**What is RDD (Resilient Distributed Dataset)?**

RDD (Resilient Distributed Dataset) is a fundamental data structure of Spark and it is the primary data abstraction in Apache Spark and the Spark Core. RDDs are fault-tolerant, immutable distributed collections of objects, which means once you create an RDD you cannot change it. Each dataset in RDD is divided into logical partitions (logical group like: year wise, country wise etc.), which can be computed on different nodes(host or server) of the cluster (group of nodes).

DS: Data Structure

{id:number, name:string,gender:string}

In other words, RDDs are a collection of objects similar to collections in Scala, with the difference being RDD is computed on several JVMs scattered across multiple physical servers also called nodes in a cluster while a Scala collection lives on a single JVM.

We will discuss about RDD operations available on RDDs, such as map, filter, and persist etc using Scala example. In addition, this tutorial also explains Pair RDD functions which operate on RDDs of key-value pairs such as groupByKey and join etc.

## RDD Advantages

– In-Memory Processing  
– Immutability  
– Fault Tolerance  
– Lazy Evolution  
– Partitioning  
– Parallelize

## Limitations

Spark RDDs are not much suitable for applications that make updates to the state store such as storage systems for a web application. For these applications, it is more efficient to use systems that perform traditional update logging and data checkpointing, such as databases. The goal of RDD is to provide an efficient programming model for batch analytics and leave these asynchronous applications.

## RDD Creation

RDD’s are created primarily in two different ways, first parallelizing an existing collection and secondly referencing a dataset in an external storage system (FS, HDFS, S3 and many more).

Before we look into examples, first let’s initialize SparkSession using the builder pattern method defined in SparkSession class. While initializing, we need to provide the master and application name as shown below.

val spark:SparkSession = SparkSession.builder()

.master("local[1]")

.appName("SparkByExamples.com")

.getOrCreate()

Copy

#### Using sparkContext.parallelize()

sparkContext.parallelize is used to parallelize an existing collection in your driver program. This is a basic method to create RDD and used mainly while POC’s or prototyping and it required all data to be present on the driver program prior to creating RDD hence it is not most used for production applications.

//Create RDD from parallelize

val dataSeq = Seq(("Java", 20000), ("Python", 100000), ("Scala", 3000))

val rdd=spark.sparkContext.parallelize(dataSeq)

Copy

For production applications, we mostly create RDD by using external storage systems like HDFS, S3, HBase e.t.c. To make it simple for this Spark tutorial we are using files from the local system and create RDD.

#### Using sparkContext.textFile()

Using textFile() method we can read a text (.txt) file into RDD.

//Create RDD from external Data source

val rdd2 = spark.sparkContext.textFile("/path/textFile.txt")

Copy

#### Using sparkContext.wholeTextFiles()

wholeTextFiles() function returns a PairRDD with the key being the file path and value being file content.

//Reads entire file into a RDD as single record.

val rdd3 = spark.sparkContext.wholeTextFiles("/path/textFile.txt")

Copy

Besides using text files, we can also create RDD from CSV file, JSON and more formats.

#### Using sparkContext.emptyRDD

Using emptyRDD() method on sparkContext we can create an RDD with no data. This method creates an empty RDD with no partition.

//Creates empty RDD with no partition

val rdd = spark.sparkContext.emptyRDD // creates EmptyRDD[0]

val rddString = spark.sparkContext.emptyRDD[String] // creates EmptyRDD[1]

Copy

#### Creating empty RDD with partition

Some times we may need to write an empty RDD to files by partition, In this case, you should create an empty RDD with partition.

//Create empty RDD with partition

val rdd2 = spark.sparkContext.parallelize(Seq.empty[String])

Copy

## RDD Parallelize and repartition

When we use parallelize() or textFile() or wholeTextFiles() methods of SparkContxt to initiate RDD, it automatically splits the data into partitions based on resource availability.

**getNumPartitions** – Returns a number of partitions our dataset split into. Any transformations applied on RDD execute parallelly. Spark will run one task for each partition of the cluster.

println("initial partition count:"+rdd.getNumPartitions)

//Outputs: initial partition count:2

Copy

**Set parallelize manually** – We can also set a number of partitions manually, all, we need is, to pass a number of partitions as the second parameter to these functions for example  sparkContext.parallelize(dataSeq, 10)).

**Repalallize using repartition and coalesce –** Some times we may need to repartition the RDD, Spark provides two ways to repartition; first using repartition() method which shuffles data from all nodes also called full shuffle and second coalesce() method which shuffle data from minimum nodes, for examples if you have data in 4 partitions and doing coalesce(2) moves data from just 2 nodes.

