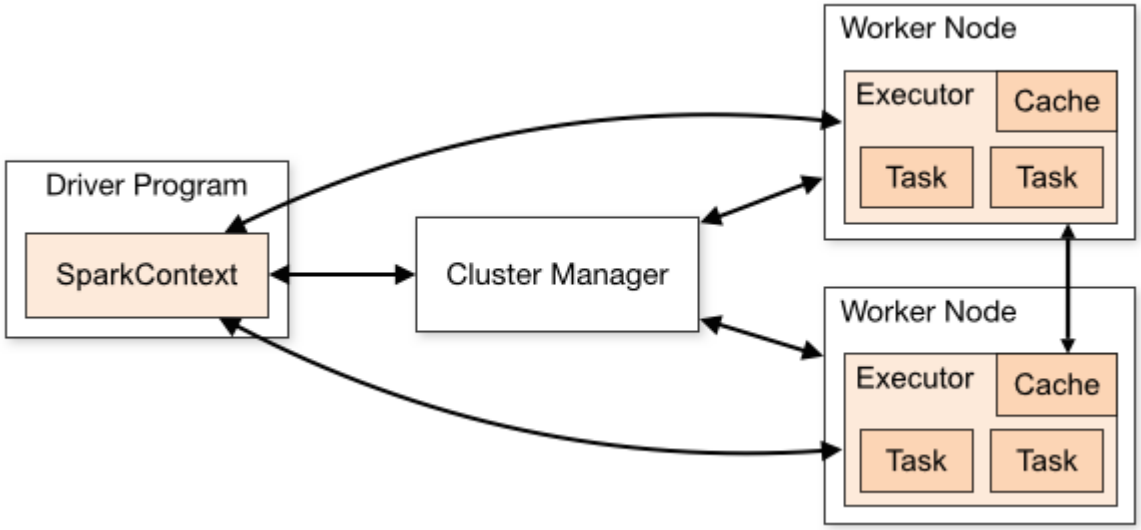
Explain Spark Architecture?

Spark follows Master Slave Architecture

- ➔ **Spark Master**
- ➔ **Spark Worker Node**



Spark Architecture Applications

1. **Spark Driver Program**
2. **SparkContext or SparkSession**
3. **Cluster Manager**
4. **Executors**
5. **Task**
6. **Shuffle**
7. **Result**

Driver Program

- ➔ It acts like a main method which will create the spark context.
- ➔ The First method which will be created.
- ➔ Coordinates and executes Spark applications.
- ➔ Runs on master node.
- ➔ Creates the execution plan.
- ➔ Distributes and coordinate tasks across worker nodes.
- ➔ Tracks the progress and status of tasks.
- ➔ It handles failures and retires for fault tolerance.

Spark Context or Spark Session

- ➔ It is the entry point to spark core of spark application.
- ➔ It will connect our spark application with execution environment or cluster.
- ➔ Interface for creating RDD's and executing Spark operations.
- ➔ It manages the distribution of tasks, resource allocation, and fault tolerance mechanisms.
- ➔ Provides a unified programming API for various programming languages. E.g., Scala, java, python, and R.
- ➔ Handles the configuration of Spark application properties and environment settings.
- ➔ Controls the parallelism and data partitioning, optimizing data processing across multiple nodes in a cluster.
- ➔ It enables the caching and persistence of intermediate data in memory or on disk.
- ➔ Integrates with various data sources, file systems, and third-party libraries (diverse formats).
- ➔ Monitoring management for Spark applications, job progress, resource usage, and execution metrics.
- ➔ It creates job execution graphs, manages partitions, scheduling etc...
- ➔ Before Spark 2.0.0 Spark context was used separately
- ➔ After Spark 2.0.0 + Spark session does all the thing.

Cluster Manager

- ➔ Used for managing Nodes and assigning tasks.
- ➔ Responsible for acquiring resources for application.
- ➔ Allows for efficient use of resources.

Types of Cluster Managers

- ➔ Standalone: Built in for testing and development.
- ➔ Apache Mesos: General purpose able to run multiple applications frameworks on a shared resource pool.
- ➔ Hadoop YARN: Part of a Hadoop ecosystem, to manage resources.
- ➔ Kubernetes: Container Orchestration used for managing clusters.

Executors

- ➔ Responsible for executing tasks.
- ➔ Through lifetime of the application
- ➔ Own JVM process.
- ➔ Executing subset of tasks.
- ➔ More Executors -> parallelism -> better performance.

Task

- ➔ The smallest unit of execution is known as Task.
- ➔ Each task will process one data partition.
- ➔ One task will be executing on one CPU core.
- ➔ One executor can run multiple tasks equal to the number of CPU cores.

Actions and Transformations in Spark

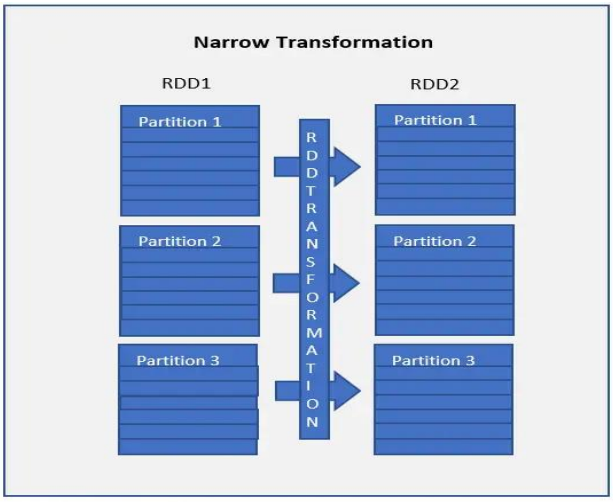
Transformations

- ➔ It is a function that produces a new RDD from existing RDD. It takes RDD as an input and generates new RDD as an output.

