Cloud-Based Distributed Data Processing Service

عبد الرحمن جهاد سمير الدشت (120221452), محمد أحمد فايز قشطة (120221445), جمال فرج مصطفى الصرفندي (120221079)

Faculty of Information Technology
The Islamic University of Gaza

A Requirement for the Course: Cloud and Distributed Systems (SICT 4313)
Instructor: Dr. Rebhi S. Baraka

Abstract

This project offers a cloud-based distributed data processing service implemented using Apache Spark.

The system analyzes a dataset of taxi trips and performs distributed data processing and machine learning tasks in a cloud-simulated environment using Google Colab.

Due to resource limitations on a single machine, Spark was run in local [N] mode to simulate a distributed cluster. Experiments were conducted using N = 1, 2, 4, and 8 simulation nodes to evaluate execution time, speed, and efficiency.

1. Introduction

This project focuses on the practical application of distributed processing concepts using Apache Spark, by performing a series of analytical tasks on taxi trip data.

The primary objective of this work is to evaluate Spark's performance when distributing computational workloads, rather than focusing solely on the analytical results themselves.

Due to limited resources on a single machine, Spark was executed in local[N] mode to simulate a virtual cluster environment, where changing the value of N represents different numbers of workers running in parallel on the same machine.

This setup allows for comparing execution times and observing system behavior as the number of workers increases, providing insight into Spark's scalability under constrained resources.

The implementation includes computing basic descriptive statistics, performing a Map/Reduce aggregation using Spark's grouping operations, and applying several machine learning algorithms using Spark MLlib.

The experimental results are analyzed to study the effects of parallelism and system overhead on performance, particularly in a simulated single-machine environment

2. Cloud Program/Service Requirements

The developed system offers a cloud-based data processing service that analyzes large-scale taxi trip data using Apache Spark[1]. The system is designed to automatically receive and process user-uploaded CSV datasets without requiring any manual configuration. Once the data is uploaded, the service handles data loading, pre-processing, analysis, and clear presentation of results.

The system is required to perform several fundamental operations that reflect the concepts of distributed systems. These operations include calculating basic descriptive statistics such as count, mean, and minimum and maximum values for selected attributes, as well as performing a Map/Reduce aggregation task using Spark's grouping operations.

In addition, the system applies multiple machine learning algorithms using Spark MLlib, including clustering, regression models, and pattern extraction, to demonstrate the application of distributed machine learning techniques on a real-world dataset.

To analyze scalability behavior, the same data processing workload is executed using different levels of parallelism. The system runs the Map/Reduce task multiple times using 1, 2, 4, and 8 workers, and records execution time, speedup, and efficiency for each configuration. This enables the evaluation of how system performance changes as the number of workers increases under limited hardware resources.

At a higher level of abstraction, the system simulates a distributed computing environment by running Spark in local[N] mode on a single machine. Each value of N represents a different number of virtual workers operating in parallel threads. Although execution takes place on a single machine, this abstraction allows the study of parallel execution behavior, system overhead, and resource contention, which are key characteristics of real distributed systems

3. Architecture and Design

The system architecture is designed to support distributed data processing and machine learning analytics in a cloud environment using Apache Spark.

The system follows a pipeline-based architecture, starting with user data input and ending with analytical and performance results. The user interacts with the system through a cloud interface provided by Google Colab[3], where a CSV dataset containing taxi trip logs is uploaded. Once the dataset becomes available, the system automatically starts the processing workflow without requiring additional user configuration.

Apache Spark represents the core processing component in this architecture. To simulate a distributed cluster environment under limited hardware resources, Spark is executed in local [N] mode. Each value of N represents a different number of virtual workers operating in parallel on the same physical machine.

From a data processing perspective, the architecture relies on Spark's distributed data abstractions to perform analytical tasks. The system supports descriptive data analysis, MapReduce-style data aggregation, and multiple machine learning workloads. The

Map/Reduce component conceptually distributes data across virtual workers during the mapping phase and then combines partial results during the reducing phase to produce aggregated outputs. This approach reflects the fundamental principles of distributed data processing systems.

The architecture also includes a machine learning layer built on top of the Spark MLlib library. This layer is responsible for executing clustering, regression, and pattern extraction tasks on the dataset. Architecturally, these machine learning tasks are treated as independent analytical modules that operate on distributed data and produce model outputs and evaluation metrics. The results of these tasks are collected and prepared for presentation to the user.

