Name:Bhushan Malpani Div:D15C Roll No.:33

Experiment-10

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

Theory:

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

Ans:
Streaming is the process of continuously processing incoming data in real-time as it is generated. It's especially useful in scenarios that demand immediate response, such as detecting fraudulent transactions, monitoring stock trends, or powering live analytics dashboards. Streaming data is endless, time-sensitive, and arrives in a continuous flow.

On the other hand, **batch processing** involves collecting data over a specific period and then processing it all at once. This method is commonly used for activities like generating reports, transforming large data sets, or loading data into a warehouse. Batch data is finite, processed at scheduled intervals, and handled in segments or chunks.
Examples:

- **Batch:** Compiling sales data to create a monthly report.
- **Stream:** Analyzing website clicks from users in real time.

2. **How data streaming takes place using Apache Spark:**

Ans:
Apache Spark enables real-time data processing through its **Structured Streaming** module. This framework treats incoming streaming data as a continuously growing table and performs operations incrementally using familiar DataFrame and SQL APIs, just like batch processing.

Data can be streamed into Spark from multiple sources like **Apache Kafka**, socket connections, folders, or cloud-based storage. Once ingested, Spark performs operations such as filtering, selecting, grouping, and aggregating the data. It also supports window functions for time-based analysis, watermarking to handle delayed data, and checkpointing to ensure recovery from failures.

Under the hood, Spark uses a **micro-batch architecture**, where it divides incoming data into small batches that are processed quickly in succession. The results can then be written to destinations like HDFS, databases, or visualization tools.

Key Features:

- Unified programming interface for batch and stream workloads

- Support for managing application state across events
- Seamless integration with structured data sources
- Scalable and fault-tolerant processing engine

Use Case Examples:

- Real-time fraud detection in financial systems
- Analyzing server logs as they are generated
- Monitoring live social media activity

Conclusion:

Through this experiment, I developed a clear understanding of the contrast between **batch** and **stream processing**. Batch jobs are best for working with historical or periodic datasets, while streaming is tailored for real-time, ongoing data flows. I explored **Structured Streaming in Apache Spark**, which offers a unified platform for handling both streaming and batch data.

I learned how to connect live data streams from tools like Kafka, apply transformations using Spark APIs, and dynamically output results. This experience deepened my appreciation for Spark's ability to manage complex data workflows with scalability and reliability, making it an excellent tool for modern, real-time analytics systems.