Project

Part1

- This part is mandatory

Project Overview:

You will set up a distributed Spark cluster using Docker, VMware, or VirtualBox, with **one master node** and **two worker nodes** (or **one worker node in a simplified setup** for real systems). The dataset will be **manually split** across the master and workers, and after reading the data into Spark, you will perform partitioning, data processing using Spark SQL, and implement a machine learning pipeline using Spark's MLlib. This project combines concepts of distributed data handling, resource management, and training a machine learning model on a distributed dataset.

Project Steps:

1. Cluster Setup:

You will set up a Spark cluster using Docker, VMware, or VirtualBox with:

- **1 master node**, and **2 worker nodes** (or **1 worker node in for real systems**).

The cluster setup will be configured for distributed task management and communication between the master and worker nodes. Docker Compose or VirtualBox settings will ensure proper resource allocation (e.g., RAM, CPU) for each worker node.

---

2. Manual Data Splitting Across Nodes:

You will manually split the **same dataset** into parts and place each part on a different node:

- **Worker 1**: Receives one portion of the dataset.
- **Worker 2**: Receives another portion of the dataset (or only one worker in the simplified setup handles half the data).
- **Master**: Holds the remaining portion of the dataset.

Each node will only have access to its part of the dataset, which will later be combined during distributed processing.

---

3. Data Loading on Each Node:

Each node (master and workers) will load its part of the dataset into Spark. Once the data is loaded, Spark will **combine** the dataset into a **unified DataFrame** across all nodes.

---

4. Data Partitioning and Processing with Spark SQL:

After loading the data, you will **repartition** the dataset using Spark SQL, ensuring that the data is efficiently distributed across the worker nodes for parallel processing. You will perform initial **data cleaning**, **exploration**, and **feature engineering** using SQL queries, such as filtering, aggregation, or transformation of features.

---

5. Machine Learning Pipeline:

Once the data is prepared, you will implement a **machine learning pipeline** using Spark MLlib. This pipeline will include:

- **Feature transformations** (e.g., scaling, vectorizing features).
- **Model training**: Train a machine learning model (e.g., logistic regression, decision tree, or random forest) on the distributed dataset.
- **Model evaluation**: Evaluate the trained model's performance using metrics like accuracy, precision, or F1-score.

---

6. Resource Management:

Resource management is one of the most critical aspects of distributed computing and machine learning, as it directly affects the performance of data processing and model training.

Key Concepts in Resource Management:

- **Executor Memory**: This defines how much memory each Spark executor has for processing tasks. Proper memory allocation ensures that the worker nodes can efficiently handle large datasets without running into memory issues like out-of-memory errors. you will configure the **spark.executor.memory** setting for each worker node to allocate an appropriate amount of memory.
    - **Best Practice**: Allocate memory slightly lower than the total available RAM on each node to leave room for OS-level processes.

- **Executor Cores**: The number of CPU cores allocated to each executor affects how many tasks can run in parallel. More cores allow Spark to process more partitions simultaneously, speeding up query execution and model training. You will adjust **spark.executor.cores** to optimize CPU usage based on the available hardware.
    - **Best Practice**: Ensure that the total number of cores used across all executors doesn't exceed the physical cores available on the worker nodes to avoid CPU contention.
- **Dynamic Resource Allocation**: In large distributed systems, Spark can dynamically adjust the number of executors based on the workload. You will explore **dynamic resource allocation** (optional), allowing Spark to scale up or down based on the current job's resource requirements.
    - **Configuration**: Enable **spark.dynamicAllocation.enabled** to allow Spark to automatically add or remove executors based on demand.
- **Shuffle Partitions**: When data is shuffled between nodes (e.g., during a join or groupBy operation), Spark uses a number of partitions to divide the work. You will learn how to adjust the **shuffle partitions** via the **spark.sql.shuffle.partitions** setting to balance memory usage and parallelism.
    - **Best Practice**: For small datasets, fewer partitions are needed. For larger datasets, more partitions may be required to ensure efficient data movement.

Steps for Resource Configuration:

1. **Configuring Memory for Executors**: Set the memory available for each worker node, ensuring efficient use of available system memory. Example: **2GB for Worker 1**, **4GB for Worker 2**.
2. **Defining CPU Core Usage**: Set the number of cores each worker can use. Example: **2 cores for Worker 1**, **4 cores for Worker 2**.
3. **Monitoring Resource Utilization**: Use the **Spark Web UI** to monitor how much memory and CPU are being used by each executor during data processing and machine learning tasks.
4. **Fine-Tuning Shuffle Partitions**: Adjust shuffle partitions to balance the trade-off between parallelism and memory efficiency. Start with the default and adjust based on the dataset size and job performance.