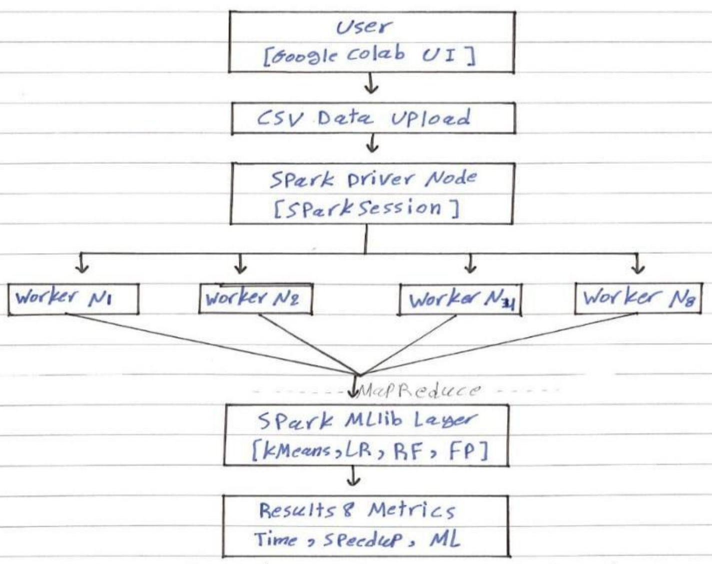
Regarding data storage, the system architecture is designed to process the dataset directly within the Spark environment without relying on an external database system. This design choice simplifies the system and is suitable for a simulated cloud-based setup. However, from an architectural point of view, a NoSQL database (such as MongoDB or Cassandra) could be integrated in a real-world deployment to store raw datasets, intermediate processing results, or machine learning outputs, providing persistent and scalable storage capabilities.

Finally, the architecture includes a results and reporting component responsible for presenting analytical outputs and performance metrics to the user. Execution time, speedup, and efficiency are systematically recorded for different values of N, enabling scalability analysis. All results are generated and accessed within the cloud execution environment, ensuring that the system satisfies cloud-based processing and storage requirements at the architectural level.

Overall, the architectural design emphasizes clarity, modularity, and alignment with distributed systems concepts. Although the system is executed on a single machine, the proposed architecture reflects how the same processing pipeline could be deployed on a real-world distributed cluster with minimal modifications.

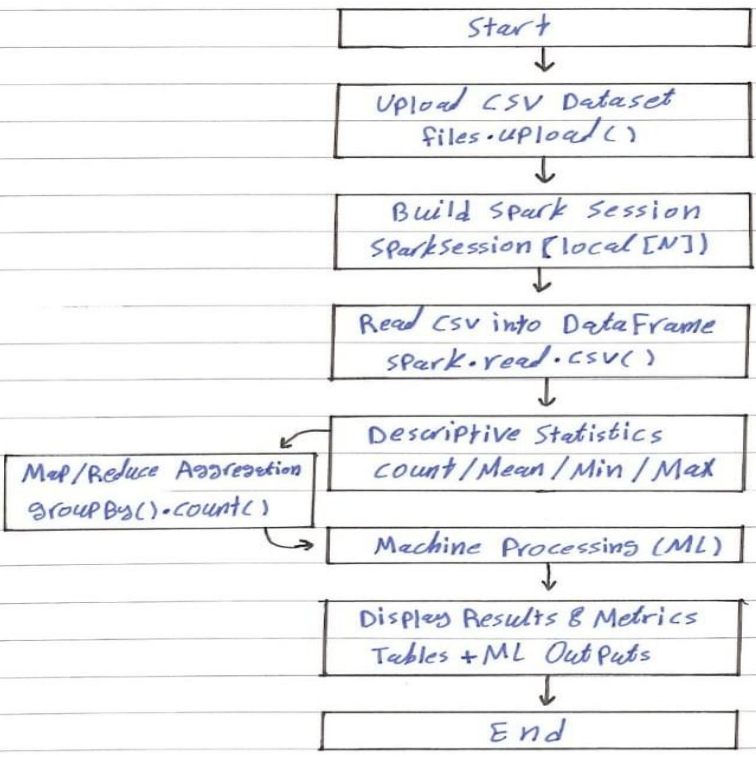
Here are diagrams to illustrate the system's mechanism:

1) Architecture Diagram:-



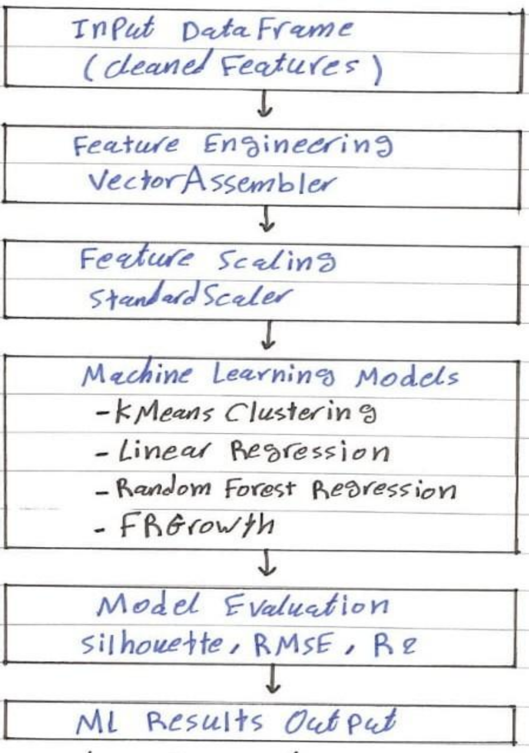
→ This diagram shows the overall system architecture. The user uploads a CSV dataset through the Google Colab interface. The Spark driver initializes the Spark session and distributes the workload across multiple virtual workers using local[N] mode. The results of data analysis and machine learning tasks are then collected and displayed to the user.

2) Flow Diagram :-



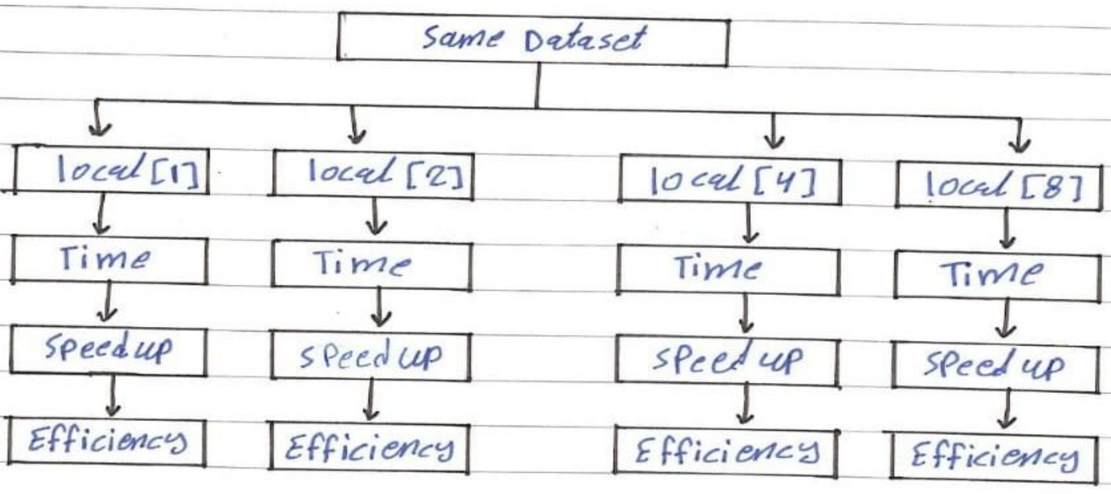
→ This flow diagram illustrates the execution sequence of the system. It starts from data upload, followed by spark session initialization, data loading, statistical analysis, aggregation, machine learning processing, and finally result presentation.

3) ML Diagram :-



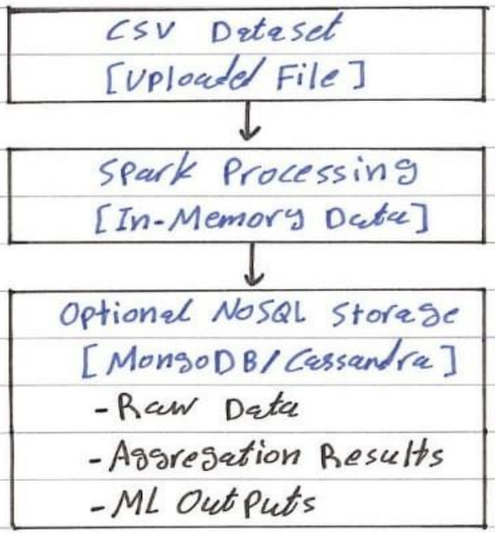
→ This diagram shows the machine learning pipeline implemented using Spark MLlib. The input data is transformed, scaled, processed by multiple ML models.

