ISIT312

Spark

Spark

- Apache Spark is an open-source, distributed computing system designed for big data processing and analytics.
- Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Spark vs Hadoop

Spark vs Hadoop

- Apache Spark and Apache Hadoop are both open-source frameworks designed for processing large datasets, but they have significant differences in terms of architecture, performances, and use cases.

Spark vs Hadoop (Similarities)

- **Big Data Processing:** Both Spark and Hadoop are designed to handle big data, allowing the processing of large datasets distributed across many machines.
- **Open Source:** Both are open-source projects maintained by the Apache Software Foundation.
- **Distributed Computing:** Both frameworks operate on clusters of machines, enabling distributed storage and processing.

Spark vs Hadoop (Differences)

1. Processing Model:
   - **Hadoop:** Primarily uses the MapReduce programming model, which involves a series of map and reduce tasks. This model can be slower due to frequent disk I/O operations between each stage.

Spark vs Hadoop (Differences)

1. **Processing Model:**
   - **Spark:** Utilizes in-memory computing to speed up data processing. Spark's Resilient Distributed Datasets (RDDs) allow it to perform operations in memory, reducing the need for slow disk I/O.

Spark vs Hadoop (Differences)

2. **Performance:**
   - **Hadoop:** Disk-based processing makes Hadoop slower compared to Spark, especially for iterative algorithms and interactive queries.
   - **Spark:** In-memory processing makes Spark up to 100 times faster than Hadoop for certain tasks. Spark can also handle batch and streaming data in a unified framework.

Spark vs Hadoop (Differences)

3. **Ease of Use:**
   - **Hadoop:** Has a steep learning curve due to its reliance on Java and the complexity of the MapReduce programming model.
   - **Spark:** Offers APIs in multiple languages (Java, Scala, Python, R) and provides higher-level libraries for SQL, streaming, machine learning, and graph processing, making it more accessible and easier to use for a wider range of developers.

Spark vs Hadoop (Differences)

4. **Fault Tolerance:**
   - **Hadoop:** Achieves fault tolerance by replicating data across multiple nodes in the Hadoop Distributed File System (HDFS). If a node fails, the data can be retrieved from another node.
   - **Spark:** Uses Resilient Distributed Datasets (RDDs) to ensure fault tolerance. RDDs can be recomputed using lineage information if a partition of an RDD is lost.

Spark vs Hadoop (Differences)

5. **Components:**
   - **Hadoop:** Consists of HDFS for storage and YARN (Yet Another Resource Negotiator) for resource management, along with the MapReduce programming model. It also has an ecosystem of tools like Hive, Pig, HBase, and more.
   - **Spark:** Includes Spark Code (the engine for distributed processing), and higher-level components like Spark SQL for structured data processing, Spark Streaming for real-time data streams, MLlib for machine learning, and GraphX for graph processing.

Spark vs Hadoop (Differences)

6. **Streaming:**
   - **Hadoop:** Originally designed for batch processing and not well-suited for real-time streaming data. However, Apache Storm and Apache Flink have been used alongside Hadoop to handle real-time data streams.
   - **Spark:** Natively supports real-time streaming with Spark Streaming, which can process live data streams and integrate them with batch and interactive processing.

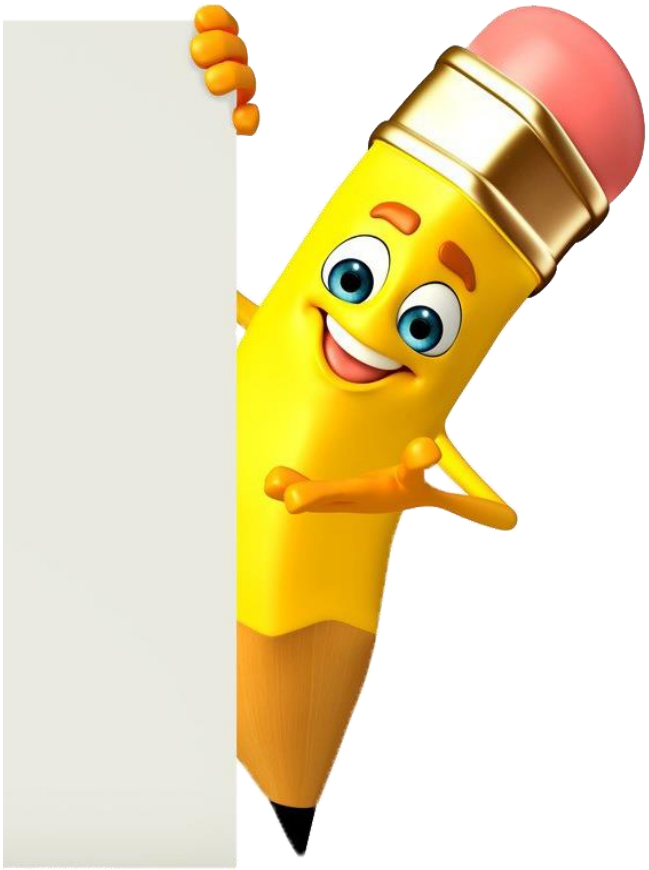
Spark vs Hadoop (Differences)

