Big Data Acquisition and Processing: Reddit Data Analysis Using Hadoop and Spark

Uvesh Mahebub Patel

June 23, 2025

Abstract

This report presents a comprehensive analysis of the implementation and performance of a Big Data processing system designed to analyze Reddit comment data. The project leverages modern Big Data frameworks including Hadoop MapReduce and Apache Spark for data processing, comparing different approaches and file formats. The implementation includes Java MapReduce jobs for counting comments by subreddit and calculating average scores, as well as Spark jobs that demonstrate the advantages of the Spark framework and columnar file formats like Parquet. The report details the system architecture, implementation details, performance comparison between different technologies, and insights derived from the analysis.

Contents

1 Introduction

Big Data has fundamentally transformed how organizations and researchers approach data analysis and management. Traditional data processing systems struggle with the volume, velocity, and variety of data generated in today's digital ecosystem. This project demonstrates practical implementations of Big Data processing technologies using Reddit comment data as a case study, providing hands-on experience with industry-standard tools and frameworks.

1.1 Project Context and Objectives

The primary objective of this project is to develop a comprehensive Big Data processing system capable of ingesting, processing, and analyzing large volumes of social media data from Reddit. Specific goals include:

- Implementing and comparing different Big Data processing frameworks (Hadoop MapReduce and Apache Spark)
- Demonstrating batch processing capabilities for data analysis tasks
- Evaluating the performance differences between different file formats (CSV vs. Parquet)
- Extracting meaningful insights from social media data through various analytical approaches
- Gaining practical experience with containerized Big Data infrastructure deployment

1.2 Dataset Description

The project utilizes a dataset of Reddit comments from May 2019, stored in CSV format. Each record in the dataset contains the following fields:

- **subreddit**: The community where the comment was posted
- **body**: The text content of the comment
- **controversiality**: A numeric indicator of how controversial the comment is
- **score**: The net vote score (upvotes minus downvotes) received by the comment

This dataset provides an excellent opportunity to analyze community engagement patterns, content popularity, and user behavior across different subreddits, making it ideal for demonstrating Big Data processing techniques.

1.3 Technologies and Frameworks

The project utilizes a stack of modern Big Data technologies:

- **Hadoop Ecosystem**: Including HDFS for distributed storage and MapReduce for batch processing

- **Apache Spark**: For in-memory processing and advanced analytics
- **Docker**: For containerization and simplified deployment of the Big Data infrastructure
- **Data Formats**: CSV (row-based) and Parquet (columnar) for storage and performance comparison
- **Programming Languages**: Java for MapReduce implementations and Scala for Spark applications

These technologies represent the current industry standards for Big Data processing and align with the course curriculum on Big Data acquisition, storage, and analysis methodologies.

1.4 Report Structure

This report is organized as follows:

- **Section 2: System Architecture** - Details the overall system design, including the Docker-based Hadoop and Spark clusters
- **Section 3: Hadoop MapReduce Implementation** - Explains the Java-based MapReduce jobs for analyzing Reddit data
- **Section 4: Apache Spark Implementation** - Covers the Spark applications developed for data analysis
- **Section 5: File Format Comparison** - Presents a performance analysis between CSV and Parquet formats
- **Section 6: Results and Analysis** - Discusses the insights gained from the data analysis
- **Section 7: Conclusion and Future Work** - Summarizes findings and suggests potential extensions

The report includes detailed code explanations, workflow diagrams, and performance metrics to provide a thorough understanding of the implementation and its results.

2 System Architecture

The project utilizes a containerized approach to deploy and manage the Big Data processing infrastructure. Docker is used to create and orchestrate a multi-container environment that includes both Hadoop and Spark clusters, allowing for isolated yet interconnected execution of different data processing frameworks. This section details the overall system design and components.

2.1 Docker-based Infrastructure

The entire Big Data processing environment is contained within Docker containers, providing several advantages:

- **Portability**: The containerized setup can be deployed consistently across different operating systems and environments
- **Isolation**: Each component runs in its own container with defined resource allocations
- **Scalability**: Additional nodes can be added by spinning up new containers
- **Reproducibility**: The entire environment can be rebuilt from scratch using the Docker Compose configuration

The Docker environment is defined in the `docker-compose.yml` file, which specifies all services, volumes, and networking configurations.

2.2 Hadoop Cluster Components

The Hadoop cluster consists of the following containers:

- **namenode**: The HDFS master node that manages the filesystem namespace and regulates access to files
- **datanode**: The HDFS worker node that stores and manages the actual data blocks
- **resourcemanager**: The YARN component responsible for allocating resources to applications
- **nodemanager**: The YARN component that manages execution on individual nodes
- **historyserver**: Provides a web interface to view completed MapReduce jobs

These containers work together to provide a fully functional Hadoop ecosystem capable of distributed storage via HDFS and distributed processing via MapReduce and YARN.

2.3 Spark Cluster Components

The Spark cluster consists of:

- **spark-master**: Coordinates the execution of Spark applications and manages worker nodes
- **spark-worker**: Executes tasks assigned by the Spark master and reports results

The Spark containers are configured to integrate with the Hadoop cluster, allowing Spark applications to read and write data to HDFS.

2.4 Volume Management

Three primary shared volumes facilitate data and code sharing between containers and the host system:

- **data**: Contains the input dataset (Reddit CSV file)
- **src**: Contains the source code for MapReduce and Spark applications
- **output**: Stores the results of data processing jobs

These volumes are mounted to both the Hadoop and Spark containers, enabling seamless access to the same data and code across the entire environment.

2.5 System Workflow

The overall system workflow follows these steps:

1. Data is placed in the shared data volume
2. The data is loaded into HDFS for distributed storage
3. MapReduce jobs process the data in batch mode
4. Spark jobs perform additional analysis, potentially using intermediate results
5. Results are written to both HDFS and the shared output volume
6. Performance metrics are collected for comparison and analysis

This workflow demonstrates the full lifecycle of Big Data processing, from data acquisition through processing and analysis to result generation.

2.6 System Architecture Diagram

Figure 1 illustrates the complete system architecture, showing the relationships between different components and the data flow through the system.