4) Workload Diagram:-



→ This diagram represents how the same workload is executed using different levels of parallelism. The experiment measures execution time, speedup, and efficiency for N=1,2,4 and 8 virtual workers.

5) Data Storage & NOSQL:-



→ Although the system processes data directly within Spark for simplicity, this diagram shows how a NoSQL database can be integrated in a real-world deployment to store raw datasets, intermediate results and machine learning outputs.

4. Implementation

Initially, user interaction is implemented through the Google Colab interface, where the user uploads a CSV file containing taxi trip data using the files.upload() function. Once the file is uploaded, the system automatically detects the filename and uses it as the input source for all subsequent processing steps. No manual configuration or parameter tuning is required from the user.

After the dataset is uploaded, a Spark session is programmatically created using SparkSession.builder. To simulate a distributed execution environment under limited hardware resources, Spark is executed in local [N] mode, where N represents the number of virtual workers (threads) running in parallel on the same physical machine. Multiple Spark sessions are created sequentially with N = 1, 2, 4, and 8 to support scalability evaluation.

The uploaded CSV file is then read into a Spark DataFrame using spark.read.csv()with schema inference enabled.[4] The implementation selects the relevant numerical attributes required for analysis, such as trip distance and fare amount, and applies basic data cleaning by removing null values. These attributes are consistently used across descriptive statistics, aggregation tasks, and machine learning operations.

Descriptive statistics are implemented using Spark SQL aggregation functions. The system computes count, mean, minimum, and maximum values for each selected attribute. These statistics are printed directly on the screen, allowing the user to obtain an initial understanding of the dataset characteristics before proceeding to more advanced processing.

To demonstrate distributed data processing, a Map/Reduce-style aggregation task is implemented using Spark's groupBy() and count() operations[5]. Taxi trips are grouped based on pickup location identifiers, triggering distributed execution across the available virtual workers.

Machine learning functionality is implemented using Spark MLlib[2]. Four machine learning tasks are executed on the dataset to simulate a realistic analytical workload. Feature engineering is performed using VectorAssembler, followed by feature scaling with StandardScaler. Clustering is implemented using the KMeans algorithm, and clustering quality is evaluated using the Silhouette metric. In addition, two regression models Linear Regression and Random Forest Regression are trained to predict fare amounts, with evaluation metrics including RMSE and R^2. Frequent pattern mining is performed using the FPGrowth algorithm to extract frequent itemsets from the data.

Performance measurement is implemented by running the same distributed workload multiple times while varying the number of virtual workers using Spark local [N] mode with N = 1, 2, 4, and 8. For each configuration, the system measures the execution time of the Map/Reduce aggregation task, allowing a direct comparison of processing speed across different numbers of workers.

The execution time obtained with a single worker (N = 1) is used as a baseline. Based on this baseline, the system calculates speedup, which shows how performance changes when additional workers are used, and efficiency, which indicates how effectively the available workers are utilized. These values are displayed in a structured table, providing a clear view of scalability behavior across all worker configurations.

Regarding data storage, all data processing is performed directly in memory within the Spark environment without relying on an external database system. This design simplifies execution in a simulated cloud environment. However, the implementation remains compatible with future integration of NoSQL storage systems.

Finally, all outputs including descriptive statistics, performance tables, and machine learning results are displayed directly on the screen within the cloud execution environment.

5. Experiments and Evaluation

This is the result of the last 3 readings while the system was running :

Cluster Size	Execution Time (sec)	Speedup	Efficiency
1 Machine	19.67	1.00	1.00
2 Machines	12.48	1.58	0.79
4 Machines	11.73	1.68	0.42
8 Machines	11.96	1.64	0.21

Explanation:

This analysis shows a decrease in execution time when moving from one node to two. This indicates that parallel processing is initially efficient.

When using four nodes, execution time improves slightly, but this improvement is less significant than adding a second node.

