Spark Core API

https://github.com/spark-in-action/first-edition

Zadatak

- Potrebno je da se generiše izvještaj o broju operacija PUSH za svakog korisnika

- Dataset

  wget https://data.gharchive.org/2015-01-01-{0..23}.json.gz

Dataset


▼ object {7}
id : 2489368070
type : PushEvent
▼ actor {5}
id : 9152315
login : davidjhulse
gravatar_id : value
url : https://api.github.com/users/davidjhulse
avatar_url : https://avatars.githubusercontent.com/u/9152315?
▶ repo {3}
▼ payload {7}
push_id : 536740396
size : 1
distinct_size : 1
ref : refs/heads/master
head : a9b22a6d80c1e0bb49c1cf75a3c075b642c28f81
before : 86ffa724b4d70fce46e760f8cc080f5ec3d7d85f
▶ commits [1]
public : true
created_at : 2015-01-01T00:00:00Z

Dataframe API

- Dataframe je RDD sa šemom, slično tabeli iz relacionog modela
- Kreiranje dataframe iz JSON fajla – automatsko izvođenje šeme
- Sintaksa je SQL-like i omogućava deklarativni način zadavanja upita
  - Razlika u odnosu na Spark Core API

Dataframe API 2

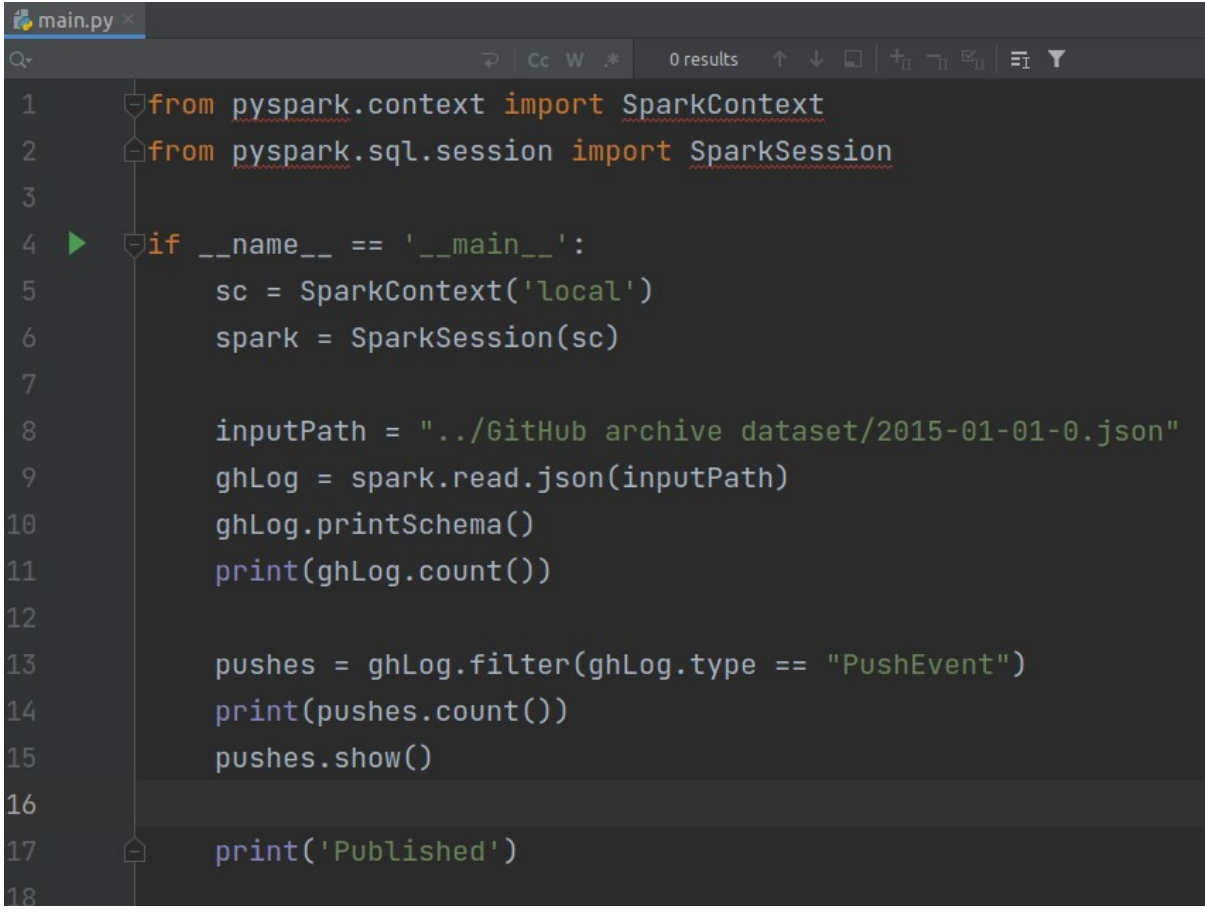
- http://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html#dataframe-apis
  - pyspark.sql.SparkSession.read
    - Moguće je pročitati više fajlova u jedan dataframe
  - pyspark.sql.DataFrame.printSchema
  - pyspark.sql.DataFrame.count
  - pyspark.sql.DataFrame.show
  - pyspark.sql.DataFrame.filter