Both of the functions take the number of partitions to repartition rdd as shown below.  Note that repartition() method is a very expensive operation as it shuffles data from all nodes in a cluster.

val reparRdd = rdd.repartition(4)

println("re-partition count:"+reparRdd.getNumPartitions)

//Outputs: "re-partition count:4

Copy

**Note:** repartition() or coalesce() methods also returns a new RDD.

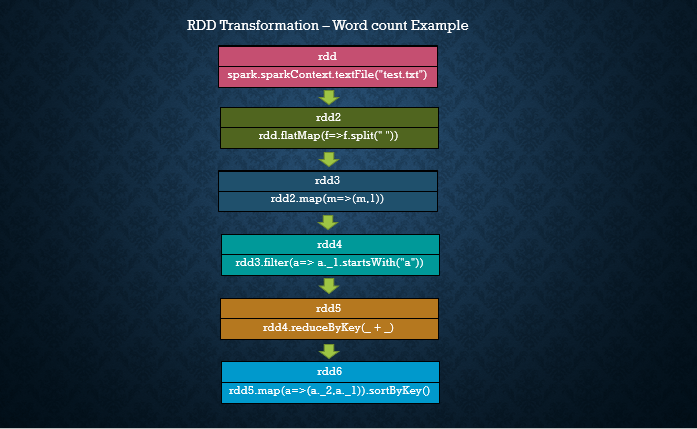
## RDD Operations

**RDD transformations –** Transformations are lazy operations, instead of updating an RDD, these operations return another RDD.  
**RDD actions –** operations that trigger computation and return RDD values.

### RDD Transformations with example

Transformations on Spark RDD returns another RDD and transformations are lazy meaning they don’t execute until you call an action on RDD. Some transformations on RDD’s are flatMap, map, reduceByKey, filter, sortByKey and return new RDD instead of updating the current.

In this Spark RDD Transformation tutorial, I will explain transformations using the word count example. The below image demonstrates different RDD transformations we going to use.

Word count spark RDD transformations

First, create an RDD by reading a text file. The text file used here is available at the GitHub project. And, the scala example I am using in this tutorial is available at GitHub project

val rdd:RDD[String] = spark.sparkContext.textFile("src/main/scala/test.txt")

Copy

**flatMap**– flatMap() transformation flattens the RDD after applying the function and returns a new RDD. On the below example, first, it splits each record by space in an RDD and finally flattens it. Resulting RDD consists of a single word on each record.

val rdd2 = rdd.flatMap(f=>f.split(" "))

Copy

**map**– map() transformation is used the apply any complex operations like adding a column, updating a column e.t.c, the output of map transformations would always have the same number of records as input.

In our word count example, we are adding a new column with value 1 for each word, the result of the RDD is PairRDDFunctions which contains key-value pairs, word of type String as Key and 1 of type Int as value. For your understanding, I’ve defined rdd3 variable with type.

val rdd3:RDD[(String,Int)]= rdd2.map(m=>(m,1))

Copy

**filter** – filter() transformation is used to filter the records in an RDD. In our example we are filtering all words starts with “a”.

val rdd4 = rdd3.filter(a=> a.\_1.startsWith("a"))

Copy

**reduceByKey** – reduceByKey() merges the values for each key with the function specified. In our example, it reduces the word string by applying the sum function on value. The result of our RDD contains unique words and their count.

val rdd5 = rdd4.reduceByKey(\_ + \_)

Copy

**sortByKey** – sortByKey() transformation is used to sort RDD elements on key. In our example, first, we convert RDD[(String,Int]) to RDD[(Int, String]) using map transformation and apply sortByKey which ideally does sort on an integer value. And finally, foreach with println statements returns all words in RDD and their count as key-value pair

val rdd6 = rdd5.map(a=>(a.\_2,a.\_1)).sortByKey()

//Print rdd6 result to console

rdd6.foreach(println)

Copy

Last statement foreach on rdd print the count of each word. Please refer to this page for the full list of RDD transformations.

### RDD Actions with example

RDD Action operation returns the raw values from an RDD. In other words, any RDD function that returns non RDD[T] is considered as an action.

In this Spark RDD Action tutorial, we will continue to use our word count example, the last statement foreach() is an action that returns all data from an RDD and prints on a console. let’s see some more action operations on our word count example.

**count** – Returns the number of records in an RDD

//Action - count

println("Count : "+rdd6.count())

Copy

**first** – Returns the first record.

//Action - first

val firstRec = rdd6.first()

println("First Record : "+firstRec.\_1 + ","+ firstRec.\_2)

Copy

**max** – Returns max record.

//Action - max

val datMax = rdd6.max()

println("Max Record : "+datMax.\_1 + ","+ datMax.\_2)

Copy

**reduce** – Reduces the records to single, we can use this to count or sum.

//Action - reduce

val totalWordCount = rdd6.reduce((a,b) => (a.\_1+b.\_1,a.\_2))

println("dataReduce Record : "+totalWordCount.\_1)

Copy

**take** – Returns the record specified as an argument.

//Action - take

val data3 = rdd6.take(3)

data3.foreach(f=>{

println("data3 Key:"+ f.\_1 +", Value:"+f.\_2)

})

Copy

**collect** – Returns all data from RDD as an array. Be careful when you use this action when you are working with huge RDD with millions and billions of data as you may run out of memory on the driver.

//Action - collect

val data = rdd6.collect()

data.foreach(f=>{

println("Key:"+ f.\_1 +", Value:"+f.\_2)

})

Copy

**saveAsTextFile** – Using saveAsTestFile action, we can write the RDD to a text file.

rdd6.saveAsTextFile("/tmp/wordCount")

Copy

Note: Please refer to this page for a full list of RDD actions.

## Types of RDD

**PairRDDFunctions or PairRDD** – Pair RDD is a key-value pair This is mostly used RDD type,

**ShuffledRDD –**

**DoubleRDD –**

**SequenceFileRDD –**

**HadoopRDD –**

**ParallelCollectionRDD –**