When the number of nodes increases to eight, execution time does not improve; in fact, it decreases slightly. This is because the system spends more time managing workers instead of performing actual calculations.

Generally, optimal performance occurs at the first node. At the eighth node, there is increased overload, and parallel processing is only beneficial up to a certain point.

Cluster Size	Execution Time (sec)	Speedup	Efficiency
1 Machine	12.48	1.00	1.00
2 Machines	10.37	1.20	0.60
4 Machines	9.68	1.29	0.32
8 Machines	10.23	1.22	0.15

Explanation

This reading shows a limited improvement when increasing the number of nodes.

Moving from one node to two nodes improves speed slightly, but less than expected.

Using four nodes improves performance slightly, but efficiency decreases because the workload is not large enough to occupy all nodes.

At eight nodes, performance deteriorates, meaning the workload of coordinating tasks outweighs the benefit of parallel execution.

Generally, the best nodes are also up to Node. 4, where using eight nodes clearly degrades performance and the coordination workload dominates execution.

Cluster Size	Execution Time (sec)	Speedup	Efficiency
1 Machine	19.64	1.00	1.00
2 Machines	13.48	1.46	0.73
4 Machines	12.16	1.62	0.40
8 Machines	11.48	1.71	0.21

Explanation

In this reading, performance improves gradually as the number of nodes increases, but this improvement is not linear. Moving from one node to two nodes shows a significant decrease in execution time, meaning Spark benefits from basic parallelism.

Using four nodes improves performance compared to using two nodes, but the acceleration is still limited. This is because all working nodes operate on the same physical machine, leading to competition for CPU and memory resources.

When using eight nodes, execution time does not improve significantly compared to using four nodes. Although a larger number of working nodes are available, the coordination and scheduling workload increases. This workload reduces overall efficiency, especially since the dataset size is not large enough to fully utilize eight parallel working nodes.

Generally, performance is best up to Node 4, and Node 8 does not offer a significant performance improvement. The main reasons for this are the system workload, shared hardware resources, and the limited dataset size.

6. User Support

The system is designed to be simple and user-friendly. The user interacts with the system through Google Colab, where the CSV dataset is uploaded directly using the provided interface. Once the file is uploaded, all processing steps start automatically without requiring any additional input or configuration from the user.

The system handles data loading, distributed processing, performance measurement, and machine learning tasks internally. All outputs, including descriptive statistics, scalability results, and machine learning outcomes, are displayed clearly on the screen.

Project Links:

Cloud Program (Google Colab):
https://colab.research.google.com/drive/1SGGSaSuOu19xS_3K2R17E84vOGWTyNSn?usp=sharing

GitHub Repository:
https://github.com/Abd-Elrahman-Jehad/spark-taxi-cloud-project.git

Video Demonstration:
https://youtu.be/Q3zemq1KLJ0

7. Conclusion

In this project, a cloud-based data processing system using Apache Spark was implemented to analyze a large dataset of taxi trips. The aim was to study the behavior of distributed processing and evaluate the system's performance when the number of nodes changed, rather than focusing solely on analytical results.

The system successfully performed metadata statistics, data aggregation using MapReduce, and several machine learning tasks using Spark MLlib. These tasks demonstrated how Spark manages data processing and machine learning workloads in a distributed manner, even when running in local [N] mode on a single machine.

The experimental results showed that increasing the number of nodes improved performance up to a certain point. Using two and four nodes reduced execution time compared to a single node, clearly demonstrating the benefits of parallel processing. However, with eight nodes, the performance improvement became limited, and efficiency decreased. This was primarily attributed to the coordination burden, task scheduling costs, shared hardware resources, and the fact that the dataset size was not large enough to fully utilize all the nodes.

Overall, the results confirm that distributed processing can improve performance when the number of nodes is chosen appropriately. Adding more nodes does not always improve performance and may even cause performance degradation in resource-constrained environments.

References

References mentioned in the report :

[1] Apache Spark Documentation:

https://spark.apache.org/docs/latest/

[2] Spark MLlib Documentation:

https://spark.apache.org/docs/latest/ml-guide.html

[3] Google Colab Documentation:

https://colab.research.google.com/

[4] Kaggle Datasets Platform:

https://www.kaggle.com

[5] MapReduce Programming Model Overview:

https://research.google/pubs/pub62/