9_summary_RDD_transformations_actions

February 21, 2022

1 Important operations of RDD - Transformations and Actions

[1]:
```
# Import SparkContext and SparkConf
from pyspark import SparkContext, SparkConf

# Initialize spark
conf = SparkConf().setAppName("IntroRDDSummary")
sc = SparkContext(conf=conf)
```

```
22/02/21 14:36:36 WARN Utils: Your hostname, ThinkCentre resolves to a loopback
address: 127.0.1.1; using 10.180.5.223 instead (on interface eno1)
22/02/21 14:36:36 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
22/02/21 14:36:36 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
22/02/21 14:36:38 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
```

[2]:
```
sc.master
```

[2]: 'local[*]'

2 1-1. Basic RDD "transform" -> will not execute immediately

2.1 parallelize

[3]:
```
# Create a RDD using parallelize - creates 2 partitions - 2 tasks
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
```

[4]:
```
# Collect the partitions to driver - Action
intRDD.collect()
```

[4]: [1, 2, 3, 5, 5]

[5]:
```python
# What if we dont mention the number of partitions?
# Takes default number of cores on your machine
# But it is machine dependent...

stringRDD = sc.parallelize(["A", "B", "A"])
stringRDD.glom().collect()
```

[5]: [[], ['A'], ['B'], ['A']]

2.2 map & reduce

[6]:
```python
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
intRDD.map(lambda x: x + 1).collect()
```

[6]: [2, 3, 4, 6, 6]

[7]:
```python
def addOne(x):
    return x + 1

intRDD.map(addOne).collect()
```

[7]: [2, 3, 4, 6, 6]

[8]:
```python
def merge(x, y):
    return x + y

intRDD.reduce(merge)
```

[8]: 16

2.3 flatMap

[9]:
```python
sc.parallelize([[1, 2], [1, 3, 4], [4, 5]]).flatMap(lambda x: x).collect()
```

[9]: [1, 2, 1, 3, 4, 4, 5]

[10]:
```python
# Replicate...using *
sc.parallelize([1, 2, 3]).map(lambda x: [x]*x).collect()
```

[10]: [[1], [2, 2], [3, 3, 3]]

[11]:
```python
[2] * 3
```

[11]: [2, 2, 2]

[12]:
```
sc.parallelize([1, 2, 3]).flatMap(lambda x: [x]*x).collect()
```

[12]: [1, 2, 2, 3, 3, 3]

2.4 WordCount

[13]:
```
lines = sc.parallelize(["I love you", "do you love me"])
```

[14]:
```
lines.flatMap(lambda l: l.split()).collect()
```

[14]: ['I', 'love', 'you', 'do', 'you', 'love', 'me']

[15]:
```
# Split...assign 1 to each word..collect
lines.flatMap(lambda l: l.split()).map(lambda w: (w, 1)).collect()
```

[15]:
```
[('I', 1),
 ('love', 1),
 ('you', 1),
 ('do', 1),
 ('you', 1),
 ('love', 1),
 ('me', 1)]
```

[16]:
```
# Count the frequency using reduceByKey
lines.flatMap(lambda l: l.split()).map(lambda w: (w, 1)) \
    .reduceByKey(lambda x, y: x + y).collect()
```

[16]: [('do', 1), ('you', 2), ('love', 2), ('I', 1), ('me', 1)]

[17]:
```
# All together..full solution..Sort by frequency..
lines.flatMap(lambda l: l.split()) \
    .map(lambda w: (w, 1)) \
    .reduceByKey(lambda x, y: x + y) \
    .sortBy(lambda t: -t[1]).collect()
```

[17]: [('you', 2), ('love', 2), ('do', 1), ('I', 1), ('me', 1)]

2.5 filter

[18]:
```
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
intRDD.filter(lambda x: x > 3).collect()
```

[18]: [5, 5]

2.6 distinct

[19]:
```
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
intRDD.distinct().collect()
```

[19]: [2, 1, 3, 5]

2.7 randomSplit

[20]:
```
dataRDD = sc.parallelize(range(10))
sampleRDDs = dataRDD.randomSplit([0.8, 0.2])

print(sampleRDDs[0].collect()) # First Random List
print(sampleRDDs[1].collect()) # SEcond Random List
dataRDD.collect()
```

```
[0, 1, 3, 4, 6, 8, 9]
[2, 5, 7]
```

[20]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[21]:
```
type(sampleRDDs)
```

[21]: list

[22]:
```
len(sampleRDDs[0].collect())
```

[22]: 7

3 1-2. Multiple RDD "transform"

[23]:
```
intRDD1 = sc.parallelize([1, 2, 3, 5, 5])
intRDD2 = sc.parallelize([5, 6])
intRDD3 = sc.parallelize([2, 7])
```