Primjer rješenja

- pyspark < main.py

  ili

- spark-submit main.



Agregiranje podataka

- pyspark.sql.DataFrame.groupBy
- pyspark.sql.DataFrame.orderBy

```
grouped = ghLog.groupBy(ghLog.actor.login).count()
grouped.show(5)

ordered = grouped.orderBy(grouped["count"].desc())
ordered.show(5)
```

Zadatak, nastavak

- Potrebno je eliminisati PUSH operacije korisnika koji nijesu na spisku zaposlenih
-
- Lista korisničkih imena zaposlenih učitava se iz tekstualne datoteke

UDFs, broadcast

```
from pyspark.sql.types import BooleanType
from pyspark.sql import functions as F

@F.udf(returnType=BooleanType())
def isemp(user):
    return user in bcEmployees.value
```

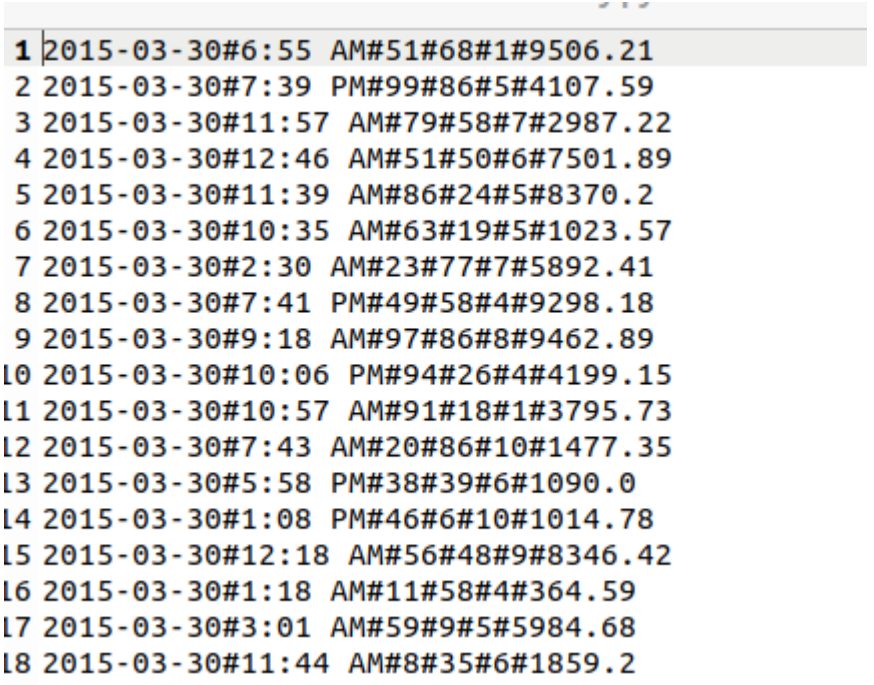
```
31      empPath = "./ghEmployees.txt"
32      employees = set(line.strip() for line in open(empPath))
33
34      bcEmployees = sc.broadcast(employees)
35      filtered = ordered.filter(isemp("actor[login]"))
36      filtered.show()
37
```

Zadatak 2, Spark Core API

- Napisati program koji na osnovnu obavljenih transakcija i određenog skupa pravila klijentima dodjeljuje nagradne proizvode
  - Klijentu koji je napravio najviše transakcija kao poklon dodjeljuje se proizvod ID=4
  - Klijentima koji su kupili >=2 proizvoda ID=25 obračunati 5% popusta
  - Klijentima koji su kupili >=5 proizvoda ID=81 pokloniti proizvod ID=70
  - Klijentu koji je potrošio najviše novca poklon je proizvod ID=63
  - Nagradni proizvodi upisuju se kao nove transakcije čija je vrijednost 0
  - Prikazati listu transakcija nagrađenih klijenata