6. **Use Cases:**
   - **Hadoop:** Often used for large-scale batch processing tasks such as ETL (extract, transform, load) processes, data warehousing, and offline analytics. It's also common in environments where disk I/O is less of a concern, or existing Hadoop infrastructure is already in place.

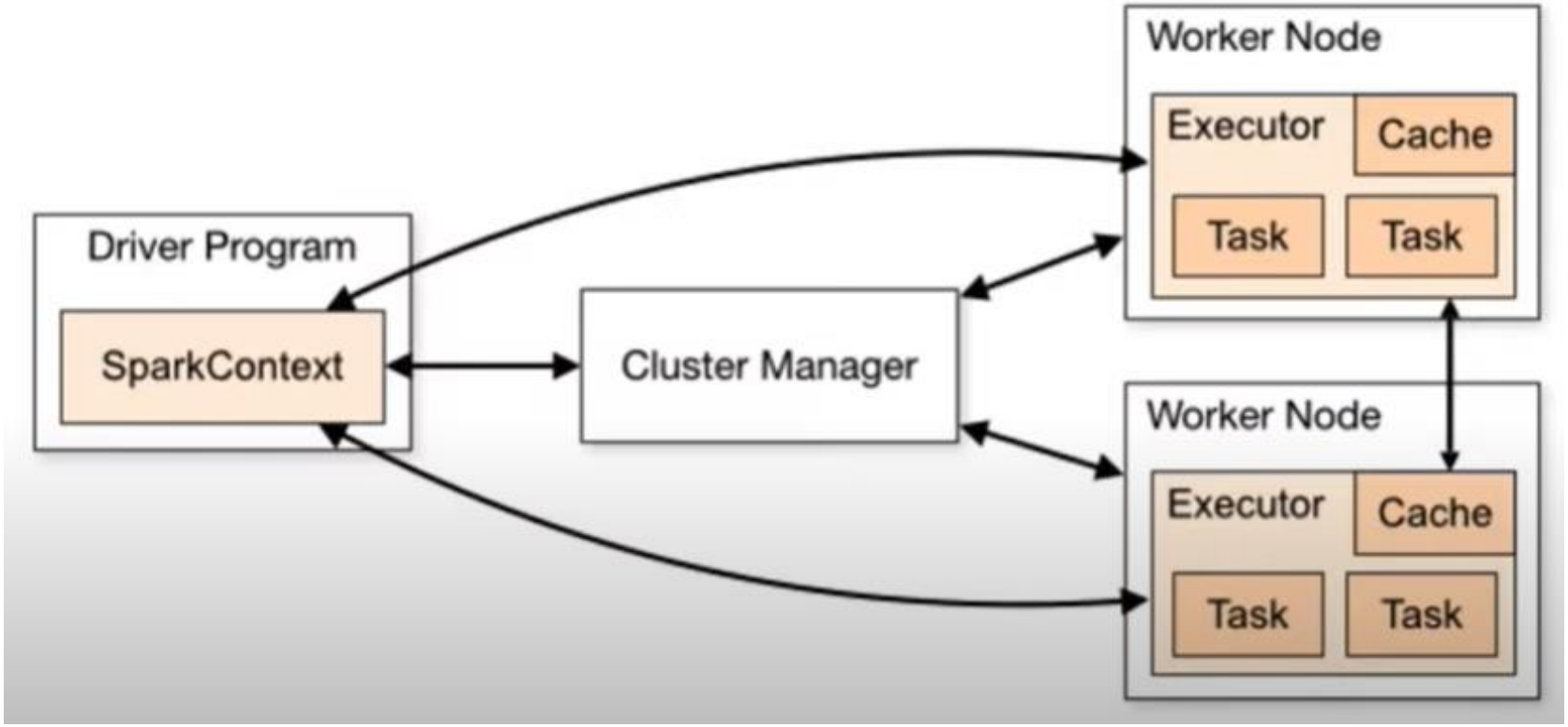
Spark vs Hadoop (Differences)

