Basic Spark 2 – Part 2

DSC 232R

1.2 getting information about an RDD

RDDs typically have hundreds of thousands of elements. It usually makes no sense to print out the content of a whole RDD. Here are some ways to get manageable amounts of information about an RDD.

Create an RDD of length n which is a repetition of the pattern 1,2,3,4

```
In [22]: n=1000000
         B=sc.parallelize([1,2,3,4]*int(n/4))
```

```
In [23]: #find the number of elements in the RDD
         B.count()

Out[23]: 1000000
```

In [24]:

```
# get the first few elements of an RDD
print('first element=',B.first())
print('first 5 elements = ',B.take(5))
```

```
first element= 1
first 5 elements =  [1, 2, 3, 4, 1]
```

1.2.1 Sampling an RDD

- RDDs are often very large.
- Aggregates, such as averages, can be approximated efficiently by using a sample.
- Sampling is done in parallel and requires limited computation.

```
In [9]: # get a sample whose expected size is m
        # Note that the size of the sample is different in different runs
        m=5.
        print('sample1=',B.sample(False,m/n).collect())
        print('sample2=',B.sample(False,m/n).collect())

        sample1= [3, 3, 3, 3, 3, 3, 2, 4, 3, 3]
        sample2= [1, 1, 1, 2, 2, 1]
```

1.2.3 Filtering an RDD

The method `RDD.filter(func)` Returns a new dataset formed by selecting those elements of the source on which func remains true.

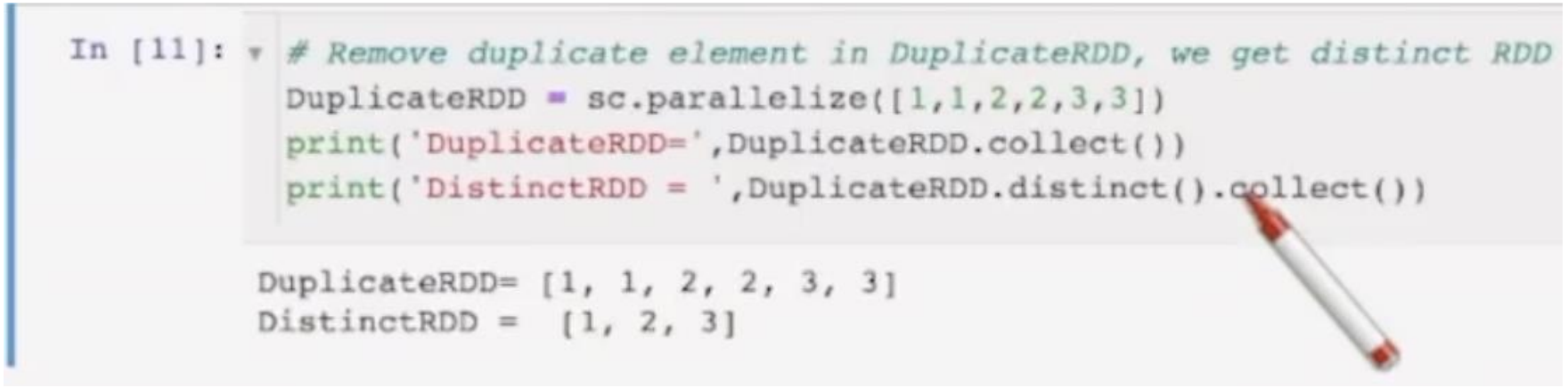
```
In [10]: print('the number of elements in B that are > 3 =',B.filter(lambda n: n > 3).count())

         the number of elements in B that are > 3 = 250000
```

1.2.4 Removing duplicate elements from an RDD

The method `RDD.distinct()` Returns a new dataset that contains the distinct elements of the source dataset.

This operation requires a shuffle in order to detect duplication across partitions.



1.2.5 flatmap an RDD

The method `RDD.flatMap(func)` is similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).

```
In [12]: text=["you are my sunshine","my only sunshine"]
         text_file = sc.parallelize(text)
         # map each line in text to a list of words
         print('map:',text_file.map(lambda line: line.split(" ")).collect())
         # create a single list of words by combining the words from all of the lines
         print('flatmap:',text_file.flatMap(lambda line: line.split(" ")).collect())

map: [['you', 'are', 'my', 'sunshine'], ['my', 'only', 'sunshine']]
flatmap: ['you', 'are', 'my', 'sunshine', 'my', 'only', 'sunshine']
```

1.2.6 Set operations

In this part, we explore set operations including union, intersection, subtract, cartesian in PySpark.

```
In [13]: rdd1 = sc.parallelize([1, 1, 2, 3])
         rdd2 = sc.parallelize([1, 3, 4, 5])
```

1. union(other)
   - Return the union of this RDD and another one.
   - Note that repetitions are allowed. The RDDs are **bags** not **sets**.
   - To make the result a set, use `.distinct`

```
In [14]: rdd2=sc.parallelize(['a','b',1])
         print('rdd1=',rdd1.collect())
         print('rdd2=',rdd2.collect())
         print('union as bags =',rdd1.union(rdd2).collect())
         print('union as sets =',rdd1.union(rdd2).distinct().collect())

rdd1= [1, 1, 2, 3]
rdd2= ['a', 'b', 1]
union as bags = [1, 1, 2, 3, 'a', 'b', 1]
union as sets = [1, 'a', 2, 3, 'b']
```

2. intersection(other)
    - Return the intersection of this RDD and another one. The output will not contain any duplicate elements, even if the input RDDs did. Note that this method performs a shuffle internally.

```
In [15]: rdd2=sc.parallelize([1,1,2,5])
         print('rdd1=',rdd1.collect())
         print('rdd2=',rdd2.collect())
         print('intersection=',rdd1.intersection(rdd2).collect())

rdd1= [1, 1, 2, 3]
rdd2= [1, 1, 2, 5]
intersection= [1, 2]
```

3. subtract(other,numPartitions=None)
    - Return each value in self that is not contained in other.

```
In [16]: print('rdd1=',rdd1.collect())
         print('rdd2=',rdd2.collect())
         print('rdd1.subtract(rdd2)=',rdd1.subtract(rdd2).collect())

         rdd1= [1, 1, 2, 3]
         rdd2= [1, 1, 2, 5]
         rdd1.subtract(rdd2)= [3]
```

4. cartesian(other)
    - Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a,b) where **a** is in **self** and **b** is in other.

```
In [17]: rdd1=sc.parallelize([1,1,2])
         rdd2=sc.parallelize(['a','b'])
         print('rdd1=',rdd1.collect())
         print('rdd2=',rdd2.collect())
         print('rdd1.cartesian(rdd2)=\n',rdd1.cartesian(rdd2).collect())

rdd1= [1, 1, 2]
rdd2= ['a', 'b']
rdd1.cartesian(rdd2)=
 [(1, 'a'), (1, 'b'), (1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
```

1.3 Summary

- Chaining: creating a pipeline of RDD operations.
- Counting, taking and sampling an RDD
- More transformations: `filter`, `distinct`, `flatmap`
- Set transformations: `union`, `intersection`, `subtract`, `cartesian`