SPARK PAIR RDD HANDS ON LAB

By www.HadoopExam.com

Note: These instructions should be used with the HadoopExam Apache Spark: Professional Trainings. Where it is executed and you can do hands on with trainer.

PairRDDFunctions

Methods defined in this interface extension become available when the data items have a two component tuple structure. Spark will interpret the first tuple item (i.e. tuplename. 1) as the key and the second item (i.e. tuplename. 2) as the associated value.

reduceByKey : This function provides the well-known *reduce* functionality in Spark. Please note that any function *f* you provide, should be commutative in order to generate reproducible results.

Definition:

```
def reduceByKey(func: (V, V) => V): RDD[(K, V)]
def reduceByKey(func: (V, V) => V, numPartitions: Int): RDD[(K, V)]
def reduceByKey(partitioner: Partitioner, func: (V, V) => V): RDD[(K, V)]
```

Example:

```
val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2)
val b = a.map(x => (x.length, x))
b.reduceByKey(_ + _).collect
// res86: Array[(Int, String)] = Array((3,dogcatowlgnuant))


val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)
val b = a.map(x => (x.length, x))
b.reduceByKey(_ + _).collect

//res87: Array[(Int, String)] = Array((4,lion), (3,dogcat), (7,panther), (5,tigereagle))
```

foldByKey: Very similar to *fold*, but performs the folding separately for each key of the RDD. This function is only available if the RDD consists of two-component tuples.

Definition:

```
def foldByKey(zeroValue: V)(func: (V, V) => V): RDD[(K, V)]
def foldByKey(zeroValue: V, numPartitions: Int)(func: (V, V) => V): RDD[(K, V)]
def foldByKey(zeroValue: V, partitioner: Partitioner)(func: (V, V) => V): RDD[(K, V)]
```

Example:

```
val a = sc.parallelize(List("dog", "cat", "owl", "gnu", "ant"), 2)
val b = a.map(x => (x.length, x))
b.foldByKey("")(_ + _).collect
//res84: Array[(Int, String)] = Array((3,dogcatowlgnuant)

val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "panther", "eagle"), 2)
val b = a.map(x => (x.length, x))
b.foldByKey("")(_ + _).collect
//res85: Array[(Int, String)] = Array((4,lion), (3,dogcat), (7,panther), (5,tigereagle))
```

combineByKey: Very efficient implementation that combines the values of a RDD consisting of two-component tuples by applying multiple aggregators one after another.

Definition:

```
def combineByKey[C](createCombiner: V => C, mergeValue: (C, V) => C,
mergeCombiners: (C, C) => C): RDD[(K, C)]

def combineByKey[C](createCombiner: V => C, mergeValue: (C, V) => C,
mergeCombiners: (C, C) => C, numPartitions: Int): RDD[(K, C)]

def combineByKey[C](createCombiner: V => C, mergeValue: (C, V) => C,
mergeCombiners: (C, C) => C, partitioner: Partitioner, mapSideCombine: Boolean =
true, serializerClass: String = null): RDD[(K, C)]
```

Example:

```
val a = sc.parallelize(List("dog","cat","gnu","salmon","rabbit","turkey","wolf","bear","bee"), 3)
val b = sc.parallelize(List(1,1,2,2,2,1,2,2,2), 3)
val c = b.zip(a)
val d = c.combineByKey(List(_), (x:List[String], y:String) => y :: x, (x:List[String], y:List[String]) => x ::: y)
d.collect

//res16: Array[(Int, List[String])] = Array((1,List(cat, dog, turkey)), (2,List(gnu, rabbit, salmon, bee,
bear, wolf)))
```

groupByKey: Very similar to *groupBy*, but instead of supplying a function, the key-component of each pair will automatically be presented to the partitioner.

Definition:

```
def groupByKey(): RDD[(K, Iterable[V])]
def groupByKey(numPartitions: Int): RDD[(K, Iterable[V])]
def groupByKey(partitioner: Partitioner): RDD[(K, Iterable[V])]
```

Example:

```
val a = sc.parallelize(List("dog", "tiger", "lion", "cat", "spider", "eagle"), 2)
val b = a.keyBy(_.length)
b.groupByKey.collect

//res11: Array[(Int, Seq[String])] = Array((4,ArrayBuffer(lion)), (6,ArrayBuffer(spider)),
(3,ArrayBuffer(dog, cat)), (5,ArrayBuffer(tiger, eagle)))
```