CMU MSP 36602: Spark, Part 2

H. Seltman, March 25, 2019

I) IntroToSpark.pdf covered the general overview, with an emphasis on pyspark. There we covered methods that do not use a key. Next, we cover those that work with keys. Methods include aggregateByKey, combineByKey, countByKey, foldByKey, groupByKey, reduceByKey, sampleByKey, sortByKey, and subtractByKey.

II) Load cust.dat and purchase.dat

```
c = sc.textFile("cust.dat").filter(lambda s: s[0:2] != 'id')
c = c.map(lambda s: s.split()).map(lambda v: [int(x) for x in v])

p = sc.textFile("purchase.dat").filter(lambda s: s[0:2] != 'id')
p = p.map(lambda s: s.split()).map(
  lambda v: [int(v[0]), int(v[1]), float(v[2]), int(v[3])])
```

III) Create key value pairs using `myRDD.map(lambda e: (key(e), value(e))`

```
pkv = a.map(lambda t: (t[0], t[1:]))
pkv.take(3)
# [(623910455, [12, 25.53, 1]), (967941424, [7, 58.11, 1]),
#  (517476391, [15, 62.31, 0])]
```

IV) Sort by key: myRDD.sortByKey(ascending=True, numPartitions=None, keyfunc=)

```
pkv.sortByKey(False).take(3)
# [(998263021, [8, 113.13, 1]), (998263021, [12, 20.07, 1]),
#  (998263021, [8, 79.43, 1])]
```

```
p.map(lambda x: (x[2], x[0])).sortByKey(
   keyfunc=lambda k: abs(k-15.0)).take(5)
# [(15.03, 425127415), (14.95, 694736375), (14.84, 13865483),
#  (14.8, 530671275), (15.24, 98728700)]
```

V) Aggregate by key: The goal is to produce one (key, value) pair per key, aggregating across multiple entries with the same key. The hidden concern is that there are multiple partitions, and the aggregation is first within partitions, then across partitions. The most clear and general method is: `myRDD.aggregateByKey(zeroValue, seqFunc, combFunc, numPartition=None)`

We can call the result type R and the RDD element type E. We have 'zeroValue' of type R that represents the result if the key is missing in a partition as well as the first argument of the 'seqFunc' that merges (aka combines, aggregates) a previous result with an RDD element. In other words the form for 'seqFunc' is R = f(R, E). First f(zeroValue, E_1) is run and stored as, say "R". Then f(R, E_2) is run and stored, etc. The function is not allowed to change the "E".

To combine across partitions, we use 'combFunc' which is of the form R = f(R, R). Sometimes the argument of type R is called an "accumulator". Because the combine function is run an unknown number of times (one per partition) the 'combFunc' operation must be associative and commutative to assure a well-defined result. Again, the second argument may not be altered.

Here is an example that takes data of the form (id, (store, amount, debit)) and returns (count, debitCount, totalAmount, debitAmount) for each id. Need: f(a, b)=f(b,a) & f(f(a, b),c)=f(a, f(b,c)).

```
def mySeqFun(acc, val):
  rslt = (acc[0] + 1, acc[1] + val[2],
          acc[2] + val[1], acc[3] + val[1]*val[2])
  return rslt

def myCombFun (acc1, acc2):
    rslt = (acc1[0] + acc2[0], acc1[1] + acc2[1],
            acc1[2] + acc2[2], acc1[3] + acc2[3])
    return rslt

# Pick a customer to focus on
myId = p.map(lambda x: x[0]).take(12)[-1]
myData = pkv.filter(lambda x: x[0] == myId).collect()
myData
(myData[0][1][1]+myData[1][1][1],
 myData[0][1][1]+myData[1][1][1]+myData[2][1][1])
# (145.18, 162.42000000000002)

# Test: (works for any value of repartition()
pkv.repartition(2).aggregateByKey((0,0,0,0), mySeqFun,
  myCombFun).filter(lambda x: x[0]==myId).collect()
# [(389155401, (3, 2, 162.42000000000002, 145.18))]

# Sum of squared "debit" values :
# f(f(a,b),c) = (a²+b²)²+c² ≠ a²+(b²+c²)² = f(a, f(b,c))
pkv.repartition(2).aggregateByKey(0, lambda a, v: a**2 + v[2]**2,
    lambda a1, a2: a1**2 + a2**2).filter(
       lambda x: x[0]==myId).collect()
# [(389155401, 4)]
pkv.repartition(4).aggregateByKey(0, lambda a, v: a**2 + v[2]**2,
    lambda a1, a2: a1**2 + a2**2).filter(
       lambda x: x[0]==myId).collect()
# [(389155401, 2)]
```

VI) Fold by key: `rdd.foldByKey(zeroValue, func)`

Example: from (id, (1, amount, debit)) create (n, total amount, count debit)

```
pkv.map(lambda kv: (kv[0], (1, kv[1][1], kv[1][2]))). \
    repartition(7).foldByKey((0, 0, 0),
                             lambda a, v: (a[0]+v[0], a[1]+v[1],
                                           a[2]+v[2])). \
    filter(lambda x: x[0] in ids[:5]).sortByKey().collect()
# [(279814400, (7, 509.96, 3)), (425127415, (9, 509.27, 4)),
#  (517476391, (6, 442.70000000000005, 4)),
#  (623910455, (6, 437.67999999999995, 5)),
#  (967941424, (6, 377.82, 5))]
```

VII) Inner join: rdd1.join(rdd2) does an inner join by key value

Use join() to combine the pdv info (above) with the customer info (above)

VIII) ***Reduce by key:*** `rdd.reduceByKey(func)`

Compared to foldByKey, this reportedly can run into problems with empty elements.

```
pkv.map(lambda kv: (kv[0], (1, kv[1][1], kv[1][2]))). \
    repartition(7).reduceByKey( \
                            lambda a, v: (a[0]+v[0], a[1]+v[1],
                                          a[2]+v[2])). \
    filter(lambda x: x[0] in ids[:5]).sortByKey().collect()
# [(279814400, (7, 509.96, 3)), (425127415, (9, 509.27, 4)),
#  (517476391, (6, 442.70000000000005, 4)),
#  (623910455, (6, 437.67999999999995, 5)),
#  (967941424, (6, 377.82, 5))]
```

IX) ***Group by key followed by map().*** rdd.groupByKey() produces key/value pairs where the value is an iterable containing all values for each key. This will be less efficient because the full data rather than reduced data will need to be swapped between partitions.

X) **Breakout**: Hadoop log file analysis

Goal: Process and examine the Hadoop log file "/usr/local/hadoop/logs/hadoop-student-namenode-Student-VM.log".

Input: space separated text file with fields for date (YYYY-MM-DD), time, type, program, and message (one to ~30 words)

i) Load the data, restricting to records starting with the year 20xx.

ii) Reformulate to key / value pairs where the key is the "type" and the value is (year, month, day, time (HH:SS only), program, message (a string holding the first 5 words)).

iii) Show the distribution of message for type ERROR

iv) Show the distribution of message for type WARNING

v) Show the distribution of the first 10 characters of the message for type MESSAGE

vi) Find N and fraction of INFO messages starting with "BLOCK" per year-month

XI) ***Next:*** Machine learning, SQL, and graphs in Spark