Key Outcome: By learning how to manage resources effectively, you will be able to optimize performance, prevent memory errors, and ensure efficient parallel processing across the cluster.

---

7. Performance Monitoring:

You will monitor the performance of the Spark cluster using the **Spark Web UI**. The UI provides detailed insights into:

- **Task distribution**: View how tasks are distributed across the worker nodes.
- **Memory usage**: Monitor memory consumption on each node and identify potential bottlenecks or inefficient resource usage.
- **CPU utilization**: Track CPU load and ensure that each worker is operating efficiently.

Performance monitoring helps identify areas where resource allocation can be improved to boost the overall performance of the distributed system.

---

8. Tuning the Machine Learning Model:

After evaluating the initial model, you will fine-tune the machine learning model by:

- **Adjusting hyperparameters**: Modify learning rates, regularization parameters, or tree depth to improve model accuracy.
- **Running cross-validation**: Use cross-validation to test multiple models with different hyperparameters and find the best-performing configuration.

Key Learning Objectives:

- **Manual Data Splitting**: Learn to manually split and distribute a dataset across workers and the master, and read it into Spark for unified processing.
- **Cluster Setup**: Set up a distributed Spark cluster using Docker, VMware, or VirtualBox and manage communication between master and worker nodes.
- **Resource Management**: Understand how to allocate and manage resources (RAM, CPU, executor memory) to optimize machine learning performance across worker nodes. Learn to fine-tune **memory**, **CPU cores**, and **shuffle partitions** to achieve the best performance.
- **Performance Monitoring**: Learn to monitor cluster performance using the Spark Web UI and optimize resource allocation based on real-time usage data.
- **Machine Learning on Distributed Data**: Train and evaluate a machine learning model using Spark's MLlib on a distributed dataset.

Part 2: Comparing Distributed Machine Learning with Separate Machine Learning on Each Node (Running Jobs Simultaneously)

Steps for Part 2:

1. **Simultaneous Machine Learning Jobs on Each Node's Data**:
    - You will train separate machine learning models on the data located in Worker 1, Worker 2, and the Master. These jobs will be executed **at the same time** to simulate parallel machine learning tasks running concurrently on the Spark cluster.
    - Each node will handle its own portion of the dataset independently, but all jobs will be triggered simultaneously, requiring you to configure the Spark cluster to handle multiple jobs efficiently.
    - Key configuration aspects include allocating memory and CPU resources for each job to avoid contention and ensuring that both jobs can run concurrently without bottlenecks.
2. **Simultaneous Distributed Machine Learning on the Unified Dataset**:
    - After running the simultaneous jobs on individual datasets, you will train a distributed machine learning model on the unified dataset. The unified dataset will be spread across all nodes in the cluster.
    - The distributed job should also run in parallel with other tasks or jobs in the system, ensuring that the cluster handles concurrent execution efficiently.
3. **Performance Monitoring for Concurrent Jobs**:
    - You will monitor the performance of **all jobs running at the same time** using the Spark Web UI, which will provide insights into resource usage such as memory and CPU load.
    - They will learn to identify potential issues with resource allocation when running multiple jobs concurrently and use this data to adjust memory and core configurations dynamically.
4. **Comparison and Analysis**:
    - **Model performance**: Compare metrics (e.g., accuracy, precision, F1-score) from models trained simultaneously on separate data and the distributed model trained on the unified dataset.
    - **Training time**: Analyze how long each job takes when run simultaneously on separate nodes compared to a distributed job on the unified dataset.

- **Resource utilization**: Examine how running multiple jobs at the same time affects CPU and memory usage compared to running a single distributed job.

Outcome:

By the end of this section, you will have a clear understanding of how Spark handles **multiple machine learning jobs running concurrently** and the trade-offs involved in running models on separate datasets simultaneously versus on a unified distributed dataset. This exercise will highlight the importance of resource management and the role of parallel processing in distributed machine learning.

This structure focuses on executing **simultaneous** jobs and monitoring their performance, which will give you experience in managing parallel workloads in Spark.

Part 2: Extra point

(Data Streaming)

- **This part is not mandatory.**
- **You can combine two groups to complete this part.**
- **If you do this, all students in the group will receive extra points**

1. Cluster Setup:

- Set up the Spark cluster with one master node and two worker nodes using Docker, virtual environment , or real cluster, configured to handle both static and streaming data.
- The workers will read data at specific intervals and send the data to the master node, which will process the data in real-time or micro-batches.

2. Streaming Data Source Setup:

- Simulate a streaming data source (e.g., using Kafka, socket streams, or file streams) to generate real-time data.
- The workers will receive their portion of the streaming data in intervals and send this data to the master node for processing. This simulates a real-time system where workers contribute new data at regular intervals.
- The streaming data could represent scenarios such as sensor readings, stock market data, or network logs.