Dataset

- Transaction date, time, customer ID, product ID, quantity, price



```
1 2015-03-30#6:55 AM#51#68#1#9506.21
2 2015-03-30#7:39 PM#99#86#5#4107.59
3 2015-03-30#11:57 AM#79#58#7#2987.22
4 2015-03-30#12:46 AM#51#50#6#7501.89
5 2015-03-30#11:39 AM#86#24#5#8370.2
6 2015-03-30#10:35 AM#63#19#5#1023.57
7 2015-03-30#2:30 AM#23#77#7#5892.41
8 2015-03-30#7:41 PM#49#58#4#9298.18
9 2015-03-30#9:18 AM#97#86#8#9462.89
10 2015-03-30#10:06 PM#94#26#4#4199.15
11 2015-03-30#10:57 AM#91#18#1#3795.73
12 2015-03-30#7:43 AM#20#86#10#1477.35
13 2015-03-30#5:58 PM#38#39#6#1090.0
14 2015-03-30#1:08 PM#46#6#10#1014.78
15 2015-03-30#12:18 AM#56#48#9#8346.42
16 2015-03-30#1:18 AM#11#58#4#364.59
17 2015-03-30#3:01 AM#59#9#5#5984.68
18 2015-03-30#11:44 AM#8#35#6#1859.2
```

Pair RDD

- RDD sa (key, value) elementima
-

```
if __name__ == '__main__':
    sc = SparkContext('local')
    spark = SparkSession(sc)

    tranFile = sc.textFile("./ch04_data_transactions.txt")
    transData = tranFile.map(lambda transa: transa.split("#"))
    transByCust = transData.map(lambda transa: (int(transa[2]), transa))
    print(transByCust.keys().distinct().count())
    print(transByCust.collect())
```

countByKeys

```
custCounts = transByCust.countByKey()
print(custCounts)

(cid, transa) = sorted(transByCust.countByKey().items(), key=operator.itemgetter(1))[-1]
print(cid, transa)
```

```
defaultdict(<class 'int'>, {51: 18, 99: 12, 79: 13, 86: 9, 63: 12, 23: 13, 49:
, 97: 12, 94: 12, 91: 13, 20: 8, 38: 9, 46: 9, 56: 17, 11: 8, 59: 9, 8: 10, 85:
9, 27: 7, 84: 9, 54: 7, 74: 11, 6: 7, 35: 10, 39: 11, 47: 13, 17: 13, 40: 10, 5
: 8, 80: 7, 87: 10, 52: 9, 30: 5, 62: 6, 41: 12, 71: 10, 61: 8, 95: 8, 5: 11, 2
 15, 78: 11, 13: 12, 4: 11, 100: 12, 19: 6, 98: 11, 53: 19, 89: 9, 15: 10, 45:
1, 12: 7, 32: 14, 16: 8, 1: 9, 72: 7, 14: 8, 7: 10, 28: 11, 43: 12, 93: 12, 70:
8, 73: 7, 65: 10, 50: 14, 3: 13, 69: 7, 60: 4, 76: 15, 66: 11, 90: 8, 10: 7, 34
 14, 83: 12, 64: 10, 18: 9, 81: 9, 44: 8, 21: 13, 88: 5, 58: 13, 24: 9, 26: 11,
77: 11, 36: 5, 22: 10, 31: 14, 96: 8, 29: 9, 33: 9, 68: 12, 75: 10, 25: 12, 48:
5, 82: 13, 9: 7, 67: 5, 37: 7, 55: 13, 92: 8, 42: 7})
```

