Fundamentals Of Spark

**1.What is Apache Spark?**

🡪Apache spark is a unified computing engine and set of libraries for parallel data processing on a computer cluster.

1. **What is Unified?**

* Spark is designed to support a wide range of tasks over the same computing engine.
* Example - Data Scientist, Data Analyst and Data Engineers all can use the same platform for their analysis, transformation and modelling.

1. **What is Computing Engine?**

* Spark is limited to a computing engine. It does not store the data.
* Spark can connect to different data sources like HDFS, JDBC/ODBC, Azure storage etc.
* Sparks works with almost all data storage systems.

**NOTE: Spark does not store the data it only computes the data.**

1. What is Computer Cluster?

* It is work on master-slave architecture that means one central machine divides the task as per requirements.

**2.Why Apache Spark?**

**🡪** Apache spark works with distributed approach so it provides high availability and can process large volume of data with less time.

1. **Big Data:**

* Big Data is nothing but the large volume of data with various types of data.
* There are 3V’s of Big Data-

1. Velocity - Data processing per minute or Hour like 100GB/Hr.
2. Variety -Type of data like structured, semi-structured or Unstructured.
3. Volume - Size of data like 100GB, 100TB

ELT – Extract, Transform and Load used in data warehouse for storing data.

ELT – Due to large volume of data it Extract, Load and Transform in data lake.

Issues:

1. Storage: Due to large volume of data.
2. Processing: 1.RAM – Needs more RAM

2 - CPU- Need High processing speed for handling large data.

For handling such type of issues, we have 2 approaches available.

1. Monolithic Approach: Increasing the hardware capacity of system like increasing Hard-Disk, RAM.
2. Distributed Approach: It works like master slave architecture that distributes the volume of data into multiple system thus it can easily process the data.

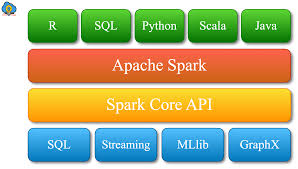
Monolithic V/S Distributed:

|  |  |
| --- | --- |
| Monolithic | Distributed |
| 1.Vertical Scaling (Increasing capacity of System.) | 1.Horizontal Scaling (Increasing system as per volume of Data) |
| 2.Operating cost is very expensive due to all things are happens on single system. | 2.Operating cost is economically low. |
| 3.Low availability. | 3.High Availability |

**3.Hadoop V/S Spark**

|  |  |  |
| --- | --- | --- |
| Parameters | Hadoop | Spark |
| 1.Performance | Hadoop is slower than spark because it writes the data back to disk and read again from disk to in memory. | Spark is faster than Hadoop because spark do all the computations in memory. |
| 2.Batch/Streaming | Build for Batch data processing | Build for Batch as well as Stram data processing |
| 3.Ease of use | I. Difficult to write code in Hadoop.  II. Hive was built to make it easier. | I. Easy to write and debug code. We have interactive shell to develop and test.  II. Spark provide high level and low-level API. |
| 4.Security | Uses Kerberos authentication and ACL authorization. | Doesn’t have solid security features. |
| 5.Fault Tolerance | It is having block of data and replication factor to handle the failure.  By default, Replication factor is 3. that means same data is replicated on different nodes in HDFS. | It uses the DAG (Directed Acyclic Graph) to provide fault tolerance.  DAG works like it can create the multiple processes that can store output of each process to next process. |

**4.Spark Eco-System**



**Core Components of Spark Ecosystem**

1. **Spark Core:**

* The foundational engine of spark, responsible for basic I/O functions, task scheduling, memory management and fault recovery.
* Provides In memory computing capabilities that enhances the processing speed and performance.

1. **Spark SQL:**

* A module for structured data processing.
* Allows querying data via SQL and integrates seamlessly with various data sources such as Hive, Avro, Parquet, and JSON.
* Supports Data Frames and Datasets API for more efficient and type-safe operations.

1. **Spark Streaming:**

* Enables real-time data processing.
* Processes live data streams from sources like Kafka, Flume, and HDFS.
* Provides high-throughput, fault-tolerant stream processing.

1. **MLlib (Machine Learning Library):**

* A scalable machine learning library.
* Includes various algorithms for classification, regression, clustering, collaborative filtering, and dimensionality reduction.
* Supports feature extraction and transformation utilities.

1. **GraphX:**

* A library for graph and graph-parallel computation.
* Allows users to build and manipulate graphs (networks of nodes and edges) and apply graph algorithms like PageRank, connected components, and triangle counting.

1. **SparkR:**

* An R package that provides a lightweight front end to use Apache Spark from R.
* Facilitates the analysis of large datasets using R, leveraging Spark’s distributed computing capabilities.

