**Chapter-4:**

**Key/value pairs in spark:**

**Paired RDDs:**

Spark provides special operations on RDDs containing key/value pairs. These RDDs are called pair RDDs.

Creating a pair RDD using the first word as the key in Scala

val pairs = lines.map(x => (x.split(" ")(0), x))

**Transformations using one pair RDD:**

rdd.reduceByKey( (x, y) => x + y)

rdd.groupByKey()

rdd.mapValues(x => x+1)

rdd.flatMapValues(x => (x to 5))

rdd.keys()

rdd.values()

rdd.sortByKey()

**Transformations using two pair RDD:**

rdd.subtractByKey(other)

rdd.rightOuterJoin(other)

rdd.join(other)

rdd.leftOuterJoin(other)

**Word count job in scala:**

val input = sc.textFile("s3://...")

val words = input.flatMap(x => x.split(" "))

val result = words.map(x => (x, 1)).reduceByKey((x, y) => x + y)

**Tuning the level of parallelism:**

Every RDD has a fixed number of partitions that determine the degree of parallelism to use when executing operations on the RDD.

We can increase the number of partitions of an RDD. This we call repartitioning.

val data = Seq(("a", 3), ("b", 4), ("a", 1))

sc.parallelize(data).reduceByKey((x, y) => x + y) // Default parallelism sc.parallelize(data).reduceByKey((x, y) => x + y, 10) // Custom parallelism

Spark also has an optimized version of repartition() called coalesce() that allows avoiding data movement, but only if you are decreasing the number of RDD partitions.

To check the size of the RDD in scala/java

rdd.partitions.size()

**Actions on paired RDDs:**

rdd.collectAsMap()--- collect the result as map

rdd.countByKey()--- returns number of values related to each key

rdd.lookup(key) --- gives values related to that key

**Data Partitioning**