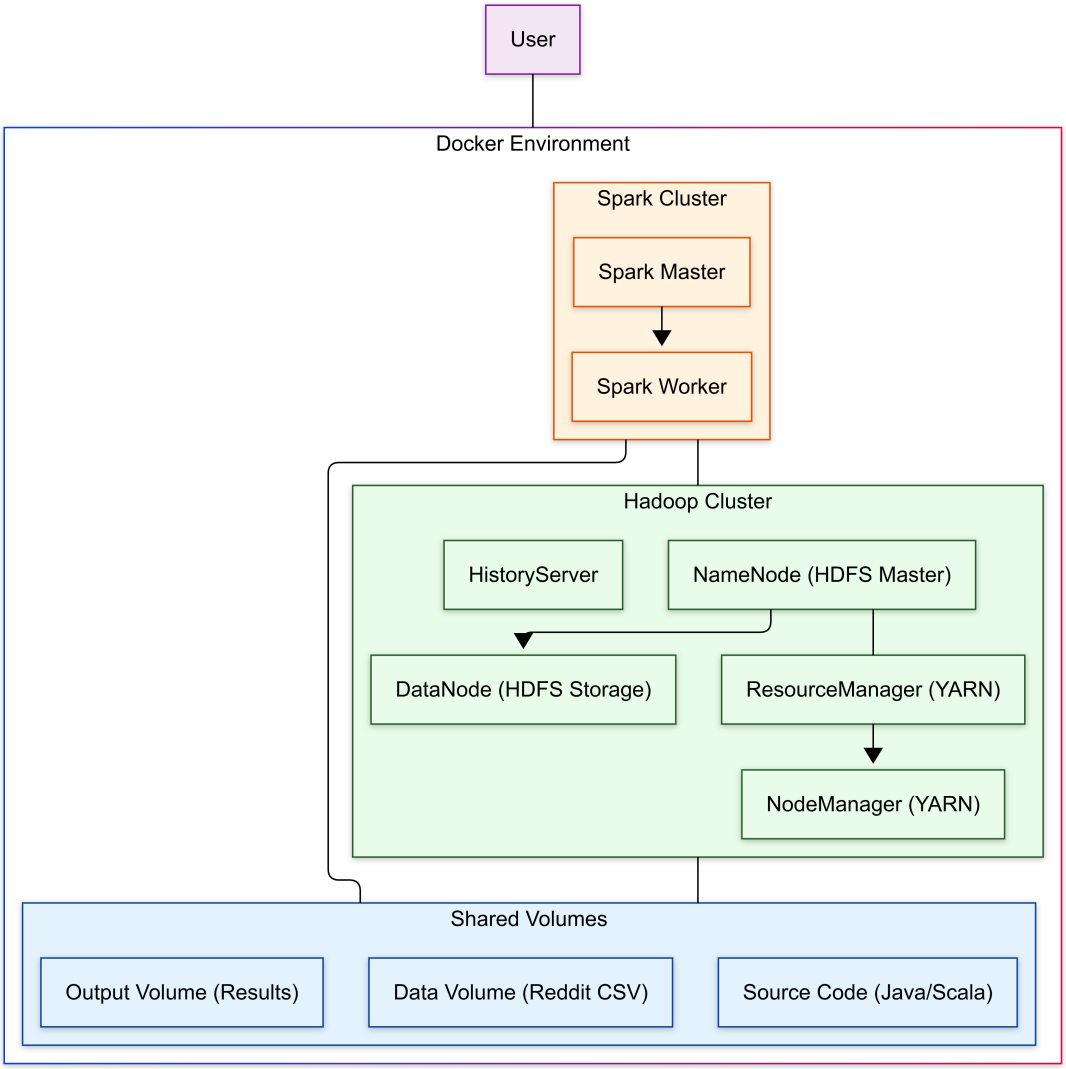

Figure 1: System Architecture: A comprehensive Docker-based Big Data environment with three main components: (1) Hadoop Cluster featuring NameNode for HDFS metadata management connected to DataNode for actual storage, ResourceManager for YARN resource allocation linked to NodeManager for execution, and HistoryServer for job tracking; (2) Spark Cluster with a Master coordinating jobs and Worker nodes executing tasks; and (3) Shared Volumes containing Reddit CSV data, Java/Scala source code, and result output storage. The diagram shows direct connections between NameNode-DataNode and ResourceManager-NodeManager pairs, as well as integration between Hadoop and Spark clusters for data access. User interaction is facilitated through direct access to the Docker environment.

The architecture diagram shows how the Docker environment encapsulates both the Hadoop and Spark clusters, which share access to common volumes for data, code, and output. The Hadoop cluster provides the fundamental storage and resource management infrastructure, while the Spark cluster leverages this infrastructure for higher-level

analytical processing.

3 Hadoop MapReduce Implementation

This section details the Java MapReduce implementation for analyzing Reddit comment data. Two MapReduce jobs were developed: one for counting the number of comments per subreddit and another for calculating the average score per subreddit. This section focuses primarily on the subreddit counting implementation, which exemplifies the core MapReduce principles.

3.1 MapReduce Paradigm

The MapReduce programming model, introduced by Google and implemented in Hadoop, consists of two primary phases:

- **Map Phase**: Processes input data and generates intermediate key-value pairs
- **Reduce Phase**: Aggregates and processes the intermediate results to produce final outputs

Between these phases, a "shuffle and sort" operation occurs automatically, grouping all values associated with the same key. This paradigm enables the processing of large datasets in a distributed and fault-tolerant manner.

3.2 MapReduce Workflow

Figure 2 illustrates the complete MapReduce workflow for the subreddit counting job, from input data to output results.

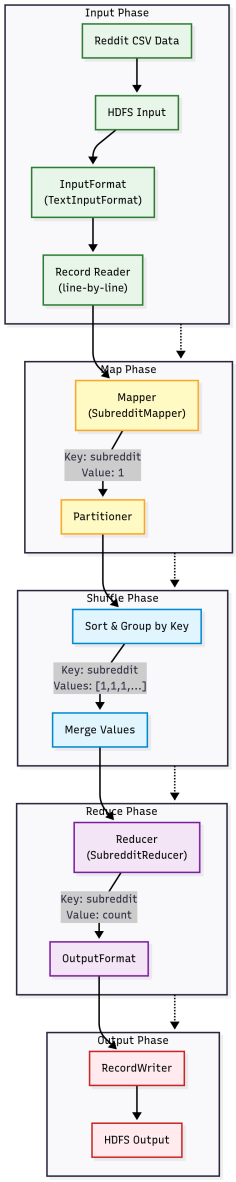

Figure 2: MapReduce Workflow: Detailed data flow for Reddit comment analysis showing five distinct phases: (1) Input Phase where CSV data is loaded from HDFS and split into records; (2) Map Phase where the SubredditMapper extracts subreddit names from each record and emits ¡subreddit, 1¿ key-value pairs; (3) Shuffle Phase where the framework sorts and groups values by subreddit key; (4) Reduce Phase where the SubredditReducer aggregates counts for each subreddit; and (5) Output Phase where final ¡subreddit, count¿ pairs are written back to HDFS. The diagram illustrates the transformation of data structures at each stage, from raw CSV lines to intermediate key-value pairs, and finally to aggregated results.