lookup

```
print(transByCust.lookup(53))
complTrans = [["2015-03-30", "11:59 PM", "53", "4", "1", "0.00"]]
```

```
[['2015-03-30', '6:18 AM', '53', '42', '5', '2197.85'], ['2015-03-30', '4:42
, '53', '44', '6', '9182.08'], ['2015-03-30', '2:51 AM', '53', '59', '5', '31
43'], ['2015-03-30', '5:57 PM', '53', '31', '5', '6649.27'], ['2015-03-30', '
1 AM', '53', '33', '10', '2353.72'], ['2015-03-30', '9:46 PM', '53', '93', '1
'2889.03'], ['2015-03-30', '4:15 PM', '53', '72', '7', '9157.55'], ['2015-03-
, '2:42 PM', '53', '94', '1', '921.65'], ['2015-03-30', '8:30 AM', '53', '38'
5', '4000.92'], ['2015-03-30', '6:06 AM', '53', '12', '6', '2174.02'], ['2015
 30', '3:44 AM', '53', '47', '1', '7556.32'], ['2015-03-30', '10:25 AM', '53'
```

mapValues

```
def applyDiscount(tran):
    if(int(tran[3])==25 and float(tran[4])>1):
        tran[5] = str(float(tran[5])*0.95)
    return tran
```

transByCust = transByCust.mapValues(lambda t: applyDiscount(t))

flatMapValues

```
def addProductID70(tran):
    if(int(tran[3]) == 81 and int(tran[4])>4):
        from copy import copy
        cloned = copy(tran)
        cloned[5] = "0.00"
        cloned[3] = "70"
        cloned[4] = "1"
        return [tran, cloned]
    else:
        return [tran]
```

```
transByCust = transByCust.flatMapValues(lambda transa: addProductID70(transa))
print(transByCust.count())
#1006
```

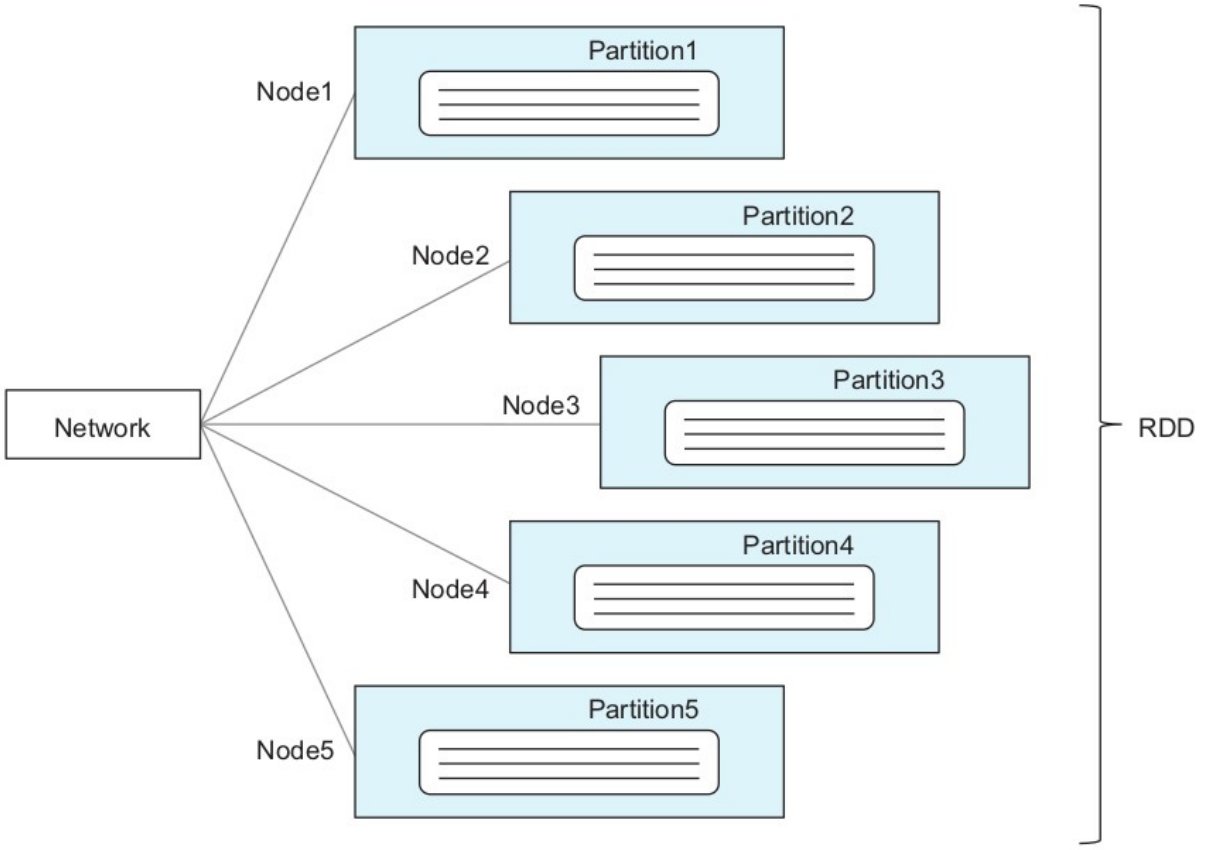
reduceByKey

```
amounts = transByCust.mapValues(lambda transa: float(transa[5]))
#amounts je pair RDD sa elementima (custID, amount)
print(amounts.collect())
totals = amounts.reduceByKey(lambda p1, p2: p1 + p2)
print(totals.collect())
(cid, max_total) = sorted(totals.collect(), key=operator.itemgetter(1))[-1]
print(cid, max_total)
complTrans += [["2015-03-30", "11:59 PM", "76", "63", "1", "0.00"]]
```

