Spark

First we create a RDD function: RDD is Resilient Distributed Datasets.

Its immutable once we create it but it can be transformed into what we want.

Creating an RDD

Data = xrange(1,1001)

Print data 🡪 shift + enter

Output: xrange(1,1001)

rdd = sc.parallelize (data) 🡪 shift + enter

rdd.filter(lambda x : x < 10 ).collect()

output: collects all the integer which is less than 10.

rdd.map(lambda x: x\*2).collect()

output: multiplies all the integer with a value of 2.

output : 1,2,3,4 -> 2,4,6,8

rdd.filter(lambda x:x%2 == 0) -> This divides the number X by 2 and returns the answer if the remainder is zero and number is fully divided by 2.

Output: 1,2,3,4 -> 2,4

We can also use map and filter in the same function .

sc.parallelize(data).map(lambda y : y-1).filter(lambda x : x < 10).collect()

print ‘the sum of 1 and 1 is {}’.format(1+1) 🡪 The {} sign is used to call the .format operator

output: ‘ the sum of 1 and 1 is 2’

x = 42

if x > 40:

print ‘ the sum of 1 and 2 is {}’.format(1+2)

output: ‘the sum of 1 and 2 is 3’

import datetime

print ‘ the current time is {}’.format(datetime.datetime.now())

def sub(value):

return (value -1)

sub(10)

output: 9

rdd2 = sc.parallelize([1,2,3,4,4,6,6,7,1)] -> Gives all the distinct value in the function .

rdd2.distinct().collect()

output: 1,2,3,4,4,6,6,7,1 -> 1,2,3,4,6,7

rdd = sc.parallelize([1,2,3])

rdd.map(lambda x :[x,x+5]).collect() -> Adds 5 to the x function .

output : ([1,2,3]) -> ([1,6] ,[2,7],[3,8])

rdd.flatMap(lambda x : [x ,x+5]) -> flattens and turns list into individual elements.

output : [1,2,3] -> [1,6,2,7,3,8]

creating a new RDD named lines :

lines = sc.textFile (“some file location “,4) -> It means its divided into 4 partitions

comments = lines.filter(isComment) -> isComment is a function here , filters RDD in files which are not comments

rdd = sc.parallelize([1,2,3]

rdd.reduce(lambda a,b : a\*b) -> mulitplies reduces the ouput as compared to input

output : 6 -> 1 times 2 times 3 = 6 .

rdd.take(2) -> returns first two elements of RDD

output: [1,2]

rdd.collect() 🡪 returns all the elements of the RDD as a list

output :[1,2,3]

rdd = sc.parallelize([5,3,1,2,4,5,6,6,7]).distinct() 🡪 this returns a RDD 5,3,1,2

rdd.takeOrdered(3, lambda s : -1 \* s) 🡪 returns the first 3 elements in decreasing value because of -1.

output: [5,3,2]

lines = sc.textFile(‘some file with location’ ,4) 🡪 divided into 4 partitons

comments = lines.filter(isComment) 🡪 Reads the number of comments in line

print lines.count(),comments.count() 🡪 prints the number of lines and comments.

To avoid reloading data we can use cache() .

Lines = sc.textFile(‘some file location ‘,4)

Lines.cache() 🡪saves and you don’t have to recompute

Comments = lines.filter(isComment)

Print lines.count() , Comments.count()

Rdd = sc.parallelize([(1,2),(3,4),(3,5),(4,5),(4,4)])

Rdd.reduceByKey(lambda x,y:x+y).collect() 🡪 adds the number and gives the output from the same key . Here 3 is the same key in the above set of numbers

Output: (3,9),(1,2),(4,9)

rdd1 = sc.parallelize([(1,'c'),(3,'a'),(3,'d'),(2,'e'),(2,'b')])

rdd1.sortByKey().collect() 🡪 sorts by key value , sets the values depending on the value of numbers

output: [(1, 'c'), (2, 'e'), (2, 'b'), (3, 'a'), (3, 'd')]

rdd2 = sc.parallelize([(1,’a’),(2,’c’),(1,’a’)])

rdd2.groupByKey().collect() 🡪 groups by same key that is same value , so all values with key value 1 will be grouped together

output: [(1,['a','b']),(2,['c'])]

rdd2.reduceByKey(lambda x,y : x+y).collect()

output : [(3, 'c'), (1, 'aa'), (4, 'a'), (2, 'bc')]

BroadCast Variable

broadcastVar = sc.broadcast([1,2,3])

broadcastVar.value

output: [1,2,3]

🡪accum = sc.accumulator(0)

🡪rdd = sc.parallelize([1,2,3,4])

🡪def f(x):

🡪global accum

🡪accum += x

🡪rdd.foreach(f)

🡪accum.value

output: 10 🡪 adds all the vaue in rdd by running a foreach loop .

file = sc.textFile('/FileStore/tables/f1dg6wrp1463216222747/textfile1.txt')

file.collect()

blankLines = sc.accumulator(0)

def extractCallSigns(line):

global blankLines

if (line.strip() == ''):

blankLines += 1

return line.split(' ')

callsings = file.flatMap(extractCallSigns)

print "blankLines : %d" % blankLines.value

Spark lines read

Lines = sc.textFile(‘hdfs://sandbox.hortonworks.com/textfile’) 🡪 the texfile.txt is in directory textfile

Mylines = lines.filter(lambda x : len(x) > 5) 🡪 This will print lines whose length is greater than 5

Print Mylines.count()

Print Mylines.take(2) 🡪 takes first two elements or lines from the textfile