07 Introduction to RDD

RDD (Resilient Distributed Dataset) is:

- The **fundamental data abstraction** in Spark
- An **immutable**, **distributed** collection of objects
- Spread across multiple nodes in a cluster

In simple words:

> RDD is a fault tolerant distributed collection of data that Spark can process in parallel.

Why RDD was introduced

Before Spark:

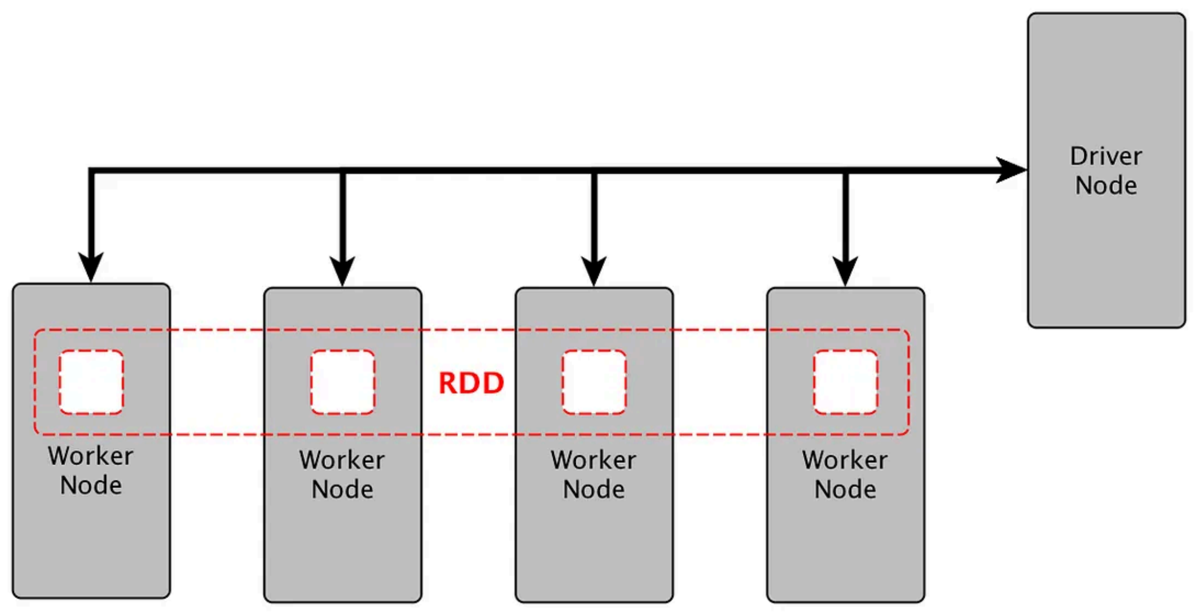
- Hadoop MapReduce
- Disk based processing
- Very slow for iterative jobs

RDD solved this by:

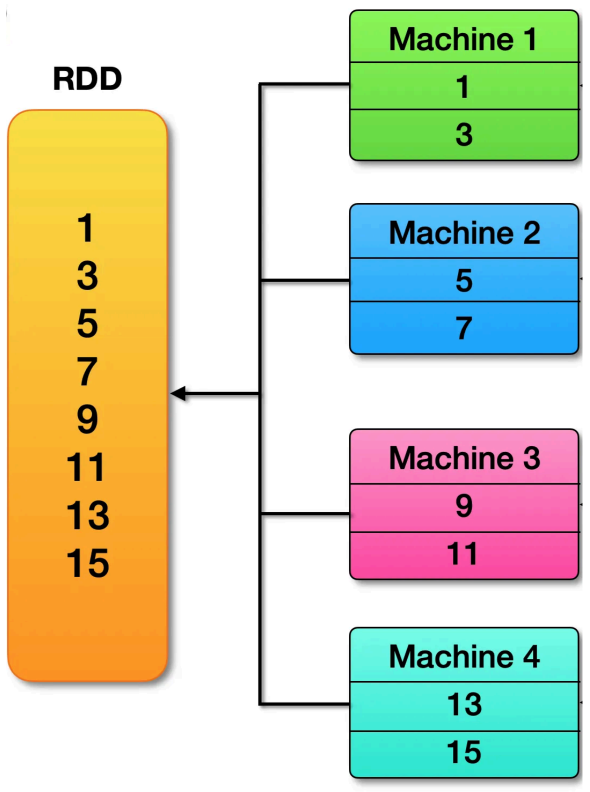
- Allowing **in memory processing**
- Supporting **reuse of intermediate data**
- Providing **fault tolerance without replication**

RDD is the basic unit which holds the data in Apache Spark.

