Introduction to Spark
Lecture 2

236826, Winter 2015
Key/Value Pairs

• Pair RDDs
• Can reduceByKey(), join() and other operations.
• Creation:

```python
>>> pairs1 = sc.parallelize([(1,2), (3,4), (3,6)])
>>> pairs1.collect()
[(1, 2), (3, 4), (3, 6)]
```
Pair RDDs - Transformations

```python
>>> a = israelLines.map(lambda x: (x.split(" ")[0], x)).collect()
```

```python
>>> a[0]
```

(u'Africans', u'Africans profess a wide variety of religious beliefs, with [[Christianity]] and [[Islam]] being the most widespread. Approximately 40% of all Africans are Christians and another 40% Muslims. Roughly 20% of Africans primarily follow indigenous [[African religions]]. A small number of Africans also have beliefs [[African Jew|from the Judaic tradition]], such as the [[Beta Israel]] and [[Lemba]] tribes.')
Pair RDDs - Transformations

```python
c >>> pairs1 = sc.parallelize([(1,2), (3,4), (3,6)])
c >>> a = pairs1.reduceByKey(lambda x,y: x+y).collect()
c >>> a
[(1, 2), (3, 10)]
c >>> c = pairs1.mapValues(lambda x: x + 1).collect()
c >>> c
[(1, 3), (3, 5), (3, 7)]
c >>> d = pairs1.keys().collect()
c >>> d
[1, 3, 3]
c >>> pairs1.values().collect()
c [2, 4, 6]
c >>> pairs1.sortByKey().collect()
c [(1, 2), (3, 4), (3, 6)]
c >>> sc.parallelize([(7,2), (3,4), (1,6)]).sortByKey().collect()
c [(1, 6), (3, 4), (7, 2)]
```
Pair RDDs - Transformations

```python
>>> range(4)
[0, 1, 2, 3]
>>> e = pairs1.flatMapValues(lambda x: range(x)).collect()
>>> e
[(1, 0), (1, 1), (3, 0), (3, 1), (3, 2), (3, 3), (3, 0), (3, 1),
(3, 2), (3, 3), (3, 4), (3, 5)]
>>> pairs = israelLines.map(lambda x: (x.split(' ')[0], x))
>>> a = pairs.mapValues(lambda y: len(y))
>>> a.collect()[0]
(u'Africans', 403)
```
Printing a structure

Print a structure of the form
    [[1,[1,2]], [2,[2,4,7]]]

```python
>>> def printStruct(b):
...    for x in b:
...        print (x[0]),
...        print("": "",
...        z = 0
...        for y in x[1]:
...            if z==0:
...                z=1
...            else:
...                print("", "",
...                print(y),
...                print(" ")
```
Pair RDDs - Transformations

```python
>>> pairs1 = sc.parallelize([(1,2), (3,4), (3,6)])
>>> b = pairs1.groupByKey().collect()
>>> printStruct(b)
1 : 2
3 : 4, 6
```
Pairs RDD as “ordinary RDDs”

```python
>>> pairs1 = sc.parallelize([(1,2), (3,4), (3,6)])
>>> pairs1.filter(lambda kv: kv[1] > 3 ).collect()
[(3, 4), (3, 6)]
```
Operating on 2 Pair RDDs: **subtractByKey**

```python
>>> pairs1.collect()
[(1, 2), (3, 4), (3, 6)]
>>> other.collect()
[(3, 9)]
>>> pairs1.subtractByKey(other).collect()
[(1, 2)]
>>> other.subtractByKey(pairs1).collect()
[]
```
Operating on 2 Pair RDDs: `join()`

```python
>>> pairs1.collect()
[(1, 2), (3, 4), (3, 6)]
>>> other.collect()
[(3, 9)]
>>> pairs1.join(other).collect()
[(3, (4, 9)), (3, (6, 9))]
>>> other.join(pairs1).collect()
[(3, (9, 4)), (3, (9, 6))]
```
Operating on 2 Pair RDDs: `rightOuterJoin()`

```python
>>> pairs1.collect()
[(1, 2), (3, 4), (3, 6)]
```

```python
>>> other.collect()
[(3, 9)]
```

```python
>>> pairs1.rightOuterJoin(other).collect()
[(3, (4, 9)), (3, (6, 9))]
```

```python
>>> other.rightOuterJoin(pairs1).collect()
[(1, (None, 2)), (3, (9, 4)), (3, (9, 6))]
```

```python
>>> pairs1.leftOuterJoin(other).collect()
[(1, (2, None)), (3, (4, 9)), (3, (6, 9))]
```

```python
>>> g = pairs1.cogroup(other).collect()
```

```python
>>> printStruct1(g)

1 :  
       [  2  ], 
       [   ]

3 :  
       [  4 ,  6  ], 
       [  9  ]
```