3. Comparing Static and Streaming Machine Learning Algorithms:

- **Static (Batch) Data**:
    - Train a machine learning model (e.g., logistic regression, decision tree, or random forest) using the entire dataset in batch mode. This model will be trained in a static manner where the entire dataset is processed at once.
- **Streaming Data**:
    - Implement two approaches for machine learning on streaming data:
        1. **Same Algorithm**: Use the same algorithm as in the static approach (e.g., logistic regression or decision tree), but applied to micro-batches of streaming data as workers send their data to the master.

2. **Different Algorithm**: Use a streaming-friendly algorithm such as Stochastic Gradient Descent (SGD) or online logistic regression, which updates the model incrementally as new data arrives at the master.

- **Algorithm Selection**:
    - Discuss the appropriateness of different algorithms for batch learning versus streaming data processing, such as batch-learning algorithms like decision trees for static data and online learning algorithms like SGD for streaming data.

4. Machine Learning Pipeline for Static and Streaming Data:

- **Static Data Pipeline**:
    - Perform data cleaning, feature transformation, and model training on the entire dataset in batch processing mode. This will be done on the master node after the static data is loaded from the workers.
- **Streaming Data Pipeline**:
    - For streaming data, the workers will send data to the master node at regular intervals. The master will:
        - Perform data cleaning and feature transformation for each micro-batch of incoming data.
        - Update the machine learning model in real-time using either the same algorithm as the static case or an online learning algorithm.
    - Two approaches:

1. Use the same algorithm that was used for static data but trained on micro-batches.

2. Use a streaming-optimized algorithm to handle incremental updates on the incoming data.

5. Resource Management for Static and Streaming Jobs:

- **Static Jobs**: Allocate memory and CPU resources to handle batch processing on the entire dataset.
- **Streaming Jobs**: Ensure the workers efficiently handle streaming data and that the master processes the micro-batches in real-time.
- Monitor memory and CPU usage using the Spark Web UI for both static and streaming jobs to optimize resource allocation.

6. Workers Sending Data to Master:

- Workers will read their portion of the data at specific intervals and send it to the master node. This simulates a real-time streaming system where workers operate independently and continuously send data for processing.
- The master will combine the data from all workers into a unified dataset for both batch processing (static data) and real-time processing (streaming data).

7. Performance Monitoring and Comparison:

- **Model Performance**:
    - Compare the performance (e.g., accuracy, precision, F1-score) of the machine learning model trained on static data with the model trained on streaming data using the same algorithm.
    - Analyze the performance differences when using a different algorithm for streaming data, such as online learning algorithms optimized for real-time updates.
- **Training Time**:
    - Compare the time taken to train the static batch model with the time it takes to continuously update the streaming model in real-time. Monitor the trade-offs between training time and performance.
- **Resource Usage**:
    - Use the Spark Web UI to track memory and CPU usage during static and streaming data processing. Evaluate the efficiency of resource consumption for both batch and streaming jobs.

8. Key Comparisons:

- **Same Algorithm for Static and Streaming**:
    - Analyze how the same machine learning algorithm behaves when applied to both static and streaming data.
    - Discuss whether the model's performance and resource consumption change when used in a streaming environment.
- **Different Algorithms for Static and Streaming**:
    - Compare the performance of batch-learning algorithms (e.g., decision tree or random forest) for static data with online-learning algorithms (e.g., SGD) for streaming data.
    - Highlight the benefits and trade-offs of using streaming-optimized algorithms for real-time data processing.

Key Learning Objectives:

1. **Cluster Setup**: Set up and manage a Spark cluster with master and worker nodes to handle both static and streaming data.
2. **Workers Sending Data to Master**: Understand how to simulate workers sending data to the master at specific intervals for real-time processing.
3. **Static and Streaming Data Processing**: Learn how to process both static (batch) and streaming (real-time) data in Spark.
4. **Same vs. Different Algorithms**: Compare the effectiveness of using the same algorithm for both static and streaming data versus using streaming-optimized algorithms.
5. **Streaming-Friendly Machine Learning**: Implement machine learning algorithms that handle real-time data updates in Spark Streaming.
6. **Resource Management**: Efficiently allocate CPU, memory, and other resources to handle both static and streaming data processing jobs across a distributed cluster.
7. **Performance Monitoring**: Use the Spark Web UI to monitor job performance, resource usage, and latency for both static and streaming tasks.
8. **Practical Application**: Gain experience in designing and managing machine learning pipelines for both static and streaming data in a distributed environment.