3.1 union

[24]:
```
# RDD1 U RDD2
intRDD1.union(intRDD2).collect()
```

[24]: [1, 2, 3, 5, 5, 5, 6]

[25]:
```
# Union of RDD1, RDD2 and RDD3 - RDD1 U RDD2 U RDD3
# Duplicates will not be deleted
intRDD1.union(intRDD2).union(intRDD3).collect()
```

[25]: [1, 2, 3, 5, 5, 5, 6, 2, 7]

3.2 intersection

[26]:
```
# No duplicates..
intRDD1 = sc.parallelize([1, 2, 3, 5, 5])
intRDD2 = sc.parallelize([5, 5,  6])

intRDD1.intersection(intRDD2).collect()
```

[26]: [5]

3.3 substract

[27]:
```
# Exists in RDD1 and does not exist in RDD2
intRDD1 = sc.parallelize([1, 2, 3, 5, 5])
intRDD2 = sc.parallelize([5, 5,  6])
intRDD1.subtract(intRDD2).collect()
```

[27]: [1, 2, 3]

3.4 cartesian

[28]:
```
# Return all x,y pairs, including duplicates
intRDD1 = sc.parallelize([1, 2, 3, 5, 5])
intRDD2 = sc.parallelize([5, 5,  6])

intRDD1.cartesian(intRDD2).collect()
```

[28]:
```
[(1, 5),
 (1, 5),
 (1, 6),
 (2, 5),
 (2, 5),
 (2, 6),
 (3, 5),
 (3, 5),
 (3, 6),
 (5, 5),
 (5, 5),
 (5, 5),
 (5, 5),
 (5, 6),
 (5, 6)]
```

4 1-3. Basic RDD Actions

4.1 first, take

[29]:
```
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
intRDD.first()
```

[29]: 1

[30]:
```
intRDD = sc.parallelize([1, 2, 3, 5, 5], 2)
intRDD.take(2)
```

[30]: [1, 2]

4.2 takeOrdered - Get it sorted..

[31]:
```
testRDD = sc.parallelize([4, 3, 1, 2])
```

[32]:
```
testRDD.takeOrdered(3)
```

[32]: [1, 2, 3]

[33]:
```
testRDD.takeOrdered(3, key=lambda x: -x) # Descending Order - Reverse Indexing
```

[33]: [4, 3, 2]

5 2-1. Basic Key-Value RDD "transform"

[34]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
```

5.1 keys

[35]:
```
kvRDD1.keys().collect() # Allows duplicate keys
```

[35]: [3, 3, 5, 1]

5.2 values

[36]:
```
kvRDD1.values().collect()
```

[36]: [4, 6, 6, 2]

5.3 map, filter

[37]:
```
# Get all keys using map and lambda
kvRDD1.map(lambda x: x[0]).collect()
```

[37]:
```
[3, 3, 5, 1]
```

[38]:
```
# Get the points where key < 5
# x - Point
# x[0] - Key
# x[1] - Value
kvRDD1.filter(lambda x: x[0] < 5).collect()
```

[38]:
```
[(3, 4), (3, 6), (1, 2)]
```

5.4 mapValues

[39]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.mapValues(lambda v: v * v).collect()
```

[39]:
```
[(3, 16), (3, 36), (5, 36), (1, 4)]
```

[40]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.mapValues(lambda v: v * v).collect()
```

[40]:
```
[(3, 16), (3, 36), (5, 36), (1, 4)]
```

5.5 sortByKey

[41]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.sortByKey().collect()
```

[41]:
```
[(1, 2), (3, 4), (3, 6), (5, 6)]
```

[42]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.sortByKey(ascending=False).collect()
```

[42]:
```
[(5, 6), (3, 4), (3, 6), (1, 2)]
```

5.6 reduceByKey and groupByKey

[43]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.reduceByKey(lambda x, y: x + y).collect()
```

[43]:
```
[(5, 6), (1, 2), (3, 10)]
```

[44]:
```python
gvs = kvRDD1.groupByKey().collect()
print(gvs)
```

```
[(5, <pyspark.resultiterable.ResultIterable object at 0x7fe6f6a616a0>), (1,
<pyspark.resultiterable.ResultIterable object at 0x7fe6f6a61d30>), (3,
<pyspark.resultiterable.ResultIterable object at 0x7fe6f6a61880>)]
```

[45]:
```python
# v is also an interable object
for k, v in gvs:
    print(k, v)
```

```
5 <pyspark.resultiterable.ResultIterable object at 0x7fe6f6a616a0>
1 <pyspark.resultiterable.ResultIterable object at 0x7fe6f6a61d30>
3 <pyspark.resultiterable.ResultIterable object at 0x7fe6f6a61880>
```