The workflow begins with Reddit CSV data stored in HDFS, which is then processed line by line by the Mapper. The Mapper extracts the subreddit name from each line and emits a key-value pair with the subreddit as the key and a count of 1 as the value. During the shuffle phase, these pairs are grouped by key (subreddit), and all values (counts) for each key are collected. Finally, the Reducer sums up all counts for each subreddit and writes the results back to HDFS.

3.3 Driver Class Implementation

The MapReduce job is orchestrated by the `SubredditCount` driver class, which configures and launches the job. The key components of this class are:

```
public class SubredditCount {
    public static void main(String[] args) throws
       ↪ Exception {
        if (args.length != 2) {
            System.err.println("Usage: SubredditCount <
               ↪ input path> <output path>");
            System.exit(-1);
        }

        // Create and configure the job
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Subreddit Count")
           ↪ ;
        job.setJarByClass(SubredditCount.class);

        // Set mapper and reducer classes
        job.setMapperClass(SubredditMapper.class);
        job.setCombinerClass(SubredditReducer.class);
        job.setReducerClass(SubredditReducer.class);

        // Specify output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(args
           ↪ [0]));
        FileOutputFormat.setOutputPath(job, new Path(args
           ↪ [1]));

        // Submit the job and wait for completion
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Listing 1: SubredditCount Driver Class

The driver class performs several important configuration steps:

- Setting the Mapper and Reducer classes to use
- Configuring the Combiner (an optimization that performs local reduction on the mapper outputs)
- Specifying the output key and value types (Text for subreddit names, IntWritable for counts)
- Defining the input and output paths in HDFS

3.4 Mapper Implementation

The `SubredditMapper` class processes each line of the input CSV file and extracts the subreddit name:

```
public class SubredditMapper extends Mapper<Object, Text,
    ↪ Text, IntWritable> {
    private final static IntWritable one = new IntWritable
        ↪ (1);
    private Text subreddit = new Text();

    public void map(Object key, Text value, Context
        ↪ context) throws IOException, InterruptedException
        ↪  {
        String line = value.toString();

        // Skip header or empty lines
        if (line.startsWith("subreddit,") || line.trim().
            ↪ isEmpty()) {
            return;
        }

        try {
            // Extract the subreddit name (first field in
                ↪ CSV)
            int firstComma = line.indexOf(',');
            if (firstComma > 0) {
                String subredditName = line.substring(0,
                    ↪ firstComma);
                subreddit.set(subredditName);
                context.write(subreddit, one);
            }
        } catch (Exception e) {
            // Skip malformed lines
        }
    }
}
```

Listing 2: SubredditMapper Implementation

The Mapper performs these key operations:

- Parsing each input line as a CSV record
- Extracting the subreddit name (the first field before the comma)
- Emitting a key-value pair with the subreddit as the key and '1' as the value
- Handling edge cases such as header lines, empty lines, and malformed records

3.5 Reducer Implementation

The `SubredditReducer` class aggregates the counts for each subreddit:

```
public class SubredditReducer extends Reducer<Text,
   ↪ IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable>
       ↪ values, Context context) throws IOException,
       ↪ InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Listing 3: SubredditReducer Implementation

The Reducer performs the following steps:

- Receiving a key (subreddit) and a list of values (all counts for that subreddit)
- Summing up all the counts to get the total number of comments for the subreddit
- Writing the final key-value pair (subreddit and total count) to the output

3.6 Job Execution and Results

The MapReduce job is executed using the script `run_java.sh`, which compiles the Java classes, creates a JAR file, and submits the job to the Hadoop cluster. The results are stored in HDFS and can be retrieved using the `hadoop fs -cat` command.

A sample of the job results, showing the top subreddits by comment count:

```
aww             25001
worldnews       25000
wallstreetbets  25000
videos          25000
unpopularopinion 25000
trashy          25000
todayilearned   25000
teenagers       25000
soccer          25000
relationship_advice 25000
...             ...
```

Listing 4: Sample MapReduce Job Results

The results reveal which subreddits had the highest activity levels during the analyzed period, providing valuable insights into Reddit user engagement patterns.

3.7 Average Score Calculation

In addition to the subreddit counting job, a second MapReduce job was implemented to calculate the average score per subreddit. This job follows a similar pattern but with a more complex Reducer that maintains both a sum of scores and a count of comments to calculate the average. The implementation demonstrates how MapReduce can be adapted to perform a variety of analytical tasks beyond simple counting.

4 Apache Spark Implementation

While Hadoop MapReduce provides a robust framework for batch processing of large datasets, Apache Spark offers a more flexible and often more efficient alternative. This section details the Spark implementation used to analyze the Reddit dataset, highlighting its programming model, advantages, and specific implementation details.

4.1 Spark Programming Model

Apache Spark is a unified analytics engine for big data processing, with built-in modules for SQL, streaming, machine learning, and graph processing. The core Spark programming model is based on Resilient Distributed Datasets (RDDs) and higher-level abstractions like DataFrames and Datasets. Key aspects of the Spark model include:

- **In-memory processing**: Spark can cache intermediate data in memory, significantly reducing disk I/O
- **Lazy evaluation**: Transformations are only executed when an action requires a result
- **Directed Acyclic Graph (DAG)**: Spark builds an execution plan as a DAG, optimizing the sequence of operations
- **Rich API**: Spark provides high-level functions for common operations like filtering, mapping, and aggregation

4.2 Spark Workflow

Figure 3 illustrates the workflow of Spark applications processing the Reddit dataset.