In `rightOuterJoin` the key in the result must be present in the “other” (right) operand.
def printStruct1(b):
    for (k,(x,m)) in b:
        print (k),
        print(". "),
        z = 0
        print("[ "),
        for y in x:
            if z==0:
                z=1
            else:
                print("\ ",),
                print(y),
                print(y),
                print(" ]",),
        z = 0
        print("[ "),
        for y in m:
            if z==0:
                z=1
            else:
                print("\ ",),
                print(y),
                print(y),
                print(" ]"),
        print(" ")
Aggregation - 1

```python
>>> pairs = sc.parallelize([('panda', 0), ('pink', 3), ('pirate', 3), ('panda', 1), ('pink', 4)])
```

```python
>>> pairs.collect()
[('panda', 0), ('pink', 3), ('pirate', 3), ('panda', 1), ('pink', 4)]
```

```python
>>> pairs.mapValues(lambda x: (x,1)).reduceByKey(lambda x, y : (x[0] + y[0], x[1] + y[1])).collect()
[['pink', (7, 2)], ['panda', (1, 2)], ['pirate', (3, 1)]]
```

- From which we can compute average per key
>>> pairs.collect()
[('panda', 0), ('pink', 3), ('pirate', 3), ('panda', 1), ('pink', 4)]

>>> sumCount1 = pairs.combineByKey( (lambda x: (x,1)),
...                                 (lambda x, y: (x[0] + y, x[1] + 1)),
...                                 (lambda x, y: (x[0] + y[0], x[1] + y[1])))

>>> >>> sumCount1.collect()
[('pink', (7, 2)), ('panda', (1, 2)), ('pirate', (3, 1))]

>>> rr = sumCount1.map(lambda (key, (x, y)): (key, x/y) )

>>> rr.collect()
[('pink', 3), ('panda', 0), ('pirate', 3)]

>>> rr = sumCount1.map(lambda (key, (x, y)): (key, x/y) ).collectAsMap()

>>> for x in rr:
...  print(x)
...  print(rr[x])
...  print()
pink
3
panda
0
pirate
3

3 functions:
- New key in partition: create acc
- Existing key in partition – add to acc
- Merge per key, 2 partitions: add entry-wise
Level of Parallelism

• Setting number of partitions

```python
>>> data = ["a", 3), ("b", 4), ("a", 1)]

>>> t = sc.parallelize(data).reduceByKey(lambda x,y: x+y)
# default
PythonRDD[18] at RDD at PythonRDD.scala:43

>>> t.glom().collect()

[[], [('b', 4)], [('a', 4)], [], [], []]

>>> sc.parallelize(data).reduceByKey(lambda x,y: x+y, 10)
# custom
PythonRDD[24] at RDD at PythonRDD.scala:43

[[], [('b', 4)], [], [], [('a', 4)], [], [], [], []]

Use glom() to display partitions
Level of Parallelism

- **repartition():** shuffles data to a new partition, expensive. Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.

```python
>>> ra.glom().collect()
[[], [('b', 4)], [], [], [('a', 4)], [], [], [], []]
>>> ra = ra.repartition(3)
>>> ra.glom().collect()
[['a', 4], ['b', 4], []]
```

- **coallesce():** avoids data movement when reducing the number of partitions

```python
>>> ra.glom().collect()
[['a', 4], ['b', 4], []]
>>> ra2 = ra.coalesce(2)
>>> ra2.glom().collect()
[['a', 4], ['b', 4]]
```

See [https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html](https://spark.apache.org/docs/1.1.1/api/python/pyspark.rdd.RDD-class.html)
Grouping Data (1)

- **groupByKey()**: paired data of type (K,V) is transformed into [K, iterable[V]].
- **groupBy()**: unpaired data or different condition besides key equality.
  - Takes a function that is applied to each RDD element to generate a key.