[46]:
```python
# [(3, 4), (3, 6), (5, 6), (1, 2)]
for k, vs in gvs:
    #print(k)
    for v in vs:
        print(k ,v,  sep="-", end=" ")
```

```
5-6 1-2 3-4 3-6
```

6 2-2. Multiple Key-Value RDD "transform"

[47]:
```python
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD1.collect()
```

[47]:
```
[(3, 4), (3, 6), (5, 6), (1, 2)]
```

[48]:
```python
kvRDD2 = sc.parallelize([(3, 8), (4, 7)])
kvRDD2.collect()
```

[48]:
```
[(3, 8), (4, 7)]
```

6.1 join (by key)

[49]:
```python
kvRDD1.join(kvRDD2).collect()
```

[49]:
```
[(3, (4, 8)), (3, (6, 8))]
```

6.2 leftOuterJoin

[50]:
```python
# Keep all the keys and values of leftside RDD of Join
# and if the key doesnt exist in RDD2 then keep it as None
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
```

```
kvRDD2 = sc.parallelize([(3, 8), (4, 7)])
kvRDD1.leftOuterJoin(kvRDD2).collect()
```

[50]: [(1, (2, None)), (3, (4, 8)), (3, (6, 8)), (5, (6, None))]

6.3 rightOuterJoin

[51]:
```
kvRDD1.rightOuterJoin(kvRDD2).collect()
```

[51]: [(3, (4, 8)), (3, (6, 8)), (4, (None, 7))]

6.4 fullOuterJoin

[52]:
```
kvRDD1.fullOuterJoin(kvRDD2).collect()
```

[52]: [(1, (2, None)), (3, (4, 8)), (3, (6, 8)), (4, (None, 7)), (5, (6, None))]

6.5 substruactByKey

[53]:
```
kvRDD1 = sc.parallelize([(3, 4), (3, 6), (5, 6), (1, 2)])
kvRDD2 = sc.parallelize([(3, 8), (4, 7)])
kvRDD1.subtractByKey(kvRDD2).collect()
```

[53]: [(1, 2), (5, 6)]

7 2-3. Key-Value RDD "action"

[54]:
```
kvRDD1.first()
```

[54]: (3, 4)

[55]:
```
kvRDD1.take(2)
```

[55]: [(3, 4), (3, 6)]

7.1 countByKey

[56]:
```
kvRDD1.collect()
```

[56]: [(3, 4), (3, 6), (5, 6), (1, 2)]

[57]:
```
cnt_dict = kvRDD1.countByKey()
print(cnt_dict)
```

```
defaultdict(<class 'int'>, {3: 2, 5: 1, 1: 1})
```

[58]:
```
sc.parallelize([1, 1, 2, 2, 2, 2, 3, 3]).countByValue()
```

[58]:
```
defaultdict(int, {1: 2, 2: 4, 3: 2})
```

7.2 collectAsMap (make sure key is distinct)

[59]:
```
kvRDD1.collect()
```

[59]:
```
[(3, 4), (3, 6), (5, 6), (1, 2)]
```

[60]:
```
kvRDD1.collectAsMap() # convert it as dict, no duplicate keys
```

[60]:
```
{3: 6, 5: 6, 1: 2}
```

7.3 lookup

[61]:
```
# Serach for a key and print the corresponding values
kvRDD1.lookup(3)
```

[61]:
```
[4, 6]
```

8 3. Broadcast

What are Broadcast Variables?

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

Broadcast variables in Apache Spark is a mechanism for sharing variables across executors that are meant to be read-only. Without broadcast variables these variables would be shipped to each executor for every transformation and action, and this can cause network overhead. However, with broadcast variables, they are shipped once to all executors and are cached for future reference.

[62]:
```
peopleMap = ['Mike', 'Mary', 'Tiffany', 'Jenny'] # {0: 'Mike', 1}
```

[63]:
```
peopleIds = sc.parallelize([1, 3, 0, 2])
```

[64]:
```
peopleIds.collect()
```

[64]:
```
[1, 3, 0, 2]
```

8.1 Before broadcast

[65]:
```
peopleIds.map(lambda x: peopleMap[x]).collect()
```

[65]:
```
['Mary', 'Jenny', 'Mike', 'Tiffany']
```

8.2 After broadcast

[66]:
```
bpeopleMap = sc.broadcast(peopleMap)
```

[67]:
```
bpeopleMap
```

[67]:
```
<pyspark.broadcast.Broadcast at 0x7fe6f6a4d7c0>
```

[68]:
```
bpeopleMap.value
```

[68]:
```
['Mike', 'Mary', 'Tiffany', 'Jenny']
```

8.3 Only one worker node saves share peopleMap in memory

9 4. Accumulator

!!! Data cannot be arbitrarily changed during the parallelization process

Accumulator variables are used for aggregating the information through associative and commutative operations. For example, you can use an accumulator for a sum operation or counters (in MapReduce). The following code block has the details of an Accumulator class for PySpark.