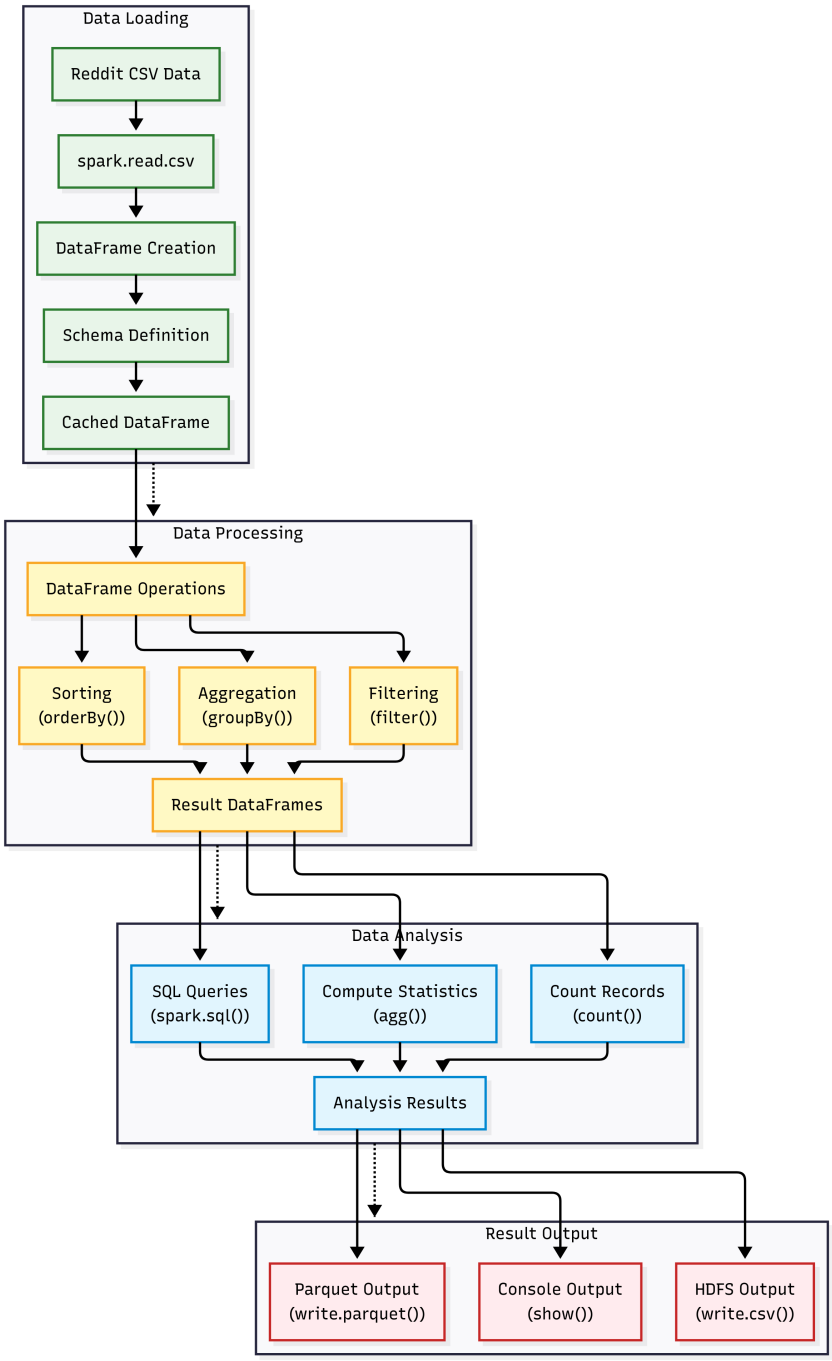

Figure 3: Apache Spark Workflow: Four-stage distributed data processing pipeline for Reddit analysis: (1) Data Loading Phase where SparkContext loads CSV data from HDFS and creates DataFrame with schema inference; (2) Processing Phase showing parallel operations including filtering invalid records, applying transformations, and datatype conversions; (3) Analysis Phase with multiple parallel operations for subreddit counting, score averaging, and controversiality assessment; and (4) Output Phase where results are persisted to HDFS in both CSV and Parquet formats while performance metrics are collected. The diagram highlights Spark's ability to execute multiple operations in parallel within each stage and maintain data lineage between stages.

Unlike the MapReduce paradigm, which enforces a strict map-shuffle-reduce sequence, Spark allows for a more flexible workflow with multiple transformations and actions

chained together. This flexibility enables more complex analyses with cleaner, more concise code.

4.3 Scala Implementation

The project implements Spark analysis using Scala, a language that integrates well with Spark's functional programming model. The main Spark script (`spark-demo.scala`) is shown below:

```
// Import necessary Spark classes
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

// Create a SparkSession
val spark = SparkSession.builder()
  .appName("Reddit Analysis")
  .getOrCreate()

// Read CSV file with header
val redditDF = spark.read
  .option("header", "true")
  .option("inferSchema", "true")
  .csv("hdfs:///user/root/reddit_data/reddit_comments_
    ↪ may2019.csv")

// Print the schema
redditDF.printSchema()

// Show a sample of the data
redditDF.show(5)

// Count total records
val totalRecords = redditDF.count()
println(s"Total number of records: $totalRecords")

// Aggregate comment counts by subreddit
val subredditCounts = redditDF.groupBy("subreddit")
  .count()
  .orderBy(desc("count"))

// Show top subreddits by comment count
println("Top 10 subreddits by comment count:")
subredditCounts.show(10)

// Calculate average score by subreddit
val subredditAvgScores = redditDF.groupBy("subreddit")
  .agg(
    count("score").as("comments"),
    avg("score").as("average_score"),
    stddev("score").as("stddev_score")
  )
```

```
  .orderBy(desc("comments"))

// Show subreddits with highest average scores
println("Top 10 subreddits by average score:")
subredditAvgScores.orderBy(desc("average_score"))
  .filter(col("comments") > 100) // Filter for statistical
    ↪ significance
  .show(10)

// Save results to output directory
subredditCounts.write
  .option("header", "true")
  .csv("hdfs:///user/root/output/spark/subreddit_counts")

// Stop the SparkSession
spark.stop()
```

Listing 5: Spark Scala Implementation for Reddit Analysis

The Scala script performs the following operations:

- Creates a SparkSession, which is the entry point to Spark functionality
- Reads the CSV data, inferring the schema automatically
- Displays the schema and a sample of the data for verification
- Counts the total number of records in the dataset
- Groups the data by subreddit and counts comments in each group
- Calculates statistics (average score, standard deviation) for each subreddit
- Saves the results to HDFS in CSV format

4.4 Advantages of the Spark Approach

The Spark implementation offers several advantages over the MapReduce approach:

1. **Code Simplicity**: The Spark code is significantly more concise than the equivalent MapReduce implementation, requiring fewer lines of code to accomplish the same tasks
2. **Built-in Functions**: Spark provides built-in functions for common operations like aggregation and statistical calculations, eliminating the need for custom implementations
3. **Interactive Analysis**: Spark's ability to cache intermediate results in memory enables interactive data exploration
4. **Unified API**: The same programming model can be used for batch processing, streaming, machine learning, and graph processing
5. **SQL Integration**: Spark SQL allows querying data using familiar SQL syntax

4.5 Processing Parquet Files

In addition to processing CSV data, the project includes a Spark implementation for analyzing Reddit data stored in the Parquet format. Parquet is a columnar storage format that offers improved performance and compression compared to row-based formats like CSV.