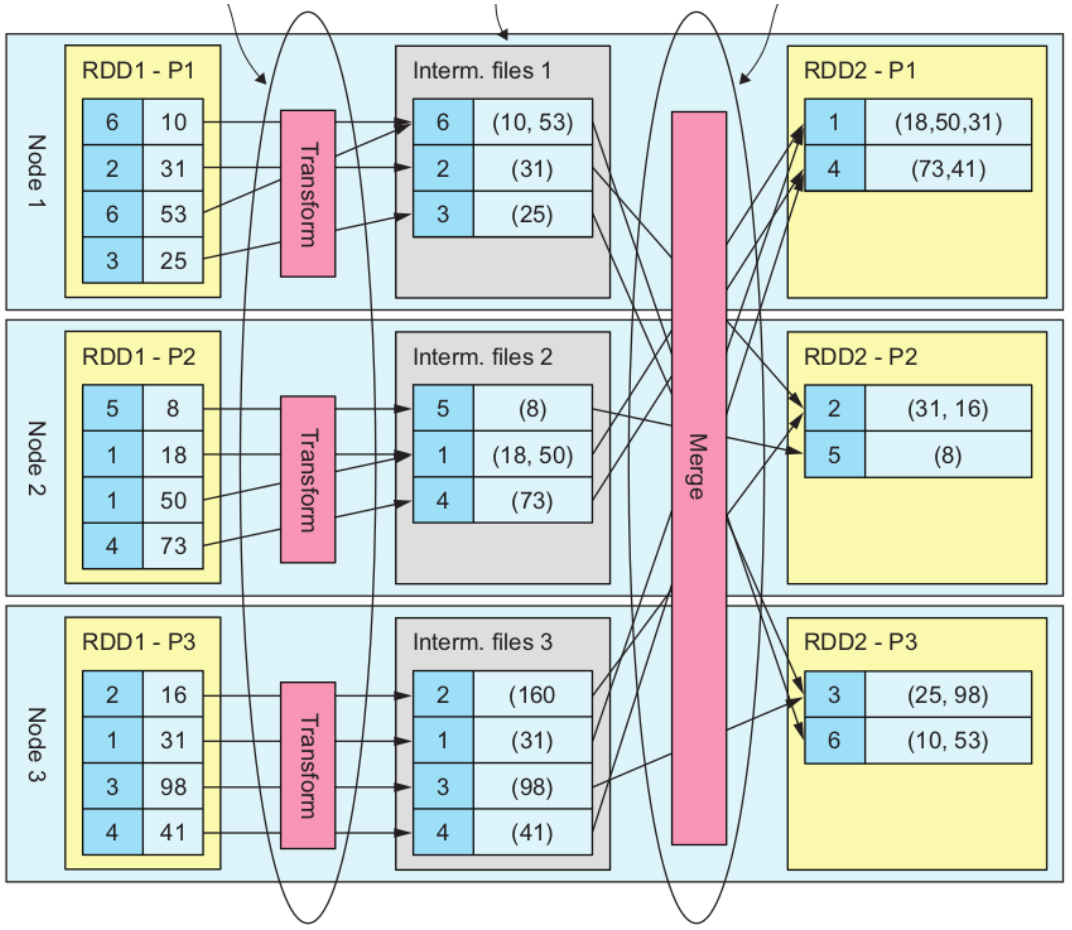
parallelize, union

```
complTransRDD = sc.parallelize(complTrans)
complTransData = complTransRDD.map(lambda transa: (int(transa[2]), transa))
transByCust = transByCust.union(complTransData)
print(transByCust.count())
#1008
```