RDD distributed across 4 machines/ worker node



The 4 partitions of RDD are distributes across cluster



RDD Demo

```
from pyspark.sql import SparkSession
# spark session is the enrty point to the cluster
spark = (
    SparkSession.builder
    .master("spark://spark-master:7077")
    .appName("rdd")
    .getOrCreate()
)

# create rdd from a file
orders_rdd = spark.sparkContext.textFile("/data/orders_300mb.csv")
orders_rdd.getNumPartitions()
orders_rdd.take(5)
orders_rdd = spark.sparkContext.textFile("/data/orders_40mb.csv")
orders_rdd.collect()

# create rdd using parallalize
data = [1,2,3,4,5,6,7,8,9,10,11,12]
rdd = spark.sparkContext.parallelize(data)
rdd.getNumPartitions()
rdd.glom().collect()
spark.sparkContext.defaultParallelism # is same as number of cores

# top 5 states with highest total sales for delivered orders

rdd_orders = spark.sparkContext.textFile("/data/orders_300mb.csv")
rdd_split = rdd_orders.map(lambda x:x.split(","))
rdd_filtered = rdd_split.filter(lambda x : x[5] == "DELIVERED")
rdd_map = rdd_filtered.map(lambda x : (x[6], float(x[3])*float(x[7])) )
rdd_final = rdd_map.reduceByKey(lambda x,y : x+y)
rdd_final.sortBy(lambda x : x[1] , ascending=False).take(5)
```

Key properties of RDD (important)

1️⃣ Distributed

- Data is split into **partitions**
- Each partition is processed independently
- Enables parallelism

Parallelism = number of partitions

2️⃣ Immutable

- RDDs cannot be changed
- Every operation creates a **new RDD**

Why this matters:

- Easier fault tolerance
- Predictable execution

3️⃣ Lazy evaluation

- RDD operations are **not executed immediately**
- Execution starts only when a **result is required**

This allows Spark to:

- Optimize execution
- Combine operations efficiently

4️⃣ Resilient (fault tolerant)

- Spark does not replicate RDD data
- Instead, it stores **how the RDD was created**

If a partition is lost:

- Spark recomputes it automatically

This is called **lineage/DAG**.

Spark Execution(How does spark work) :

let say i have a files orders.csv in s3/ hdfs

sudo code

rdd1 = load(orders.csv)

rdd2 = rdd1.map → some transformation

rdd3 = rdd2.filter → another transformation

rdd3.collect()

each new RDD will be in memory

There are 2 kind of operations in Apache Spark:

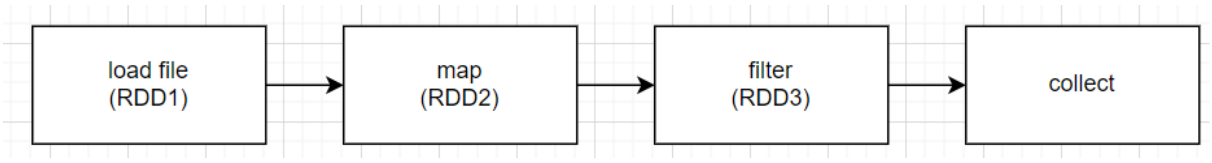
1- Transformation

2- Action

Transformation are lazy but actions are not .

All transformations becomes part of execution plan and executed only when a action is called

A simple DAG: Directed Acyclic Graph



RDD : Resilient , Distributed , Dataset

Resilient : Resilient to failures →if I lose RDD3 then it can be again recreated from RDD2.

RDDs are immutable ..you always create a new RDD , it helps you to get resiliency.

→ why laziness is good

Why transformations are lazy ?

example 1:

rdd1 = load(orders_1gb.csv)

rdd1.take(10) → collect 10 records

if transformations are not lazy then rdd1 will load 1 gb of data in memory

as they are lazy so when you call the action then it will bring only 10 records from storage to memory

example 2:

rdd1 = load(orders_1gb.csv)

rdd2 = rdd1.map() > sales = price * quantity

rdd3 = rdd3.filter (category = Technology) → 100 mb

rdd3.collect() ,

spark with apply the filter first and then apply the map transformation