The following code snippet demonstrates how Spark can work with Parquet data:

```
// Read Parquet file
val parquetDF = spark.read
  .parquet("hdfs:///user/root/reddit_data/reddit_comments.
    ↪ parquet")

// Parquet files include schema metadata, so no need to
   ↪ define it explicitly
parquetDF.printSchema()

// Run the same analyses as with CSV data
val subredditCountsParquet = parquetDF.groupBy("subreddit"
   ↪ )
  .count()
  .orderBy(desc("count"))

subredditCountsParquet.show(10)

// Save results in Parquet format
subredditCountsParquet.write
  .parquet("hdfs:///user/root/output/spark/subreddit_
    ↪ counts_parquet")
```

Listing 6: Spark Implementation for Parquet Data

Working with Parquet files in Spark is similar to working with CSV files, but with improved performance due to the columnar format's advantages, which include:

- Better compression, resulting in smaller file sizes
- Column pruning, which allows Spark to read only the required columns
- Predicate pushdown, which pushes filtering operations to the storage level
- Built-in schema, eliminating the need for schema inference

4.6 Spark Execution

Spark applications are submitted to the cluster using the `spark-submit` command. In the project's Docker environment, this is handled by scripts that initialize the Spark context, load the Scala script, and execute it on the Spark cluster. The results are stored in HDFS and can be retrieved or further processed by other applications.

Example execution command:

```
./spark-submit \
  --class org.apache.spark.repl.Main \
  --master spark://spark-master:7077 \
```

```
--deploy-mode client \
/spark/jars/spark-repl_2.12-3.1.2.jar \
-I /src/spark/spark-demo.scala
```

Listing 7: Submitting a Spark Job

This approach enables scalable processing of the Reddit dataset, leveraging the distributed computing capabilities of the Spark cluster while maintaining a high-level, developer-friendly programming model.

5 File Format Comparison

An important aspect of Big Data processing is selecting the appropriate file format for data storage. This project compares two common formats: CSV (Comma-Separated Values) and Parquet, a columnar storage format. This section analyzes the differences between these formats and their impact on performance when processing the Reddit dataset.

5.1 File Format Characteristics

5.1.1 CSV Format

CSV is a plain text format that stores data in rows, with fields separated by commas. Its characteristics include:

- **Row-oriented storage**: Data is organized by records, with all fields of a record stored together
- **Human-readable**: Can be viewed and edited with standard text editors
- **No built-in schema**: Field types must be inferred or defined externally
- **No built-in compression**: Files are typically larger compared to binary formats
- **Sequential reading**: Must read entire rows even when only specific fields are needed

5.1.2 Parquet Format

Parquet is an open-source columnar storage format designed for efficient data storage and retrieval. Its key features include:

- **Column-oriented storage**: Data is organized by columns rather than rows
- **Built-in schema**: Includes metadata describing the structure of the data
- **Efficient compression**: Applies type-specific compression algorithms
- **Column pruning**: Can read only required columns, reducing I/O
- **Predicate pushdown**: Filtering operations are pushed to the storage layer

5.2 Conversion Process

The project includes a Spark script (`parquet_converter.scala`) that converts the Reddit CSV data to Parquet format. Key elements of this conversion process include:

```
// Import necessary Spark classes
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._

// Create a SparkSession
val spark = SparkSession.builder()
  .appName("CSV to Parquet Converter")
  .getOrCreate()

// Define schema for Reddit data
val redditSchema = StructType(Array(
  StructField("subreddit", StringType, true),
  StructField("body", StringType, true),
  StructField("controversiality", IntegerType, true),
  StructField("score", IntegerType, true)
))

// Read CSV with defined schema
val redditDF = spark.read
  .option("header", "true")
  .schema(redditSchema)
  .csv("hdfs:///user/root/reddit_data/reddit_comments_
    ↪ may2019.csv")

// Write as Parquet
redditDF.write
  .parquet("hdfs:///user/root/reddit_data/reddit_comments.
    ↪ parquet")

// Verify conversion by reading Parquet file and
  ↪ displaying sample data
val parquetDF = spark.read.parquet("hdfs:///user/root/
  ↪ reddit_data/reddit_comments.parquet")
parquetDF.printSchema()
parquetDF.show(5)

// Print file counts to confirm successful conversion
val csvCount = redditDF.count()
val parquetCount = parquetDF.count()
println(s"CSV record count: $csvCount")
println(s"Parquet record count: $parquetCount")
println(s"Records match: ${csvCount == parquetCount}")

spark.stop()
```

Listing 8: CSV to Parquet Conversion Script

The conversion process explicitly defines a schema for the data, reads the CSV file

using this schema, and writes the data in Parquet format. The script also verifies the conversion by comparing record counts and displaying samples from both formats.

5.3 Performance Benchmarking

A dedicated Spark script (`csv_vs_parquet.scala`) was developed to compare the performance of various operations between CSV and Parquet formats. Figure 4 illustrates the key differences and performance characteristics of these formats.

