internal

December 3, 2017

0.0.1 Finding Prime Numbers

```
In [1]: n = 500000
        allnumbers = sc.parallelize(xrange(2, n), 8).cache()
        composite = allnumbers.flatMap(lambda x: xrange(x*2, n, x)) #.repartition(8)
        prime = allnumbers.subtract(composite)
        print prime.take(10)
```

```
[17, 401537, 462641, 97, 47137, 113, 43649, 467009, 193, 488833]
```

```
In [2]: # Find the number of elements in each parttion
        def partitionsize(it):
            s = 0
            for i in it:
                s += 1
            yield s

        #print allnumbers.mapPartitions(partitionsize).collect()
        print composite.mapPartitions(partitionsize).collect()
        #print prime.mapPartitions(partitionsize).collect()
        #print prime.glom().collect()[1][0:4]
```

```
[5216986, 254759, 104166, 62499, 0, 0, 0, 0]
```

0.0.2 Data Partitioning

```
In [6]: data = [8, 96, 240, 400, 401, 800]
        rdd = sc.parallelize(zip(data, data),4)
        print rdd.partitioner
        print rdd.glom().collect()  #return a list of all partitions
        rdd = rdd.reduceByKey(lambda x,y: x+y)
        print rdd.glom().collect()
        print rdd.partitioner.partitionFunc  # show partitioning is hash partitioning
        rdd = rdd.sortByKey# sort inside partitions
        print rdd.glom().collect()
        print rdd.partitioner.partitionFunc # show partitioning is range partitioning
```

```
None
[[(8, 8)], [(96, 96), (240, 240)], [(400, 400)], [(401, 401), (800, 800)]]
[[(8, 8), (96, 96), (400, 400), (240, 240), (800, 800)], [(401, 401)], [], []]
<function portable_hash at 0x7fd12809df50>
[[(8, 8), (96, 96)], [(240, 240), (400, 400)], [(401, 401)], [(800, 800)]]
<function rangePartitioner at 0x7fd12710e140>
```

```
In [9]: a = sc.parallelize(zip(range(10000), range(10000)), 8)
        b = sc.parallelize(zip(range(10000), range(10000)), 10)
        print a.partitioner
        a = a.reduceByKey(lambda x,y: x+y)
        print a.partitioner.partitionFunc
        b = b.reduceByKey(lambda x,y: x+y)
        c = a.join(b)
        print c.getNumPartitions()
        print c.partitioner.partitionFunc
        print c.glom().first()[0:4]
```

```
None
<function portable_hash at 0x7fd12809df50>
18
<function portable_hash at 0x7fd12809df50>
[(0, (0, 0)), (2052, (2052, 2052)), (4104, (4104, 4104)), (6156, (6156, 6156))]
```

```
In [1]: # A 'real' example from SF Express
        # Prepare three relational tables

        from pyspark.sql.functions import *

        num_waybills = 100000000
        num_customers = 10000000

        rdd = sc.parallelize((i, ) for i in xrange(num_waybills))
        waybills = spark.createDataFrame(rdd).select(floor(rand()*num_waybills).alias('waybill
                                                     floor(rand()*num_customers).alias('custome
                        .groupBy('waybill').max('customer').withColumnRenamed('max(customer)',
                        .cache()
        waybills.show()
        print waybills.count()

        rdd = sc.parallelize((i, i) for i in xrange(num_customers))
        customers = spark.createDataFrame(rdd, ['customer', 'phone']).cache()
        customers.show()
        print customers.count()

        rdd = sc.parallelize((i, ) for i in xrange(num_waybills))
```

```
        waybill_status = spark.createDataFrame(rdd).select(floor(rand()*num_waybills).alias('wa
                                                           floor(rand()*10).alias('version')) \
                              .groupBy('waybill').max('version').cache()
        waybill_status.show()
        print waybill_status.count()
```

```
+--------+--------+
| waybill|customer|
+--------+--------+
|19598711| 2936210|
|27925456| 4714650|
|85803846| 9371571|
|18161982| 8512119|
|68880032| 3555704|
|99571604| 7462577|
|36117674| 9505447|
|90934078| 1242445|
|68061416| 9930894|
|40538990| 9460875|
|55026752| 4493488|
|76897685| 2843346|
|22700507|  260965|
|32475525| 5656885|
|41766619| 4803479|
|47136640| 8131660|
|85889167| 4620277|
|99170331| 9619089|
|25540635| 7959351|
|15121704| 7977025|
+--------+--------+
only showing top 20 rows

63209204
+--------+-----+
|customer|phone|
+--------+-----+
|       0|    0|
|       1|    1|
|       2|    2|
|       3|    3|
|       4|    4|
|       5|    5|
|       6|    6|
|       7|    7|
|       8|    8|
|       9|    9|
|      10|   10|
|      11|   11|
```

```
|      12|   12|
|      13|   13|
|      14|   14|
|      15|   15|
|      16|   16|
|      17|   17|
|      18|   18|
|      19|   19|
+--------+-----+
only showing top 20 rows

10000000
+--------+------------+
| waybill|max(version)|
+--------+------------+
|49920425|           5|
|47672894|           9|
|13436597|           1|
|55699455|           3|
|71060754|           9|
|57668239|           9|
|24474721|           5|
| 3747982|           9|
|15754571|           4|
| 1059596|           4|
| 8402604|           9|
|35730516|           6|
|57990249|           8|
|71428271|           2|
|36630274|           6|
|74538215|           9|
|44464385|           6|
|73120156|           9|
| 8754850|           4|
|85379722|           3|
+--------+------------+
only showing top 20 rows

63211676
```

