BIG DATA PROJECT

Report project

Fournier Pierre-Antoine 1820253067
Bichurina Sofia 1820253048
Dyoilis Anastasios 1820253075
Volkova Diana 1820253051

Contents

1 Introduction

In recent years, the exponential growth of data has profoundly reshaped the landscape of information technology, giving rise to the concept of *Big Data*. Characterized by enormous volume, velocity, and variety, Big Data has created unprecedented opportunities along with equally significant challenges. Traditional data-processing methodologies have increasingly proven insufficient, primarily due to their inherent limitations in computational power, scalability, and adaptability. This has necessitated the development of innovative technologies and frameworks specifically designed to handle large-scale datasets more efficiently and effectively.

Apache Spark, particularly its Python interface known as PySpark, represents one of the most impactful advancements in the domain of Big Data analytics. Spark provides a powerful distributed-computing framework capable of parallelizing tasks across multiple nodes or machines, greatly improving the speed and efficiency of processing large-scale datasets. PySpark, specifically, integrates the ease and readability of Python with the powerful computational capabilities of Spark, making it especially appealing and accessible to data scientists, analysts, software developers, and other professionals working extensively with data. Its straightforward syntax simplifies interactions with complex data structures, enabling rapid development and deployment of sophisticated data-processing workflows.

Despite the powerful features offered by PySpark, a noticeable usability gap still exists between advanced data analytics tools and the simplicity of traditional command-line interfaces familiar to Linux users. Linux-based command-line tools have long been praised for their straightforwardness, versatility, and ease of integration, enabling users to chain multiple commands seamlessly into complex data-processing pipelines. This project aims explicitly to bridge the gap between the simplicity and intuitiveness of the Linux command-line interface and the powerful data-processing capabilities of PySpark.

The core objective of this project is therefore the creation of command-line tools that behave similarly to standard Linux utilities but internally leverage PySpark's distributed-computing framework to process large datasets efficiently. Unlike traditional command-line utilities, which typically output textual or tabular data, the commands developed as part of this project will produce visual outputs, including graphs and geographical maps. Such visualizations are essential because they offer clear, intuitive representations of complex datasets, facilitating immediate insights into data trends, distributions, patterns, and relationships.

2 Tools, Technologies, and Environment

2.1 Apache Spark and PySpark

Apache Spark is a widely-used, open-source distributed computing framework specifically designed for efficient Big Data processing. Spark leverages in-memory computation and parallelism to provide significant performance advantages over traditional Big Data technologies, enabling it to process large volumes of data rapidly and efficiently across multiple nodes.

PySpark is the Python-based API built on top of Apache Spark, combining Spark's powerful distributed computing capabilities with the simplicity and flexibility of Python. PySpark allows developers and data scientists to write complex distributed computations and analytics tasks easily using Python's intuitive syntax, thus significantly reducing development complexity and enhancing productivity.

2.2 Key Advantages for Big Data Processing

PySpark and Apache Spark offer several critical advantages when working with Big Data:

- **Distributed Computing:** Spark can distribute data-processing tasks across multiple clusters or nodes, greatly speeding up computations and enabling efficient processing of massive datasets.
- **Scalability:** Spark easily scales from single-node applications to multi-node clusters, effectively adapting to growing dataset sizes and processing demands.
- **Performance:** Spark's ability to utilize memory caching and optimized task scheduling results in significantly faster processing compared to disk-based systems like Hadoop MapReduce.

2.3 Choice of PySpark over Alternative Solutions

Several alternative solutions exist for processing Big Data, including Hadoop MapReduce, Dask, and Apache Flink. Hadoop MapReduce, while robust for extremely large-scale data tasks, generally suffers from slow performance due to extensive disk usage. Apache Flink excels in stream processing scenarios, but its complexity is often unnecessary for datasets of moderate size. Dask, a parallel computing library in Python, provides ease of use but lacks the maturity and comprehensive integration provided by Spark's ecosystem.

We specifically selected PySpark for this project because our dataset size is medium-scale, making PySpark an ideal compromise in terms of performance, ease of use, and overall complexity. Additionally, PySpark's compatibility with standard Python libraries allowed us to conveniently use NumPy for

efficient and straightforward graph generation, simplifying our visualization workflow.

2.4 Visualization Libraries

For geographic data visualization, we utilized *Folium*, an interactive mapping library that simplifies geographic visualizations directly within Python. Additionally, we integrated *OpenRouteService* to handle routing, location mapping, and spatial analyses effectively. These tools enabled us to efficiently create insightful and interactive geographical visualizations directly from our processed datasets.

2.5 Hosting and Version Control

The project was hosted and managed on GitHub, leveraging Git for version control and collaboration. Using GitHub facilitated code sharing, easy collaboration among team members, issue tracking, and seamless integration into our continuous development and deployment workflow.

3 Implementation

3.1 Command-Line Interface

The project provides a command-line tool named `taxi`, which replicates typical Linux command-line syntax to simplify the usage for end-users. An example command is:

```
taxi pay 2009
```

This particular command generates a pie chart that visualizes the distribution of payment methods used for taxi trips during the specified year. The command currently supports three years: 2009, 2010, and 2011. If a user inputs a different year, the program gracefully handles the error by displaying a user-friendly error message informing the user about the supported options.