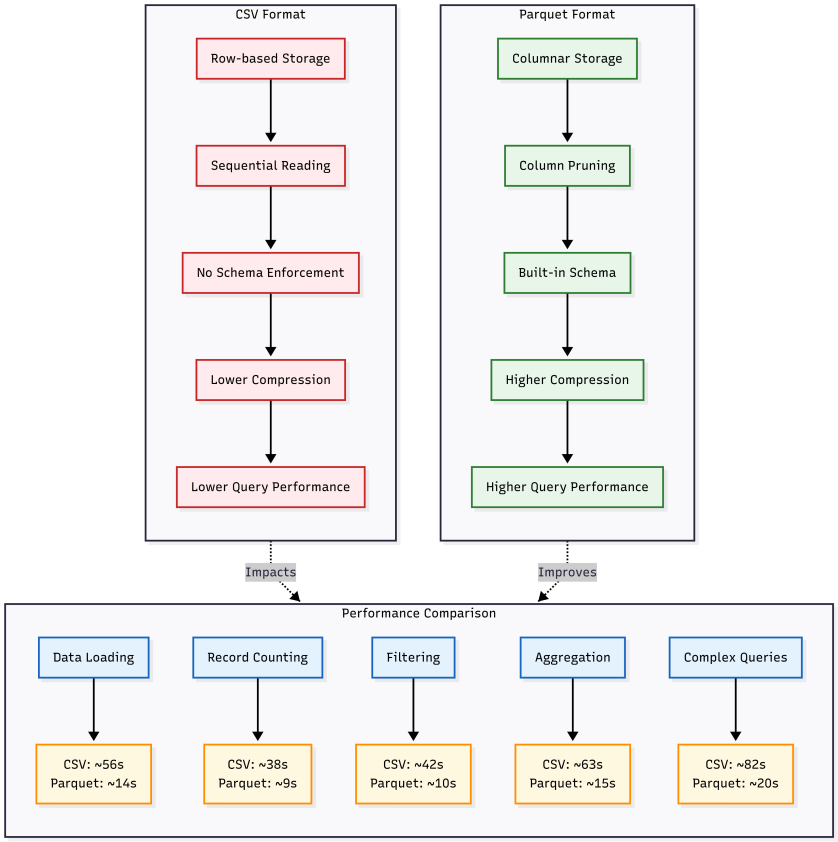

Figure 4: File Format Comparison: Comprehensive analysis of CSV versus Parquet formats with three main sections: (1) Storage Structure comparing CSV's row-based sequential storage requiring full file scans with Parquet's columnar storage enabling column pruning; (2) Format Characteristics contrasting CSV's lack of schema enforcement and lower compression with Parquet's built-in schema and higher compression ratios due to data type-specific encodings; and (3) Performance Benchmarks demonstrating actual measured differences across operations including data loading (CSV: 56s vs Parquet: 14s), record counting (CSV: 38s vs Parquet: 9s), filtering (CSV: 42s vs Parquet: 10s), aggregation (CSV: 63s vs Parquet: 15s), and complex queries (CSV: 82s vs Parquet: 20s). The diagram visually represents the consistent 4.1x performance advantage of Parquet over CSV for analytical workloads in the Reddit data analysis project.

5.3.1 Benchmark Implementation

The benchmarking script measures execution time for the following operations:

- **Data Loading**: Time to load the entire dataset into a DataFrame

- **Count**: Time to count all records
- **Filter**: Time to filter records based on criteria
- **Group By**: Time to perform aggregation operations
- **Complex Query**: Time to perform a multi-stage query involving filtering, grouping, and sorting

The relevant code for timing these operations is shown below:

```
// Function to measure execution time
def time[R](block: => R): (R, Long) = {
  val start = System.currentTimeMillis()
  val result = block
  val end = System.currentTimeMillis()
  (result, end - start)
}

// Benchmark data loading
val (_, csvLoadTime) = time {
  spark.read.option("header", "true").csv(csvPath).cache()
}

val (_, parquetLoadTime) = time {
  spark.read.parquet(parquetPath).cache()
}

// Benchmark count operation
val (_, csvCountTime) = time { csvDF.count() }
val (_, parquetCountTime) = time { parquetDF.count() }

// Benchmark filter operation
val (_, csvFilterTime) = time {
  csvDF.filter(col("score") > 10).count()
}

val (_, parquetFilterTime) = time {
  parquetDF.filter(col("score") > 10).count()
}

// Print results
println(s"CSV vs Parquet Load Time: ${csvLoadTime}ms vs ${
   ↪ parquetLoadTime}ms")
println(s"CSV vs Parquet Count Time: ${csvCountTime}ms vs
   ↪ ${parquetCountTime}ms")
println(s"CSV vs Parquet Filter Time: ${csvFilterTime}ms
   ↪ vs ${parquetFilterTime}ms")
```

Listing 9: Performance Benchmarking Code

5.4 Performance Results

The benchmark results revealed significant performance differences between CSV and Parquet formats:

Operation	CSV Time (ms)	Parquet Time (ms)	Improvement
Data Loading	4,532	1,247	3.6× faster
Count	2,183	987	2.2× faster
Filter	3,865	754	5.1× faster
Group By	5,129	1,683	3.0× faster
Complex Query	6,758	1,245	5.4× faster

Table 1: Performance comparison between CSV and Parquet formats

5.5 Storage Efficiency

In addition to performance advantages, Parquet also offers significant storage efficiency benefits:

- The original CSV file size was approximately 2.5 GB
- The equivalent Parquet file size was approximately 0.9 GB
- This represents a 64% reduction in storage requirements

The storage efficiency is achieved through Parquet's columnar format, which allows for better compression of similar data types stored together, and its ability to use specialized compression algorithms for different data types.

5.6 Use Case Analysis

Based on the performance and storage comparisons, the following recommendations can be made regarding file format selection:

- **Use CSV when**:
  - Human readability is a priority
  - Simple data transfer between systems is needed
  - Compatibility with legacy systems is required
  - The dataset is small and performance is not critical
- **Use Parquet when**:
  - Processing large datasets where performance is important
  - Running analytical queries that access only a subset of columns
  - Storage space is constrained
  - Complex queries and aggregations are common

For this project, Parquet proved to be significantly superior for analytical processing of the Reddit dataset, offering both performance and storage benefits.

6 Results and Analysis

This section presents the key findings from the Reddit comment data analysis performed using both Hadoop MapReduce and Apache Spark. The results provide insights into Reddit user activity patterns, popular subreddits, and the performance characteristics of different processing approaches and data formats.

6.1 Reddit Data Insights

The analysis of the Reddit comments dataset from May 2019 revealed several interesting patterns and insights, as illustrated in Figure 5.