6. **Use Cases:**
   - **Spark:** Suitable for a variety of tasks, including real-time data processing, machine learning, interactive querying, and iterative processing tasks. Its flexibility and performance make it a good choice for modern big data applications that require fast processing and a unified analytics engine.

Spark Architecture



Spark Architecture



https://www.javatpoint.com/apache-spark-architecture

Spark Architecture

- Spark driver is the main process of a Spark Application. It is the master node that controls the Spark application's execution, and it maintains all of the states of the Spark cluster by creating a SparkContext.

Spark Architecture

- The Spark driver is responsible to:
  - Translates the user's code into Directed Acyclic Graph (DAG) of stages.
  - Schedules tasks on the cluster nodes.
  - Manages application lifecycle and resource allocation.
  - Collects and aggregates results from executors.

Spark Architecture

- Cluster Manager is responsible for maintaining a cluster of machines that will run the user's Spark Application(s).
- The role of the cluster manager is to allocate resources across applications.
- Spark currently supports different types of cluster managers:
  - Hadoop YARN, Apache Mesos, Kubernetes, or Spark's standalone cluster manager.

Spark Architecture

- Worker Node is a slave node. Its role is to run the application code in the cluster.
- An executor is a process launched for an application on a worker node.
  - It runs tasks and keeps data in memory or disk storage across them.
  - It reads and writes data to the external sources.

Workflow of Spark Architecture

- A user submits a Spark application using the SparkContext in their code.
- The SparkContext connects to the Cluster Manager to request resources for the application.
- The Cluster Manager allocates resources (CPU and memory) across the cluster nodes.
- Executors are launched on the worker nodes with the allocated resources.

Workflow of Spark Architecture

- The Driver translates high-level operations (transformation and actions) into a DAG of stages.
- Each state is a set of tasks that can be executed in parallel based on data partitioning.

Workflow of Spark Architecture

- The Driver schedules tasks on executors, optimizing for data locality and resource efficiency.
- The tasks are executed in stages, where each stage processes a subset of the data.

Workflow of Spark Architecture

- The Executors run the tasks assigned by the Driver. They process data, apply transformations, and perform actions.
- The Driver collects results from the tasks.
- Intermediate data can be stored in memory for fast access or on disk for fault tolerance.
- If an executor fails, the Driver reschedules the tasks on another executor.

Spark's Data Structures

RDD, Dataframe, and Dataset

Spark's Data Structures

- Apache Spark supports several data structures that facilitate efficient and scalable data processing.
- The following are the three primary data structures in Spark:
  - Resilient Distributed Datasets (RDD)
  - DataFrame
  - Dataset

RDD

- **RDDs** or Resilient Distributed Dataset (2011) is the fundamental data structure of the Spark.
- It is the collections of immutable (once created, the object cannot be altered) objects which is capable of storing the data partitioned across the multiple nodes of the cluster, and also allows them to do processing in parallel.

DataFrame

- Spark DataFrames are the distributed collection of the data points, but here, the data is organized into the named columns, similar to a table in a relational database.
- They allow developers to debug the code during the runtime which was not allowed with the RDDs.

DataFrame

- DataFrames can read and write the data into various formats like CSV, JSON, AVRO, HDFS, and HIVE tables.
- It is already optimized to process large datasets for most of the preprocessing tasks so that we do not need to write complex functions on our won.

Dataset

- Spark dataset is a combination of RDDs and DataFrame, providing the typed API advantages of RDDs with the optimizations and ease of use of DataFrame.
- If is fast as well as provides a type-safe interface. Type safety means that the compiler will validate the data types of all the columns in the dataset while compilation only and will throw an error if there is any mismatch in the data types.