[69]:
```
intRDD = sc.parallelize(range(10))
```

[70]:
```
total = sc.accumulator(0.0)
```

[71]:
```
num = sc.accumulator(0)
```

[72]:
```
intRDD.foreach(lambda x: [total.add(x), num.add(1)])
```

[73]:
```
total.value, num.value, (total.value / num.value)
```

[73]:
```
(45.0, 10, 4.5)
```

Another example...

[74]:
```
num = sc.accumulator(1)

def accum(x):    # accum is a user defined function
    global num
    num+=x

rdd = sc.parallelize([2,3,4,5])
rdd.foreach(accum) # For each value of RDD call accum
sum5 = num.value
print("Accumulated value is -> %i" % (sum5))
```

```
Accumulated value is -> 15
```

10 5. persist

[75]:
```
myRDD = sc.parallelize(range(1000))
#myRDD.collect()
```

10.1 Before doing something different, cache myRDD

[76]:
```
myRDD.persist()
```

[76]: PythonRDD[176] at RDD at PythonRDD.scala:53

[77]:
```
myRDD.count()
```

[77]: 1000

[78]:
```
myRDD.take(5)
```

[78]: [0, 1, 2, 3, 4]

[79]:
```
# Filter all the numbers divisible by 100
myRDD.filter(lambda x: x%100 == 0).collect()
```

[79]: [0, 100, 200, 300, 400, 500, 600, 700, 800, 900]

10.2 unpersist

[80]:
```
myRDD.unpersist()
```

[80]: PythonRDD[176] at RDD at PythonRDD.scala:53

11 6. Text IO

11.1 textFile

[81]:
```
lines = sc.textFile("./news.txt")
```

[82]:
```
lines.take(1)
```

[82]: ['Google, which not along ago was using artificial intelligence to identify cat pictures, has moved onto something bigger -- breast cancer.']

[83]:
```
lines.count()
```

[83]: 12

[84]:
```python
# Count the number of lines having the word google
new_text = lines.filter(lambda line: 'Google' in line)
new_text.collect()
```

[84]:
```
['Google, which not along ago was using artificial intelligence to identify cat
pictures, has moved onto something bigger -- breast cancer.',
 'Google announced Friday that it has achieved state-of-the-art results in using
artificial intelligence to identify breast cancer. The findings are a reminder
of the rapid advances in artificial intelligence, and its potential to improve
global health.',
 "Google used a flavor of artificial intelligence called deep learning to
analyze thousands of slides of cancer cells provided by a Dutch university. Deep
learning is where computers are taught to recognize patterns in huge data sets.
It's very useful for visual tasks, such as looking at a breast cancer biopsy.",
 "With 230,000 new cases of breast cancer every year in the United States,
Google (GOOGL, Tech30) hopes its technology will help pathologists better treat
patients. The technology isn't designed to, or capable of, replacing human
doctors.",
 '"What we\'ve trained is just a little sliver of software that helps with one
part of a very complex series of tasks," said Lily Peng, the project manager
behind Google\'s work. "There will hopefully be more and more of these tools
that help doctors [who] have to go through an enormous amount of information all
the time."',
 "Related: Google's artificial intelligence can actually help the environment",
 'Peng described to CNNTech how the human and the computer could work together
to create better outcomes. Google\'s artificial intelligence system excels at
being very sensitive to potential cancer. It will flag things a human will miss.
But it sometimes will falsely identify something as cancer, whereas a human
pathologist is better at saying, "no, this isn\'t cancer."',
 "For now, Google's progress is still research mode and remains in the lab.
Google isn't going to become your pathologist's assistant tomorrow. But Google
and many other players are striving toward a future where that becomes a
reality.",
 "Jeroen van der Laak, an associate professor in digital pathology at Radboud
University Medical Center, believes the first algorithms for cancer will be
available within a couple years, and large-scale routine use will occur in about
five years. His university provided the slides for Google's research."]
```

[85]:
```python
new_text.count()
```

[85]: 9

11.2 saveAsTextFile

Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.

[86]:
```
# If output dir does not exist then it creates...
# Saves as multiple partitions
# Wont support overwriting...
new_text.saveAsTextFile("./outputs/news1")
```

[87]:
```
# How to save it as a single file?
# Merge partitions and then save..
new_text.coalesce(1).saveAsTextFile("./outputs/news2")
```