3.2 Shell Script and Python Integration

The tool uses a Bash script as the primary entry point, which internally calls Python scripts using the standard Python interpreter. Specifically, the Bash script invokes commands using the syntax:

```
python3 -m main pay <year>
```

This modular integration between the Bash shell and Python scripts provides ease of use, clarity, and maintains a straightforward workflow from command execution to data visualization.

3.3 Data Parsing and Querying with PySpark

Data for this project is stored in local `.parquet` files. A separate utility script included within the repository facilitates downloading these files onto the user's machine using the `wget` command. Once these files are downloaded and placed appropriately, a dedicated Python function parses the command-line arguments (such as the year provided by the user) and dynamically selects the appropriate `.parquet` dataset for further processing.

Using PySpark's powerful distributed processing capabilities, the tool performs data querying and transformations. The primary PySpark operations used include:

- **select**: To extract specific columns relevant to the visualization.
- **groupBy**: To aggregate data, particularly when generating charts like pie charts or histograms.

The processed data is then transformed into Python-native structures (e.g., lists or arrays) suitable for visualization tasks.

3.4 Data Visualization with NumPy and Folium

Visualization tasks within the project leverage Python libraries such as NumPy and Folium. Specifically, the following visualizations have been implemented:

- **Bar charts and Pie charts**: Visualize categorical data distributions clearly and effectively.
- **Histograms**: Illustrate frequency distributions for continuous variables like trip durations or fares.
- **Scatter plots**: Display relationships between pairs of variables (e.g., trip duration vs. fare amount).
- **Heat maps with Folium**: Create interactive geographical visualizations, showing spatial density or distributions of taxi trips clearly on a map.

The visualizations are generated using NumPy to structure and process data effectively, while plotting itself is typically done with Matplotlib (which integrates seamlessly with NumPy) and Folium (for geographical maps).

3.5 Geographical Visualization with OpenRouteService

For geographical visualizations involving specific taxi trip routes, the project utilizes a locally hosted instance of OpenRouteService. A local OpenRouteService server instance is launched separately to facilitate routing requests. The Python script communicates with this server through HTTP requests, retrieving routing information to plot taxi-trip polylines on interactive maps, which are then presented using Folium. This allows users to interactively explore trip routes within their web browsers directly from command outputs.

3.6 Memory Management and Batch Processing

A significant technical challenge encountered during the implementation was memory management, particularly when dealing with large datasets. This issue was resolved by adopting batch processing methods. The data was processed incrementally in smaller batches rather than all at once. This strategy effectively managed memory consumption, ensuring stable and efficient performance even when handling medium-to-large datasets.

This batching approach allowed the PySpark and visualization processes to operate smoothly, maintaining optimal resource usage without compromising the integrity or accuracy of the final results.

4 Results

The implementation of our PySpark-based Linux command-line tool successfully produced clear, informative, and visually intuitive outputs. The primary results obtained from the execution of various commands, particularly the example command `taxi pay 2009`, included visualizations such as pie charts, bar charts, histograms, scatter plots, and geographical heat maps. For example, executing:

```
taxi pay 2009
```

generates a pie chart clearly illustrating the distribution of different payment methods used for taxi trips in the year 2009. This visualization immediately provides users with insight into the prevalence of cash, credit card, and other payment methods, enabling intuitive analysis and interpretation of trends.

Moreover, our geographical visualizations leveraging *Folium* and *OpenRouteService* provided interactive maps displaying taxi trip density and specific trip routes, significantly enhancing the interpretability and utility of geographic data. Users can now visually identify high-traffic regions and frequent travel paths in a user-friendly and interactive manner.

Batch processing resolved earlier memory issues effectively, allowing for smooth execution and visualization even with moderately large datasets. Overall, the results clearly demonstrate PySpark's capability to handle complex data transformations efficiently, while the visualization techniques employed significantly aid in practical data interpretation.

5 Conclusion

This project effectively demonstrated the integration of PySpark's distributed computing power into an intuitive, familiar Linux-style command-line environment. By developing custom commands that internally utilize PySpark for data processing, the tool has successfully simplified complex analytical tasks, providing end-users with immediate, meaningful visual outputs such as pie charts, histograms, scatter plots, and interactive geographical maps.

PySpark proved to be the ideal framework for our medium-scale dataset, offering substantial performance and scalability benefits without undue complexity. Leveraging NumPy for straightforward visualizations and Folium combined with OpenRouteService for interactive mapping further streamlined the user experience and ensured that visual outputs were both clear and insightful.

The main technical challenges encountered, notably memory management, were effectively addressed by introducing batch processing techniques. This solution not only resolved immediate issues but also enhanced the scalability and robustness of the overall approach.

In conclusion, the successful completion of this project validates PySpark's effectiveness in processing medium-sized datasets and underscores the practical advantages of embedding Big Data analytics capabilities within simple command-line tools. This approach facilitates easy access to powerful analytics and insightful visualizations, laying a solid foundation for further enhancements and broader applications in data-driven environments.