Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 10

Aim: To perform Batch and Streamed Data Analysis using Apache Spark.

Theory:

1. What is Streaming? Explain Batch and Stream Data.

Answer:

In Data processing, we generally deal with two major types: 1)Batch processing and 2)Stream processing.

- Batch Data Processing refers to collecting data over a period of time and then processing it all at once. Think of it like baking cookies: you prepare a whole batch and then put it in the oven. It's great when you don't need real-time insights and can wait for results.

- Stream Data Processing (also called real-time processing) deals with data that flows in continuously like sensor data, social media updates, or payment transactions. Instead of waiting for a large chunk, stream processing handles data piece by piece, as it comes in.

Streaming is the process of analyzing data in motion. It's especially useful when timely insights matter like fraud detection, server health monitoring, or stock market analytics.

2. How Does Data Streaming Take Place Using Apache Spark?
Answer:
Apache Spark provides a module called Spark Streaming (and its newer version called Structured Streaming) to process real-time data streams.

Working:
- Data Source: Data is continuously received from sources like Kafka, socket connections, or files being updated in real-time.

- Spark Streaming Engine: Spark divides the incoming data into small, manageable batches (micro-batches). Even though it's called streaming, it's actually processing in very small intervals like every second or every few seconds.

- Transformations and Actions: Just like in batch mode, you can apply filtering, grouping, or aggregation logic to the data in each micro-batch.

- Output Sink: The results can be saved or pushed to dashboards, databases, or alerts systems giving near-instant updates based on the incoming data.

Spark special is that the same programming model can be used for both batch and stream processing making it flexible and powerful for developers and analysts alike.

Steps for Batch Data Analysis using Apache Spark

1. **Start Apache Spark Environment**
   Launch Apache Spark through a local installation, cloud platform, or a notebook interface like Jupyter or Databricks. This initializes the engine to process your data.

2. **Read a Batch Dataset**
   Load a static file (like a CSV or JSON) that contains stored historical data. This is typically a complete dataset with a defined start and end.

3. **Explore the Dataset**
   Get an overview of the data by checking column names, data types, and sample records. This helps identify what kind of analysis or cleaning might be required.

4. **Clean and Prepare the Data**
   Handle missing values, rename columns for clarity, fix incorrect data types, and remove duplicates. Clean data is essential for meaningful analysis.

5. **Perform Transformations and Aggregations**
   Apply operations like filtering specific rows, grouping by a column (e.g., region, product), and calculating metrics like average, count, or total sales.

6. **Store or Display Output**
   After analysis, the results can be saved to a new file (like CSV or Parquet), visualized using tools, or displayed in the console.

Steps for Streamed Data Analysis using Apache Spark

1. **Initialize Spark with Structured Streaming**
   Start a Spark session and configure it to accept live data using Structured Streaming, which is Spark's modern approach to handling real-time data.

2. **Connect to a Streaming Data Source**
   Link Spark to a live data source like Apache Kafka, Socket, or a directory where new files keep arriving. This tells Spark where to expect new data continuously.

3. **Define the Schema for Streaming Data**
   Since streaming data doesn't always come with a header, you define what each field means e.g., timestamp, value, sensor_id so Spark knows how to process it.

4. **Apply Streaming Operations**
   As new data flows in, apply operations such as filtering rows (e.g., temperature >

100), grouping over time windows (like every 10 seconds), or aggregating (e.g., average values).

5. **Write Stream Output to a Sink**
   The results are written continuously to a destination like the console, a file system, a database, or even a live dashboard.

6. **Monitor the Streaming Pipeline**
   Monitor the job to ensure the stream is running smoothly. You can check data arrival, processing speed, and memory usage. The stream continues running until manually stopped.

Conclusion

This experiment helps us understand two powerful ways of working with data using Apache Spark: batch and streamed processing. Batch analysis is well-suited for historical or static data, allowing deep dives and summary reports. On the other hand, streamed analysis enables real-time decision-making by processing live data as it arrives. Apache Spark handles both seamlessly using its unified framework and scalable infrastructure. By learning both approaches, we equip ourselves with the flexibility to tackle a wide variety of real-world data problems whether they require immediate insight or deep historical trends.