Apache Spark 2 using Scala - Data Processing – Overview

Apache Spark 2 – Data Processing Overview (Scala)

Spark's data processing consists of:

- **Transformations** → Lazy operations that define a new dataset from an existing one (don't execute immediately).
- **Actions** → Trigger execution and return results to the driver or save them to storage.

1. Spark Transformations

Transformation	Description	Scala Example
map	Applies a function to each element and returns a new RDD/DataFrame.	val rdd2 = rdd.map(x => x * 2)
flatMap	Like map, but flattens the results (one input → many outputs).	val words = lines.flatMap(line => line.split(" "))
filter	Keeps only elements satisfying a condition.	val even = rdd.filter(_ % 2 == 0)
mapPartitions	Runs a function on each partition (more efficient for batch ops).	val result = rdd.mapPartitions(iter => iter.map(_ * 2))
mapPartitionsWithIndex	Same as above but also gets partition index.	val indexed = rdd.mapPartitionsWithIndex((idx, iter) => iter.map(x => (idx, x)))
distinct	Removes duplicates.	val unique = rdd.distinct()
sample	Randomly samples elements.	val sampleRDD = rdd.sample(false, 0.1)
union	Combines two RDDs.	val unionRDD = rdd1.union(rdd2)
intersection	Elements common to both RDDs.	val interRDD = rdd1.intersection(rdd2)
subtract	Elements in one RDD but not in the other.	val diff = rdd1.subtract(rdd2)

cartesian	Cartesian product between two RDDs.	val cart = rdd1.cartesian(rdd2)
groupBy	Groups elements using a function.	val grouped = rdd.groupBy(_ % 2)
groupByKey	Groups key-value pairs by key (expensive).	val grouped = rdd.groupByKey()
reduceByKey	Merges values for each key using an operation (faster than groupByKey).	val reduced = rdd.reduceByKey(_ + _)
aggregateByKey	Aggregates values with separate seqOp and combOp.	val agg = rdd.aggregateByKey(0)(_+_, _+_)
combineByKey	Most general form of aggregation.	val combined = rdd.combineByKey(v => (v,1), (c:(Int,Int),v)=> (c._1+v, c._2+1), (c1:(Int,Int),c2:(Int,Int))=> (c1._1+c2._1, c1._2+c2._2))
sortByKey	Sorts key-value RDD by key.	val sorted = rdd.sortByKey()
join	Inner join between two key-value RDDs.	val joined = rdd1.join(rdd2)
leftOuterJoin	Left join between RDDs.	val leftJoin = rdd1.leftOuterJoin(rdd2)
rightOuterJoin	Right join between RDDs.	val rightJoin = rdd1.rightOuterJoin(rdd2)
cogroup	Groups values from multiple RDDs sharing the same key.	val cogrouped = rdd1.cogroup(rdd2)
repartition	Changes number of partitions (shuffle).	val repart = rdd.repartition(4)
coalesce	Reduces partitions without full shuffle.	val coalesced = rdd.coalesce(2)
pipe	Sends RDD elements to an external process.	val piped = rdd.pipe("wc -l")
zip	Combines two RDDs element-wise.	val zipped = rdd1.zip(rdd2)

2. Spark Actions

Action	Description	Scala Example
collect	Returns all elements to driver (careful with large data).	val data = rdd.collect()
count	Returns number of elements.	val cnt = rdd.count()
first	Returns the first element.	val firstVal = rdd.first()
take(n)	Returns first n elements.	val few = rdd.take(5)
takeOrdered(n)	Returns first n elements in order.	val ordered = rdd.takeOrdered(5)
top(n)	Returns top n elements in descending order.	val topVals = rdd.top(5)
reduce	Aggregates elements using a function.	val sum = rdd.reduce(_ + _)
fold	Same as reduce but with zero value.	val sum = rdd.fold(0)(_ + _)
aggregate	Aggregates with separate seqOp and combOp.	val agg = rdd.aggregate(0)(_ + _, _ + _)
foreach	Runs a function on each element (no return).	rdd.foreach(println)

countByKey	Returns count of values per key.	val counts = rdd.countByKey()
countByValue	Returns counts of each element.	val counts = rdd.countByValue()
saveAsTextFile	Saves RDD to text file.	rdd.saveAsTextFile("output")
saveAsSequenceFile	Saves RDD as Hadoop sequence file.	rdd.saveAsSequenceFile("output")
saveAsObjectFile	Saves RDD as serialized objects.	rdd.saveAsObjectFile("output")

Example Program – Transformations + Actions

```
import org.apache.spark.sql.SparkSession
object SparkTransformationsActions {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("Spark Transformations and Actions")
      .master("local[*]")
      .getOrCreate()
    val sc = spark.sparkContext
    // Sample RDD
    val rdd = sc.parallelize(List(1, 2, 3, 4, 5, 6, 2, 4))
    // Transformation: Filter even numbers and double them
    val transformedRDD = rdd.filter(_ % 2 == 0).map(_ * 2).distinct()
    // Action: Collect and print
    println("Transformed Data: " + transformedRDD.collect().mkString(", "))
    // Transformation: Create pair RDD
    val pairRDD = rdd.map(x => (x, 1))
    // Reduce by key
    val counts = pairRDD.reduceByKey(_ + _)
    // Action: Display word counts
    println("Element Counts:")
    counts.collect().foreach(println)
    spark.stop()
  }
}
```

✅ **Key Points to Remember**

- Transformations are **lazy** → nothing happens until an action is called.
- Use **reduceByKey** instead of groupByKey for performance.
- **collect()** should be avoided for large datasets.
- Partitioning and persistence can hugely improve performance.