Experiment 7

Aim : Implement RDD using PySpark

BDT Experiment 7 DATE:

Kreena Shah
60004210243
C'32

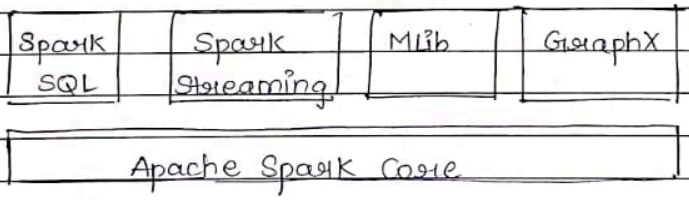
Aim : Implement RDD using PySpark

Theory :

Apache Spark is an open source, distributed processing system used for big data workloads.

It utilizes in-memory caching & optimized query execution for fast analytic queries against data of any size.

Sparks helps to run an application in Hadoop cluster upto 100 times faster in memory & 10 times faster when running on disk. This is possible by reducing the no. of read/write operations to disk



RDD { Resilient Distributed Dataset }

RDD in Apache Spark is a immutable collection of objects which computes on the different node of the cluster

It is the fundamental datastructure of Apache Spark.

Features :

(1) Resilience
(2) Distributed
(3) Lazy evaluation
(4) Immutability
(5) In-memory Computation
(6) Partitioning

Example 1 :
RDD of Squares of numbers [1,2,3,4,5] that are above 10

```
from pyspark import SparkContext
sc = SparkContext('local','RDD Example')
numbers = [1,2,3,4,5]
rdd = sc.parallelize(numbers)
squared_rdd = rdd.map(lambda x : x*x )
filtered_rdd = squared_rdd.filter(lambda x : x > 10)
result = filtered_rdd.collect()
print(result)
```

```
[16, 25]
```

Example 2 :
Rdd of Square Roots between 1 and 20

```
from pyspark import SparkContext, SparkConf
import math
conf = SparkConf().setAppName("SquareRootNumbers").setMaster("local")
sc = SparkContext.getOrCreate(conf=conf)
numbers_rdd = sc.parallelize(range(1, 21))
square_root_rdd = numbers_rdd.map(lambda x: math.sqrt(x))
square_roots = square_root_rdd.collect()
for square_root in square_roots:
print(square_root)
sc.stop()
```

```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
3.3166247903554
3.4641016151377544
3.605551275463989
3.7416573867739413
3.872983346207417
4.0
4.123105625617661
4.242640687119285
4.358898943540674
4.47213595499958
```

Example 3 :
Rdd of Armstrong number between 100 and 9999

```
conf = SparkConf().setAppName("ArmstrongNumbers").setMaster("local")
sc = SparkContext.getOrCreate(conf=conf)
def is_armstrong_number(num):
l = len(str(num))
sum_of_powers = 0
n = num
while n > 0:
digit = n % 10
sum_of_powers += digit ** l
n //= 10
return num == sum_of_powers
armstrong_numbers_rdd = sc.parallelize(range(100, 10000))
armstrong_numbers =
armstrong_numbers_rdd.filter(is_armstrong_number).collect()
print(armstrong_numbers)
sc.stop()
```

Output :

```
[153, 370, 371, 407, 1634, 8208, 9474]
```

Example 4:
Rdd of perfect number between 1 and 100

```
conf = SparkConf().setAppName("PerfectNumbers").setMaster("local")
sc = SparkContext.getOrCreate(conf=conf)
def is_perfect_number(n):
sum_of_divisors = 0
for i in range(1, n):
if n % i == 0:
sum_of_divisors += i
return sum_of_divisors == n
perfect_numbers_rdd = sc.parallelize(range(1, 101))
perfect_numbers = perfect_numbers_rdd.filter(is_perfect_number).collect()
print(perfect_numbers)
sc.stop()
```

Output :

```
[6, 28]
```

Conclusion : Thus, we understood spark & RDD, ensuring proper implementation of RDD using pySpark.