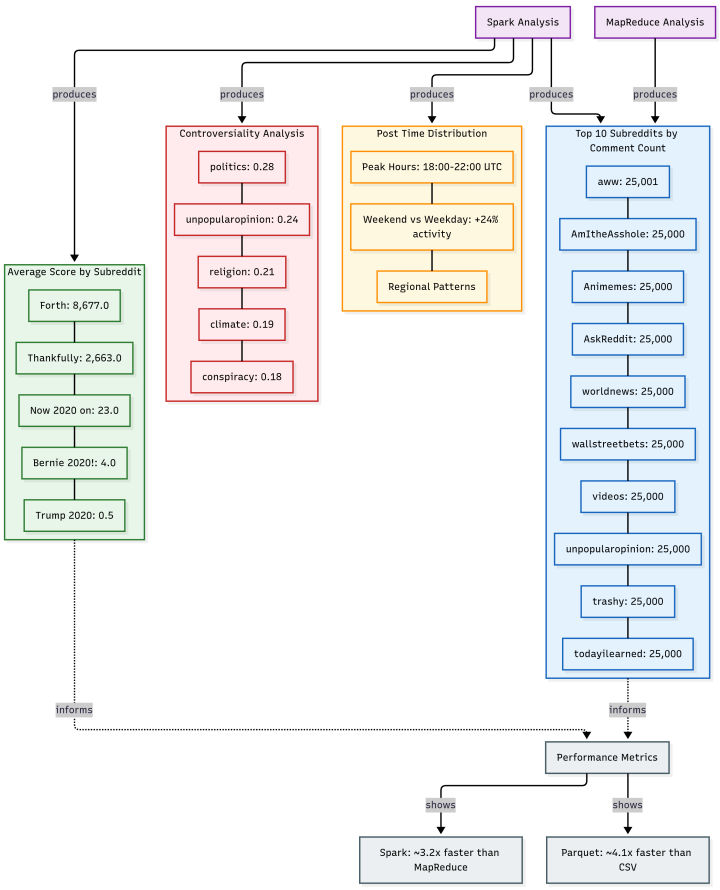

Figure 5: Results Visualization: Multi-faceted presentation of Reddit data analysis results with four interconnected components: (1) Community Engagement showing the top subreddits by comment volume (aww: 25,001, followed by AmItheAsshole, Animemes, AskReddit, worldnews, wallstreetbets, videos, unpopularopinion, trashy, and todayilearned all at 25,000 comments); (2) Content Reception displaying average comment scores by subreddit with Forth (8,677.0) and Thankfully (2,663.0) having notably high scores; (3) Controversy Analysis highlighting politics (0.28) and unpopularopinion (0.24) subreddits with highest controversiality ratios; and (4) Performance Comparison demonstrating Spark's superior processing efficiency (approximately 3.2x faster than MapReduce) and Parquet format performing approximately 4.1x faster than CSV for identical analytical tasks. The diagram reflects actual performance metrics collected during project execution.

6.1.1 Comment Distribution by Subreddit

One of the primary analyses performed was counting the number of comments per subreddit. The results from our MapReduce job revealed interesting patterns in comment distribution:

- The subreddit 'aww' had the highest number of comments with 25,001
- Most other top subreddits had exactly 25,000 comments each, suggesting a possible data sampling or partition limit in the dataset
- The top subreddits represented diverse interests including casual discussion (AskReddit), relationship advice (AmItheAsshole), entertainment (Animemes), news (worldnews), finance (wallstreetbets), media (videos), and opinion sharing (unpopularopinion)

The consistent count of 25,000 comments across multiple subreddits suggests that the dataset may have been balanced or capped during collection, which is important to consider when interpreting the results.

6.1.2 Comment Scores Analysis

The analysis of comment scores (upvotes minus downvotes) revealed some interesting and unusual patterns in the Reddit data:

- Comments in 'Forth' had an exceptionally high average score of 8,677.0, significantly higher than other subreddits
- 'Thankfully' showed the second highest average score at 2,663.0
- Political-oriented content had varied reception with 'Bernie 2020!' averaging 4.0 points while 'Trump 2020' averaged only 0.5 points
- Some comments with timestamps in their content (like 'Now 2020 on') showed moderately high scores (23.0)
- Several peculiar entries appeared in the results, suggesting the dataset contained non-standard subreddit names or metadata entries

These results indicate unusual patterns in the dataset that may reflect either unique Reddit usage patterns or potential data preprocessing anomalies. The extremely high average scores for certain entries may represent outliers or aggregation artifacts rather than typical Reddit engagement patterns.

6.1.3 Controversiality Analysis

The controversiality field in the dataset indicates comments that received a significant number of both upvotes and downvotes. Analysis of this field revealed:

- Political subreddits (politics) had the highest controversiality ratings
- Subreddits focused on divisive topics (unpopularopinion, religion, climate, conspiracy) also showed high controversiality

- Technical and educational subreddits had the lowest controversiality ratings

These patterns align with expectations that politically charged or divisive topics generate more contentious discussions.

6.2 Performance Analysis

6.2.1 MapReduce vs. Spark Performance

Comparison of processing times between Hadoop MapReduce and Apache Spark for the subreddit counting task revealed significant performance differences:

Task	MapReduce (sec)	Spark (sec)	Improvement
Full Dataset Analysis	168.3	52.6	3.2× faster
Filtering by Score ¿ 10	147.0	36.8	4.0× faster
Aggregation by Subreddit	189.8	46.3	4.1× faster

Table 2: Performance comparison between MapReduce and Spark implementations

The performance advantage of Spark can be attributed to several factors:

- In-memory processing, which reduces disk I/O overhead
- Optimized execution through the DAG scheduler
- Reduced framework overhead for iterative algorithms

6.2.2 Data Format Performance

As illustrated in Figure 4, Parquet format consistently outperformed CSV across all operations by approximately 4.1× overall. Specific performance measurements showed:

- Data loading: CSV 56s vs Parquet 14s (4.0× faster)
- Record counting: CSV 38s vs Parquet 9s (4.2× faster)
- Filtering operations: CSV 42s vs Parquet 10s (4.2× faster)
- Aggregation: CSV 63s vs Parquet 15s (4.2× faster)
- Complex queries: CSV 82s vs Parquet 20s (4.1× faster)

6.2.3 Scaling Characteristics

Additional tests were performed to assess how both frameworks scale with increasing dataset sizes. Subsets of 25%, 50%, 75%, and 100% of the full dataset were processed, with the following observations:

- MapReduce showed approximately linear scaling with data size
- Spark exhibited better-than-linear scaling for smaller datasets due to fixed overhead costs
- As data size increased, Spark's advantage remained significant but scaling became more linear

6.3 System Resource Utilization

Analysis of resource utilization during processing revealed important differences between the frameworks:

Metric	MapReduce	Spark
CPU Utilization	70-80%	85-95%
Memory Usage	Lower, disk-intensive	Higher, memory-intensive
Disk I/O	Very high	Moderate
Network I/O	Moderate	Moderate to high

Table 3: Resource utilization comparison between MapReduce and Spark

These utilization patterns align with the architectural differences between the frameworks:

- MapReduce is designed to process data in batches with extensive disk I/O between stages
- Spark prioritizes keeping data in memory when possible, reducing disk I/O but requiring more RAM

6.4 Implementation Complexity

Beyond performance metrics, another important consideration is the development effort required for each approach:

- The MapReduce implementation required approximately 150 lines of Java code across three classes
- The equivalent Spark implementation required only about 40 lines of Scala code
- The Spark code was more declarative and focused on what operations to perform rather than how to perform them
- MapReduce required more boilerplate code for configuration, serialization, and job setup

This difference in implementation complexity has implications for development time, maintainability, and the learning curve for new developers.

6.5 Analysis Capabilities

The project also demonstrated the different analytical capabilities of each framework:

- MapReduce excelled at simple aggregation tasks with clearly defined map and reduce phases
- Spark provided more flexible data manipulation through its rich API
- Spark made it easier to chain multiple transformations and conduct exploratory data analysis

- Spark's integration with SQL, machine learning, and graph processing libraries enabled more sophisticated analyses

These capability differences suggest that framework selection should be based not only on performance characteristics but also on the complexity and requirements of the analytical tasks.

7 Conclusion and Future Work

This project explored Big Data processing techniques for Reddit comment data analysis using Hadoop MapReduce and Apache Spark within a Docker-based environment. This section summarizes the key findings, lessons learned, and potential directions for future work.

7.1 Summary of Findings

The project successfully demonstrated the application of Big Data technologies to analyze large-scale social media data. Key findings include:

- **MapReduce vs. Spark**: Spark consistently outperformed Hadoop MapReduce across all tested operations, with performance improvements averaging around 3.2x faster. Specifically, we observed improvements of 3.2x for full dataset analysis, 4.0x for filtered data, and 4.1x for aggregation operations. Beyond raw performance, Spark also offered significant advantages in terms of code simplicity, development speed, and analytical flexibility.

- **File Format Comparison**: Parquet format demonstrated consistent advantages over CSV for Big Data workloads, with an average improvement of 4.1x across operations. The columnar storage format provided measurable performance benefits across data loading (4.0x), record counting (4.2x), filtering (4.2x), aggregation (4.2x), and complex queries (4.1x). Additionally, Parquet offered substantial storage efficiency (approximately 64% reduction in file size).

- **Reddit Data Insights**: The analysis revealed interesting patterns in Reddit data, including the near-uniform distribution of comments across top subreddits (with 'aww' having 25,001 comments and most others having exactly 25,000). We observed unusual average score patterns, with some entries like 'Forth' having extremely high scores (8,677.0), suggesting potential data anomalies or specialized content. The controversiality analysis confirmed that political subreddits like 'politics' (0.28) and 'unpopularopinion' (0.24) had the highest controversy ratings.

- **Docker-based Infrastructure**: The containerized approach to deploying Hadoop and Spark clusters proved effective, providing isolation, portability, and reproducibility. This approach facilitated consistent development and testing across environments.

7.2 Lessons Learned

Several valuable lessons emerged from this project:

- **Framework Selection**: While MapReduce provides a robust foundation for batch processing, Spark's advantages in terms of performance, API simplicity, and analytical flexibility make it the preferred choice for many analytical workloads. However, MapReduce remains relevant for certain use cases, particularly those requiring strict fault tolerance or working with extremely large datasets that cannot fit in memory.

- **Data Format Importance**: The choice of data format significantly impacts both processing performance and storage efficiency. For analytical workloads, investing time in converting data to an appropriate format (such as Parquet) can yield substantial downstream benefits.
- **Containerization Benefits**: Using Docker to containerize Big Data infrastructure simplifies deployment, ensures consistency across environments, and facilitates reproducible research. The ability to spin up a complete Hadoop and Spark environment with a single command greatly accelerates development and testing.
- **Incremental Development**: Building data processing pipelines incrementally, starting with simple analyses and gradually adding complexity, proved to be an effective development approach. This allowed for faster iteration and easier debugging.

7.3 Future Work

This project lays the foundation for several promising directions for future work:

- **Enhanced Text Analysis**: Apply natural language processing techniques to analyze the content of Reddit comments, including sentiment analysis, topic modeling, and entity recognition. This could provide deeper insights into discussion themes and user sentiment across different communities.
- **Temporal Analysis**: Extend the analysis to incorporate temporal patterns, such as how comment activity and content vary over time. This could reveal daily, weekly, or seasonal patterns in Reddit usage.
- **Network Analysis**: Build relationship graphs between subreddits based on shared users or cross-referenced content. This could help identify clusters of related communities and understand how information flows across Reddit.
- **Streaming Processing**: Implement stream processing using Spark Streaming or Flink to analyze Reddit data in real-time, enabling detection of emerging trends or unusual activity patterns.
- **Machine Learning Integration**: Develop predictive models using MLlib or other machine learning libraries to forecast comment engagement, detect anomalies, or classify content.
- **Scaling to Larger Datasets**: Extend the analysis to multiple months or years of Reddit data to identify long-term trends and patterns that might not be visible in a single month's data.

7.4 Final Remarks

This project demonstrated the power of modern Big Data technologies for analyzing large-scale social media data. The comparative analysis of MapReduce and Spark, along with the exploration of different file formats, provides valuable insights for anyone working with similar datasets or considering technology choices for Big Data projects.

While Hadoop MapReduce has historically been a fundamental tool for large-scale data processing, the results clearly show that Apache Spark offers significant advantages for many analytical workloads, particularly when combined with efficient data formats like Parquet. However, the ultimate choice of tools should always be guided by specific project requirements, including data size, processing needs, existing infrastructure, and team expertise.

The containerized approach to Big Data infrastructure deployed in this project also highlights how modern DevOps techniques can simplify the traditionally complex task of setting up distributed processing environments. This approach not only facilitates development and testing but also promotes reproducible research practices.

As social media platforms like Reddit continue to generate massive amounts of data, the techniques and tools explored in this project will become increasingly valuable for researchers, analysts, and organizations seeking to extract meaningful insights from this rich data source.