Data partitioning



Data shuffling



Primjer

```
>>>
>>> a = sc.parallelize([1, 2, 3, 4, 5, 6])
>>> a.getNumPartitions()
4
>>> a.glom().collect()
[[1], [2, 3], [4], [5, 6]]
>>> b = a.repartition(2)
>>> b.glom().collect()
[[1, 4, 5, 6], [2, 3]]
>>>
```

Data shuffling 2

- Operacije koje uzrokuju shuffling
  - Pair RDD transformacije: aggregateByKey, foldByKey, reduceByKey, groupByKey, join, leftOuterJoin, rightOuterJoin, fullOuterJoin, subtractByKey
  - RDD transformacije: subtract, intersection, groupWith
  - sortByKey
  - partitionBy ili coalesce sa parametrom shuffle=true

Zadatak 2, nastavak

- Prikazati nazive proizvoda i ukupnu vrijednost prodaje za svaki od njih
-
- Spisak proizvoda je u tekstualnom fajlu

```
10#LEGO The Hobbit#7314.55#9
11#LEGO Minecraft#5646.81#3
12#LEGO Hero Factory#6911.2#1
13#LEGO Architecture#604.58#5
14#LEGO Technic#7423.48#3
15#LEGO Storage & Accessories#3125.96#2
16#LEGO Classic#9933.3#10
17#LEGO Galaxy Squad#5593.16#4
18#LEGO Mindstorms#6022.88#10
19#LEGO Minifigures#5775.99#1
20#LEGO Elves#4589.79#4
```

Joining data

```
transByProd = transData.map(lambda transa: (int(transa[3]), transa))
transByProdDouble = transByProd.mapValues(lambda transa: float(transa[5]))
totalsByProdTotal = transByProdDouble.reduceByKey(lambda p1, p2: p1 + p2)
print(totalsByProdTotal.collect()[0])
#(68, 62133.90000000001)
products = sc.textFile("./ch04_data_products.txt")
productsByIds = products.map(lambda line: line.split("#"))
productsByIntIds = productsByIds.map(lambda line: (int(line[0]), line))
print(productsByIntIds.collect()[67])
#(68, ['68', 'Niacin', '6295.48', '1'])
```

Joining Data 2

- Dva pair RDD objekta sa elementima (K, V) i (K, W)
  - join – kreira se pair RDD sa (K, (V, W)), kao inner join u DBMS
  - leftOuterJoin
  - rightOuterJoin
  - fullOutherJoin

Joining Data 3

```
totalsAndProducts = totalsByProdTotal.join(productsByIntIds)
print(totalsAndProducts.first())
#(68, (62133.90000000001, ['68', 'Niacin', '6295.48', '1']))
```

Zbog kompletnosti

- substract - works on pair RDDs and returns an RDD with pairs from the first RDD whose keys aren’t in the second RDD
- intersection - accepts an RDD of the same type as the enclosing one and returns a new RDD that contains elements present in both RDDs
- cartesian - makes a cartesian product of two RDDs in the form of an RDD of tuples (T, U) containing all possible pairs of elements from the first RDD (containing elements of type T) and second RDD (containing elements of type U)

Grupisanje

```
>>> a = sc.parallelize([('A', 1), ('A', 2), ('B', 1), ('B', 3), ('C', 1)])
>>> a.groupByKey()
PythonRDD[33] at RDD at PythonRDD.scala:53
>>> b = a.groupByKey()
>>> b.collect()
[('A', <pyspark.resultiterable.ResultIterable object at 0x7f1c86cd7970>), ('B',
<pyspark.resultiterable.ResultIterable object at 0x7f1c85e49fa0>), ('C', <pyspar
k.resultiterable.ResultIterable object at 0x7f1c85e53040>)]
>>> def createComb(t):
...     return int(t)
...
>>> def mergeVal(a, b):
...     return a+b
...
>>> def mergeComb(a, b):
...     return a+b
...
>>> a.combineByKey(createComb, mergeVal, mergeComb)
PythonRDD[43] at RDD at PythonRDD.scala:53
>>> b = a.combineByKey(createComb, mergeVal, mergeComb)
>>> b.collect()
[('A', 3), ('B', 4), ('C', 1)]
>>>
```

RDD zavisnosti