**Key Features and Benefits**

1. **Speed:**

In-memory processing boosts performance, making Spark up to 100 times faster than Hadoop MapReduce for certain applications.

1. **Ease of Use:**

High-level APIs in Java, Scala, Python, and R simplify the development of big data applications.

1. **Scalability:**

Easily scales from a single server to thousands of machines, processing petabytes of data.

1. **Flexibility:**

Integrates with Hadoop and its ecosystem (HDFS, YARN, etc.), as well as various data sources and formats.

**5. Spark Architecture:**

**Key Components of Spark Architecture**

1. **Driver:**

* The driver is the master node of a Spark application.
* It is responsible for maintaining the Spark Context, which coordinates all tasks and resources.
* The driver translates the user's code into a series of tasks that are distributed across the cluster.

1. **Cluster Manager:**

* Spark can run on various cluster managers such as Standalone, Apache Mesos, Hadoop YARN, or Kubernetes.
* The cluster manager allocates resources (CPU, memory) across the cluster.

1. **Executors:**

* Executors are worker nodes that run individual tasks assigned by the driver.
* Each executor is responsible for executing tasks and storing data for future tasks.
* Executors provide in-memory storage for RDDs (Resilient Distributed Datasets) that are cached by user programs through Spark Context.

1. **Tasks:**

* Tasks are the smallest unit of work in Spark, representing a single operation on a partition of data.
* Each job submitted by the user is divided into multiple stages, which are further divided into tasks.

**Spark Application Lifecycle**

1. **Job Submission:**

* A Spark application starts when the user submits a job.
* The driver creates a Spark Context, which connects to the cluster manager.

1. **Task Scheduling:**

* The driver breaks down the application into jobs, stages, and tasks.
* Tasks are scheduled and distributed to executors by the cluster manager.

1. **Task Execution:**

* Executors run the assigned tasks and perform operations on the data.
* Intermediate data can be cached in memory for faster access in subsequent tasks.

1. **Result Collection:**

* Executors send the results of the tasks back to the driver.
* The driver collects and processes the results to complete the job.

**Resilient Distributed Datasets (RDDs)**

* RDDs are the fundamental data structure of Spark, representing an immutable, distributed collection of objects.
* Transformations create new RDDs from existing ones (e.g., map, filter).
* Actions trigger computation and return results to the driver or write data to storage (e.g., count, collect).

**Key Features:**

1. **In-Memory Computation:**

Spark performs operations in memory, significantly speeding up data processing compared to disk-based systems.

1. **Fault Tolerance:**

RDDs can recover from node failures, as lineage information allows re-computation of lost data.

1. **Lazy Evaluation:**

Transformations are evaluated lazily, meaning they are not executed until an action requires a result. This optimizes the execution plan.

**Execution Flow**

1. **Transformations:**

The user defines a series of transformations on the data (e.g., filtering, mapping).

1. **Action:**

When an action is called, Spark builds a directed acyclic graph (DAG) of stages.

1. **DAG Scheduler:**

Breaks the DAG into stages of tasks.

1. **Task Scheduler:**

Sends tasks to the cluster manager.

1. **Cluster Manager:**

Distributes tasks to executors.

1. **Executors:**

Execute tasks and store intermediate results.

**6.Transformation and Actions in Spark**

**Potential Interview Questions**

1. **What is transformation and how many types of transformation do we have?**

* transformation is the process of taking raw data that has been extracted from data sources and turning it into usable datasets that means cleaning, modifying.

**Types of Transformation**

**1.Narrow Dependency:**

A transformation that does not required data movement between partitions.

Ex – filter, select, union, map() etc.

**2. Wide Dependency:**

A transformation that does required data movement between partitions.

**Ex** – Group By, Join, Distinct etc.

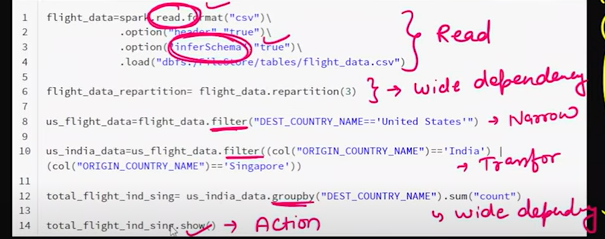
1. **What happens when we use group by or joins in transformation?**

* Data shuffling is happening between partitions thus we can consider it as wide transformation.

**7.DAG And Lazy Evaluation In Spark**

**DAG:**

DAG is nothing but the Directed Acyclic Graph which means it never goes run in cycle. Also, for each job DAG is generated and Job is created while we hit action.

****

**Lazy Evaluation:**

It means that Spark doesn't immediately execute operations when they are called. Instead, it builds an execution plan, which is only triggered when an action (such as a data output operation) is called.