```python
>>> rdd = sc.parallelize([1, 1, 2, 3, 5, 8])
>>> result = rdd.groupBy(lambda x: x % 2).collect()
>>> for (x,y) in result:
...    print (str(x) + ":"),
...    sorted(y)
...
0: 
[2, 8]
1: 
[1, 1, 3, 5]
```

```python
>>> sorted([(x, sorted(y)) for (x, y) in result])
[[0, [2, 8]], (1, [1, 1, 3, 5])]
```
Grouping Data (2)

- `cogroup()`: takes 2 RDDs with same key type `K` and value types `V`, `W`. Produces RDD of type `[K, iterable[V], iterable[W]]`, can have an empty group if key is present in only 1 group.
  - Used as a building block for joins
Remark

Is `rdd.reduceByKey(func)` equivalent to `Rdd.groupByKey().mapValues(lambda x: x.reduce(func))`?

```
>>> a = sc.parallelize([("a", 1), ("b", 2), ("a", 3)])
>>> b = a.reduceByKey(lambda x,y: x + y)
>>> b.collect()
[('b', 2), ('a', 4)]
>>> c = a.groupByKey().mapValues(lambda x: x.reduce(lambda x,y: x + y))
.... AttributeError: 'ResultIterable' object has no attribute 'reduce'
```

Joins

```python
>>> ra = [ ("Ritual", "a1"), ("Philz", "a2"), ("Philz", "a4"),
        ("Star", "a3") ]
>>> rr = [ ("Ritual", 4.9), ("Philz", 4.75) ]
>>> sAddress = sc.parallelize(ra)
>>> sRating = sc.parallelize(rr)
>>> sAddress.join(sRating).collect()
[('Philz', ('a2', 4.75)), ('Philz', ('a4', 4.75)), ('Ritual', ('a1',
  4.9000000000000004))]
>>> sAddress.leftOuterJoin(sRating).collect()
[('Philz', ('a2', 4.75)), ('Philz', ('a4', 4.75)), ('Ritual', ('a1',
  4.9000000000000004)), ('Star', ('a3', None))]
>>> sAddress.rightOuterJoin(sRating).collect()
[('Philz', ('a2', 4.75)), ('Philz', ('a4', 4.75)), ('Ritual', ('a1',
  4.9000000000000004))]
```
Sorting data

• Can sort a paired RDD when there’s an ordering defined on key.

```python
>>> sAddress.sortByKey(ascending=True, numPartitions=None, keyfunc=lambda x: str(x)).collect()
>>> sAddress.sortByKey(ascending=True, numPartitions=None, keyfunc=lambda x: str(x)[3]).collect()
[('Philz', 'a2'), ('Philz', 'a4'), ('Star', 'a3'), ('Ritual', 'a1')]
>>> sAddress.sortByKey(ascending=True, numPartitions=None, keyfunc=lambda x: x[::-1]).collect()
[('Ritual', 'a1'), ('Star', 'a3'), ('Philz', 'a2'), ('Philz', 'a4')]
This is extended slice syntax. It works by doing [begin:end:step] - by leaving begin and end off and specifying a step of -1, it reverses a string.
**Actions on Pair RDDs**

- `countByKey()`

```python
>>> sc.parallelize([(1,2), (3,4), (3,6)]).countByKey()
defaultdict(<type 'int'>, {1: 1, 3: 2})
```

```python
>>> sc.parallelize([(1,2), (3,4), (3,6)]).collectAsMap()
{1: 2, 3: 6}
```

```python
>>> sc.parallelize([(1,2), (1,900), (3,4), (3,6), (3,7), (3,2)]).lookup(3)
[4, 6, 7, 2]
```

*collectAsMap() produces a function, so it keeps the last pair in which a key is mentioned.*