GB in bytes

with open(input_file, 'r', encoding='utf-8') as infile, \
     open(output_file, 'w', encoding='utf-8') as outfile:
    total_bytes = 0
    for line in infile:
        line_bytes = len(line.encode('utf-8'))
        if total_bytes + line_bytes > target_size:
            break
        outfile.write(line)
        total_bytes += line_bytes

print(f"Output file size: {total_bytes / (1024**3):.2f} GB")
```

Listing 2. Python script to extract 4 GB subset

The resulting subset was uploaded to Amazon S3 (s3://electronics-reviews-bucket/Electronics_4GB.json) for processing.

C. Distributed Ingestion Using PySpark on EMR

An EMR cluster (release emr-7.9.0, Spark 3.5) was provisioned via AWS Learner Lab with one master, two core, and one task node (all m5.xlarge, 4 vCPUs, 16 GB RAM). A bootstrap action ensured TextBlob was available for later sentiment tasks:

```bash
#!/bin/bash
sudo yum install -y python3 python3-pip
pip3 install textblob
python3 -m textblob.download_corpora
```

Listing 3. Bootstrap script for TextBlob installation

After SSH access and dependency setup (e.g., `python3`, `psutil`), the PySpark ingestion script was executed:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower
from pyspark.sql.types import StructType, StructField,
    StringType, DoubleType
import time, psutil, os
from pyspark.sql.utils import AnalysisException

def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)  # MB

spark = SparkSession.builder \
    .appName("ElectronicsDataIngestion") \
    .config("spark.hadoop.fs.s3a.endpoint", "s3.us-east-1.amazonaws.com") \
```

```python
    .config("spark.default.parallelism", "32") \
    .config("spark.executor.memory", "6g") \
    .config("spark.executor.cores", "4") \
    .config("spark.executor.instances", "4") \
    .config("spark.sql.files.maxPartitionBytes", "67108864") \
    .config("spark.dynamicAllocation.enabled", "true") \
    .config("spark.dynamicAllocation.minExecutors", "1") \
    .config("spark.dynamicAllocation.maxExecutors", "8") \
    .getOrCreate()

input_path = "s3://electronics-reviews-bucket/Electronics_4GB.json"
output_path = "s3://electronics-reviews-bucket/processed_data/"

start_time = time.time()
schema = StructType([
    StructField("asin", StringType(), True),
    StructField("overall", DoubleType(), True),
    StructField("reviewText", StringType(), True),
    StructField("reviewTime", StringType(), True)
])

try:
    df = spark.read.schema(schema).option("mode", "PERMISSIVE").json(input_path)
    df = df.repartition(32)
    record_count = df.count()
    df = df.select(
        col("asin"),
        col("overall"),
        col("reviewText"),
        col("reviewTime"),
        lower(col("reviewText")).alias("cleaned_reviewText")
    )
    df.write.mode("overwrite").parquet(output_path)

    print(f"Records processed: {record_count}")
    print(f"Execution time: {time.time() - start_time:.2f} seconds")
    print(f"Memory usage: {get_memory_usage():.2f} MB")
    print(f"Output saved to: {output_path}")

except AnalysisException as e:
    print(f"AnalysisException: {str(e)}")
except Exception as e:
    print(f"Error: {str(e)}")
finally:
    spark.stop()
```

Listing 4. PySpark script for scalable data ingestion

D. Performance Evaluation

Key performance metrics from the ingestion pipeline:

- **Records Processed:** 6,956,621
- **Execution Time:** 49.32 seconds
- **Throughput:** 141,037.83 records/second
- **Partitions:** 32 (parallel execution)
- **Memory Usage:** 51.28 MB (driver), 3.0 GB per executor

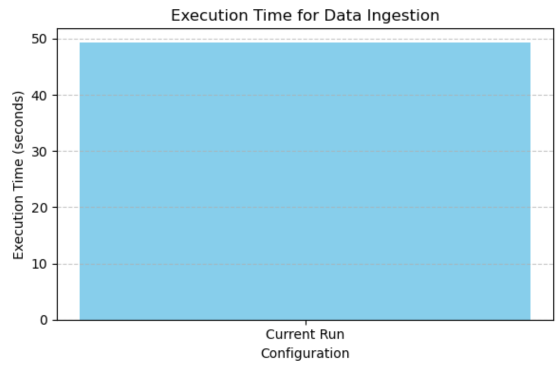

Fig. 1. Execution Time for Data Ingestion

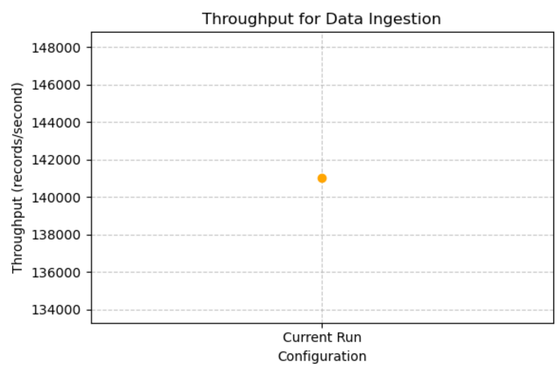

Fig. 2. Throughput during Ingestion

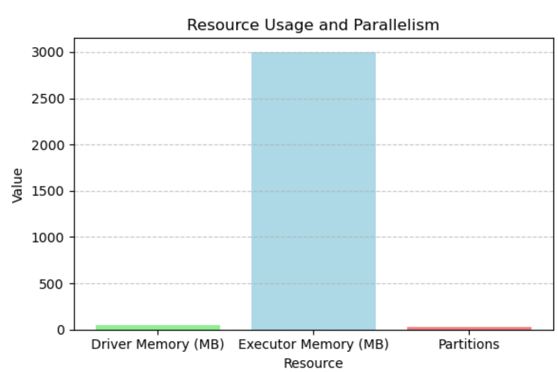

Fig. 3. Executor Resource Usage and Parallelism

References

[1] J. Ni, J. Li, and J. McAuley, "Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects," in *Proc. 2019 Conf. Empir. Methods Natural Lang. Process. (EMNLP-IJCNLP)*, Hong Kong, China, 2019, pp. 188–197.

[2] I. Foster and D. B. Gannon, *Cloud Computing for Science and Engineering*. MIT Press, 2017.

[3] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly Media, 2015.