- ➔ There are two types of transformations.
    - Narrow Transformation
    - Wide Transformation

Narrow Transformation (No data Shuffling among the partitions)

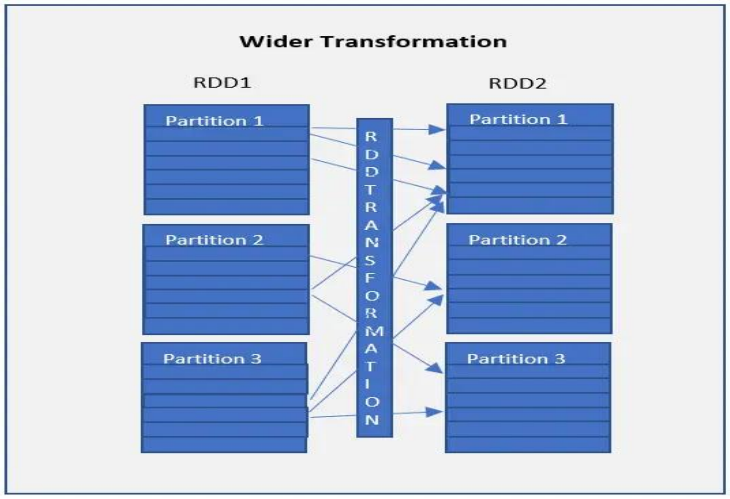
"Narrow Transformation" refers to an operation on a dataset where each input partition contributes only to a small number of output partitions. Narrow transformations do not require shuffling or reorganizing of data across partitions because the operation can be applied locally within each partition.



- ➔ Map()
- ➔ Flatmap()
- ➔ Filer()
- ➔ Union()
- ➔ Sample()

Wide Transformation (Data shuffling)

"Wide Transformation" involves operations on data that may require shuffling and redistribution of data across partitions. Unlike narrow transformations, which can be applied independently within each partition, wide transformations involve operations that require coordination and communication among partitions to produce the result.



All the elements that are required to compute the records in a single partition may live in many partitions of the parent RDD.

- ➔ Reduce by key
- ➔ Group by key
- ➔ Join
- ➔ Cartesian join
- ➔ Repartition
- ➔ Coalesce

Actions in Spark

Actions are operations that trigger the execution of the computation plan defined by transformations on distributed datasets (RDDs, DataFrames, or Datasets).

Actions instruct Spark to perform computations and return results either to the driver program or to external storage systems.

Action is one way of sending data from the executor to the driver.

- ➔ Count()
- ➔ Collect()
- ➔ Take(n)
- ➔ Top()
- ➔ Reduce()
- ➔ Aggregate()
- ➔ Foreach()

Clients

- ➔ Can be any machine from where we are sending our spark job for execution.
- ➔ Remote server, web interface like Data Stax, Hue.

Resource Manager

- ➔ It will create one container and start Application Master Service in it.
- ➔ Application master will request to resource manager for complete resource allocation or to start the required executors.
- ➔ Executors:
    - Virtual Entity which will have CPU, Memory, Network Bandwidth
    - Executors do the actual computation on partitioned data.

- ➔ The Resource Manager will create required number of executors for processing.
- ➔ Executors will start interacting with the Driver program to send the updates for Job progress and other meta data information.

What is RDD in Spark?

RDD – Resilient Distributed Dataset

- ➔ RDD are the main logical data units in spark. They are distributed collection of objects, which are stored in memory or on disks of different machines on cluster.

Features of RDD

- ➔ Resilience: RDDs tract data lineage information (lineage Graph) to recover from failed state automatically. It is also known as fault-tolerance.
- ➔ Distributed: Data present in an RDD resides on multiple nodes.
- ➔ Lazy Evaluation: Data does not get loaded in an RDD even if you define it.
- ➔ Immutability: Data stored in an RDD is in the read-only mode. You cannot edit the data which is present in RDD.
- ➔ In-Memory Computation: An RDD stores any intermediate data that is generated in the RAM so that faster access can be provided.

Spark DataFrames

- ➔ Higher-level abstractions than RDD.
- ➔ Distributed collection of data organized into named columns (schema).
    - Like a table in a relational database.
- ➔ Simplifies data processing.
- ➔ Excellent for processing semi & structured data, such as CSV, JSON, Parquet, Avro and more.
- ➔ Support filters, aggregations, joins, etc.!
- ➔ It uses Spark Query Optimizer.

Spark Datasets

- ➔ Distributed collection of data.
- ➔ Higher abstraction on top of RDDs and DataFrames.

When to use what: RDD, DataFrames, Datasets

- ➔ **RDD are better for low-level.**
  - Unstructured data
  - No schema validation
- ➔ **DataFrames and Datasets are better for:**
  - Semi-structured data
  - Schema validation

Spark SQL

- ➔ The same execution engine is used.
- ➔ Developers can easily switch back and forth between different APIs.