Spark notes from Learning Spark

Creating RDD

```
val lines = sc.textFile("/path/to/README.md")
```

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function Name	Purpose	Example	Result
map	Apply a function to each element in the RDD and return an RDD of the result	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
flatMap	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
filter	Return an RDD consisting of only elements which pass the condition passed to filter	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
distinct	Remove duplicates	`rdd.distinct()`	`{1, 2, 3}`
sample(withReplacement, fraction, [seed])	Sample an RDD	`rdd.sample(false, 0.5)`	non-deterministic

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function Name	Purpose	Example	Result
union	Produce an RDD contain elements from both RDDs	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
intersection	RDD containing only elements found in both RDDs	`rdd.intersection(other)`	`{3}`
subtract	Remove the contents of one RDD (e.g. remove training data)	`rdd.subtract(other)`	`{1, 2}`
cartesian	Cartesian product with the other RDD	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

```
val sum = rdd.reduce((x, y) => x + y)
```

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function Name	Purpose	Example	Result
collect()	Return all elements from the RDD	`rdd.collect()`	`{1, 2, 3, 3}`
count()	Number of elements in the RDD	`rdd.count()`	`4`
take(num)	Return num elements from the RDD	`rdd.take(2)`	`{1, 2}`
top(num)	Return the top num elements the RDD	`rdd.top(2)`	`{3, 3}`
takeOrdered(num)(ordering)	Return num elements based on providing ordering	`rdd.takeOrdered(2)(myOrdering)`	`{3, 3}`
takeSample(withReplacement, num, [seed])	Return num elements at random	`rdd.takeSample(false, 1)`	non-deterministic
reduce(func)	Combine the elements of the RDD together in parallel (e.g. sum)	`rdd.reduce((x, y) => x + y)`	`9`
fold(zero)(func)	Same as reduce but with the provided zero value	`rdd.fold(0)((x, y) => x + y)`	`9`
aggregate(zeroValue)(seqOp, combOp)	Similar to reduce but used to return a different type	`rdd.aggregate(0, 0)({case (x, y) => (y._1() + x, y._2() + 1)}, {case (x, y) => (y._1() + x._1(), y._2() + x._2())})`	`(9, 4)`
foreach(func)	Apply the provided function to each element of the RDD	`rdd.foreach(func)`	nothing

Working with Key-Value Pairs

Example 4-2. Scala create pair RDD using the first word as the key

```
input.map(x => (x.split(" ")(0), x))
```

Example 4-5. Scala simple filter on second element

```
pair.filter{case (x, y) => y.length < 20}
```

Example 4-8. Scala per key average with reduceByKey and mapValues

```
rdd.mapValues(x => (x, 1)).reduceByKey((x, y) => (x._1 + y._1, x._2 +
y._2))
```

Example 4-10. Scala word count example

```
val input = sc.textFile("s3://...")
val words = input.flatMap(x => x.split(" "))
val result = words.map(x => (x, 1)).reduceByKey((x, y) => x + y)
```

Example 4-13. Scala per-key average using combineByKey

```
val input = sc.parallelize(List(("coffee", 1),("coffee", 2),("panda",
4)))
val result = input.combineByKey(
   (v) => (v, 1),(acc: (Int, Int), v) => (acc._1 + v, acc._2 +
1),(acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 + acc2._1,
acc1._2 + acc2._2) ).map{ case (key, value) => (key, value._1 /
value._2.toFloat) }
result.collectAsMap().map(println(_))
```

Example 4-16. Scala reduceByKey with custom parallelism

```
val data = Seq(("a", 3), ("b", 4), ("a", 1))
sc.parallelize(data).reduceByKey(_ + _) // Default parallelism
sc.parallelize(data).reduceByKey(_ + _, 10) // Custom parallelism
```

Example 4-17. Scala shell inner join example

```
storeAddress = {(Store("Ritual"), "1026 Valencia St"),
(Store("Philz"), "748 Van Ness Ave"),(Store("Philz"), "3101 24th St"),
(Store("Starbucks"), "Seattle")}

storeRating = {(Store("Ritual"), 4.9), (Store("Philz"), 4.8))}

storeAddress.join(storeRating) = {(Store("Ritual"), ("1026 Valencia
St", 4.9)),(Store("Philz"), ("748 Van Ness Ave", 4.8)),
(Store("Philz"), ("3101 24th St", 4.8))}
```

Example 4-18. Scala shell leftOuterJoin / rightOuterJoin examples

```
storeAddress.leftOuterJoin(storeRating) = {(Store("Ritual"),("1026
Valencia St",Some(4.9))),(Store("Starbucks"),("Seattle",None)),
(Store("Philz"),("748 Van Ness Ave",Some(4.8))),
(Store("Philz"),("3101 24th St",Some(4.8)))}

storeAddress.rightOuterJoin(storeRating) =
{(Store("Ritual"),(Some("1026 Valencia St"),4.9)),
(Store("Philz"),(Some("748 Van Ness Ave"),4.8)), (Store("Philz"),
(Some("3101 24th St"),4.8))}
```

Table 4-1. Transformations on one Pair RDD (example ({(1, 2), (3, 4), (3, 6)}))

Function Name	Purpose	Example	Result
combineByKey(createCombiner, mergeValue, mergeCombiners, partitioner)	Combine values with the same key together	See combine by key example	
groupByKey()	Group together values with the same key	rdd.groupByKey()	{(1, [2]), (3, [4, 6])}
reduceByKey(func)	Combine values with the same key together	rdd.reduceByKey((x, y) => x + y)	{(1, 2), (3, 10)}
mapValues(func)	Apply a function to each value of a Pair RDD without changing the key	rdd.mapValues(x => x+1)	{(1, 3), (3, 5), (3, 7)}
flatMapValues(func)	Apply a function which returns an iterator to each value of a Pair RDD and for each element returned produce a key-value entry with the old key. Often used for tokenization.	rdd.flatMapValues(x => x.to(5))	{(1,2), (1,3), (1,4), (1,5), (3, 4), (3,5)}
keys()	Return an RDD of just the keys	rdd.keys()	{1, 3, 3}
values()	Return an RDD of just the values	rdd.values()	{2, 4, 6}
sortByKey()	Returns an RDD sorted by the key	rdd.sortByKey()	{(1, 2), (3, 4), (3, 6)}

Table 4-2. Transformations on two Pair RDD (example ({(1, 2), (3, 4), (3, 6)}) other ({(3, 9)})

Function Name	Purpose	Example	Result
subtractByKey	Remove elements with a key present in the other RDD	`rdd.subtractByKey(other)`	`{1, 2}`
`join`	Perform an inner join between two RDDs	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDS where the key must be present in the first RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDS where the key must be present in the other RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`
`cogroup`	Group together data from both RDDs sharing the same key	`rdd.cogroup(other)`	`{(1,([2],[])), (3, ([4, 6],[9]))}`

Table 4-3. Actions on Pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

countByKey()	**Count the number of elements for each key**	**rdd.countByKey()**	**{(1, 1), (3, 2)}**
`collectAsMap()`	Collect the result as a map to provide easy lookup	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key	`rdd.lookup(3)`	`[4, 6]`