In [12]:
```
# We want to join 3 tables together.
# Knowing how each table is partitioned helps optimize the join order.

# waybills.join(customers, 'customer').join(waybill_status, 'waybill').show()
waybills.join(waybill_status, 'waybill').join(customers, 'customer').show()
```

```
+--------+--------+------------+-----+
|customer| waybill|max(version)|phone|
```

```
+--------+--------+-----------+-----+
|      29|14165698|          4|   29|
|      29|60096983|          4|   29|
|      29|49797517|          3|   29|
|     474|84204623|          4|  474|
|     964|23818017|          9|  964|
|     964|43141845|          0|  964|
|    1677| 2290942|          9| 1677|
|    1677|65944458|          2| 1677|
|    1677|47044996|          0| 1677|
|    1697|18844362|          3| 1697|
|    1697|73575919|          1| 1697|
|    1697|23137131|          7| 1697|
|    1806|35814782|          4| 1806|
|    1950|75470717|          9| 1950|
|    1950|83194159|          9| 1950|
|    1950|47200019|          5| 1950|
|    1950|43998641|          8| 1950|
|    1950|32856212|          6| 1950|
|    2214|29679632|          6| 2214|
|    2214|32706152|          8| 2214|
+--------+--------+-----------+-----+
only showing top 20 rows
```

```
In [14]: def partitionsize(it): yield len(list(it))

         n = 40000000

         def f(x):
             return x / (n/8)

         data1 = range(0, n, 16) + range(0, n, 16) #0,16,32,64,80,0,16,32,64,80
         data2 = range(0, n, 8)                                    #0,8,16,24,32,40,48,56,64,72
         rdd1 = sc.parallelize(zip(data1, data2), 8)
         rdd1 = rdd1.partitionBy(8, f)
         rdd2 = rdd1.reduceByKey(lambda x,y: x+y, partitionFunc=f)
         # rdd2 = rdd1.reduceByKey(lambda x,y: x+y)
         rdd2.mapPartitions(partitionsize).collect()
```

```
Out[14]: [312500, 312500, 312500, 312500, 312500, 312500, 312500, 312500]
```

0.0.3 Partitioning in DataFrames

```
In [1]: data1 = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4]
        data2 = [2, 2, 3, 4, 5, 3, 1, 1, 2, 3]
        df = spark.createDataFrame(zip(data1, data2))
```

```
        print df.rdd.getNumPartitions()
        print df.rdd.glom().collect()
```

```
8
[[Row(_1=1, _2=2)], [Row(_1=1, _2=2)], [Row(_1=1, _2=3)], [Row(_1=2, _2=4), Row(_1=2, _2=5)],
```

```
In [2]: df1 = df.repartition(6, df._2)
        print df1.rdd.glom().collect()
        df1.show()
```

```
[[], [], [Row(_1=1, _2=2), Row(_1=1, _2=2), Row(_1=2, _2=4), Row(_1=2, _2=5), Row(_1=3, _2=2)]
+---+---+
| _1| _2|
+---+---+
|  1|  2|
|  1|  2|
|  2|  4|
|  2|  5|
|  3|  2|
|  1|  3|
|  2|  3|
|  4|  3|
|  3|  1|
|  3|  1|
+---+---+
```

0.0.4 Threading

```
In [16]: import threading
         import random

         partitions = 40
         n = 5000000 * partitions

         # use different seeds in different threads and different partitions
         # a bit ugly, since mapPartitionsWithIndex takes a function with only index
         # and it as parameters
         def f1(index, it):
             random.seed(index + 987231)
             for i in it:
                 x = random.random() * 2 - 1
                 y = random.random() * 2 - 1
                 yield 1 if x ** 2 + y ** 2 < 1 else 0

         def f2(index, it):
             random.seed(index + 987232)
```

```
    for i in it:
        x = random.random() * 2 - 1
        y = random.random() * 2 - 1
        yield 1 if x ** 2 + y ** 2 < 1 else 0

def f3(index, it):
    random.seed(index + 987233)
    for i in it:
        x = random.random() * 2 - 1
        y = random.random() * 2 - 1
        yield 1 if x ** 2 + y ** 2 < 1 else 0

def f4(index, it):
    random.seed(index + 987234)
    for i in it:
        x = random.random() * 2 - 1
        y = random.random() * 2 - 1
        yield 1 if x ** 2 + y ** 2 < 1 else 0

def f5(index, it):
    random.seed(index + 987245)
    for i in it:
        x = random.random() * 2 - 1
        y = random.random() * 2 - 1
        yield 1 if x ** 2 + y ** 2 < 1 else 0

f = [f1, f2, f3, f4, f5]

# the function executed in each thread/job
def dojob(i):
    count = sc.parallelize(xrange(1, n + 1), partitions) \
              .mapPartitionsWithIndex(f[i]).reduce(lambda a,b: a+b)
    print "Worker", i, "reports: Pi is roughly", 4.0 * count / n

# create and execute the threads
threads = []
for i in range(5):
    t = threading.Thread(target=dojob, args=(i,))
    threads += [t]
    t.start()

# wait for all threads to complete
for t in threads:
    t.join()
'''
for i in range(5):
    dojob(i)
'''
```

```
Worker 1 reports: Pi is roughly 3.1415736
Worker 0 reports: Pi is roughly 3.14156124
Worker 3 reports: Pi is roughly 3.14157024
Worker 2 reports: Pi is roughly 3.14157354
Worker 4 reports: Pi is roughly 3.14154802
```

Out[16]:

```
for i in range(5):
    dojob(i)
```