

Анализ больших данных с Apache Spark

Лекция 3. Spark API

Мурашкин Вячеслав
2017

<https://github.com/a4tunado/lectures-hse-spark/tree/master/003>

Лекция 3. Spark API

- Концепция функционального программирования
- Функции-преобразования RDD (transformation)
- Функции-действия RDD (actions)
- Операция Shuffle (перемещение данных)
- Shared variables (общие переменные)
- DataFrame API и SparkSQL
- Отладка программы
- Примеры

Функциональное программирование

- Вычисления представляют собой последовательность вызовов **функций**
- **Не сохраняется** состояние между вызовами функций
- Результат выполнения функции зависит **только** от входных параметров функции
- Исходные данные доступны **только для чтения** (immutable), в результате выполнения функции создаются **новые** объекты

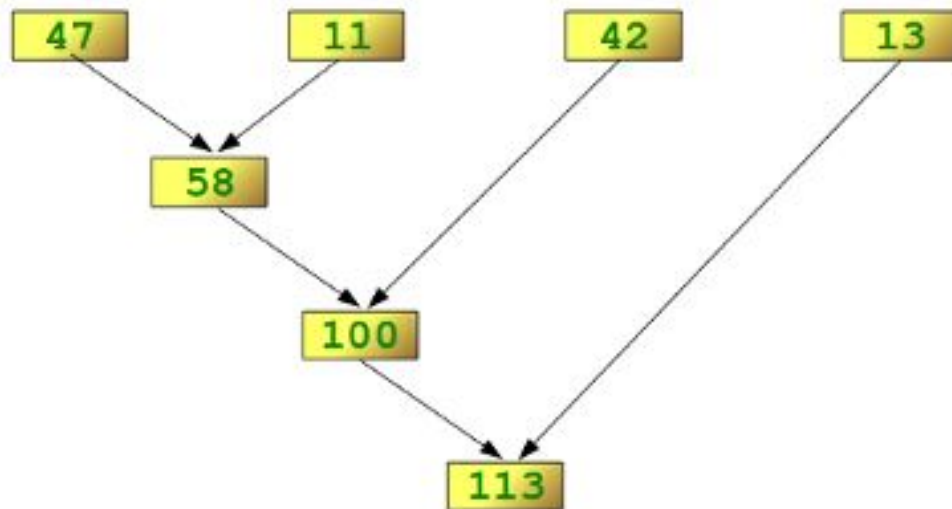
Функциональное программирование

```
a = 0
def increment():
    global a
    a += 1
```

```
def increment(a):
    return a + 1
```

Функциональное программирование

```
>>> reduce(lambda x,y: x+y, [47,11,42,13])  
113
```



Функции-преобразования (transformations) RDD

<i>map</i> ($f : T \Rightarrow U$)	:	$RDD[T] \Rightarrow RDD[U]$
<i>filter</i> ($f : T \Rightarrow \text{Bool}$)	:	$RDD[T] \Rightarrow RDD[T]$
<i>flatMap</i> ($f : T \Rightarrow \text{Seq}[U]$)	:	$RDD[T] \Rightarrow RDD[U]$
<i>sample</i> (<i>fraction</i> : Float)	:	$RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling)
<i>groupByKey</i> ()	:	$RDD[(K, V)] \Rightarrow RDD[(K, \text{Seq}[V])]$
<i>reduceByKey</i> ($f : (V, V) \Rightarrow V$)	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
<i>union</i> ()	:	$(RDD[T], RDD[T]) \Rightarrow RDD[T]$
<i>join</i> ()	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$
<i>cogroup</i> ()	:	$(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (\text{Seq}[V], \text{Seq}[W]))]$
<i>crossProduct</i> ()	:	$(RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$
<i>mapValues</i> ($f : V \Rightarrow W$)	:	$RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning)
<i>sort</i> ($c : \text{Comparator}[K]$)	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$
<i>partitionBy</i> ($p : \text{Partitioner}[K]$)	:	$RDD[(K, V)] \Rightarrow RDD[(K, V)]$

Функции-действия (actions) RDD

<i>count()</i>	:	$\text{RDD}[T] \Rightarrow \text{Long}$
<i>collect()</i>	:	$\text{RDD}[T] \Rightarrow \text{Seq}[T]$
<i>reduce</i> ($f : (T, T) \Rightarrow T$)	:	$\text{RDD}[T] \Rightarrow T$
<i>lookup</i> ($k : K$)	:	$\text{RDD}[(K, V)] \Rightarrow \text{Seq}[V]$ (On hash/range partitioned RDDs)
<i>save</i> ($path : \text{String}$)	:	Outputs RDD to a storage system, <i>e.g.</i> , HDFS

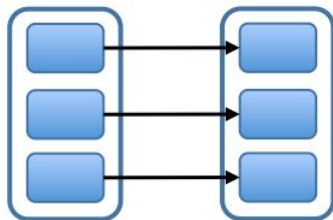
foreach(func)

Run a function *func* on each element of the dataset. This is usually done for side effects such as updating an [Accumulator](#) or interacting with external storage systems.

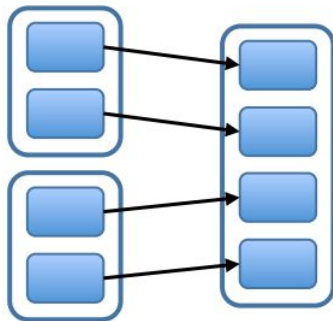
Note: modifying variables other than Accumulators outside of the `foreach()` may result in undefined behavior. See [Understanding closures](#) for more details.

Операция Shuffle

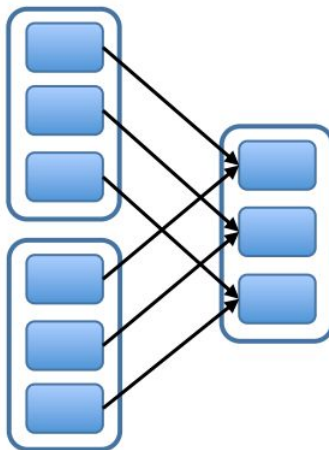
Narrow Dependencies:



map, filter

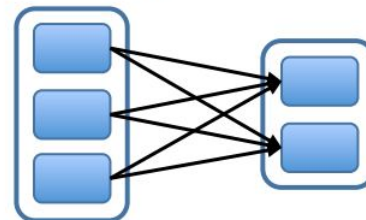


union

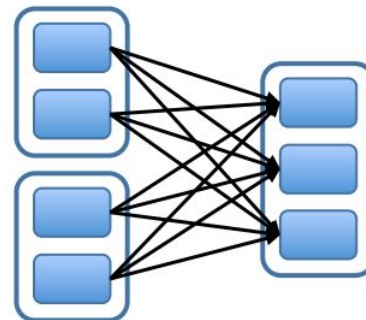


join with inputs
co-partitioned

Wide Dependencies:



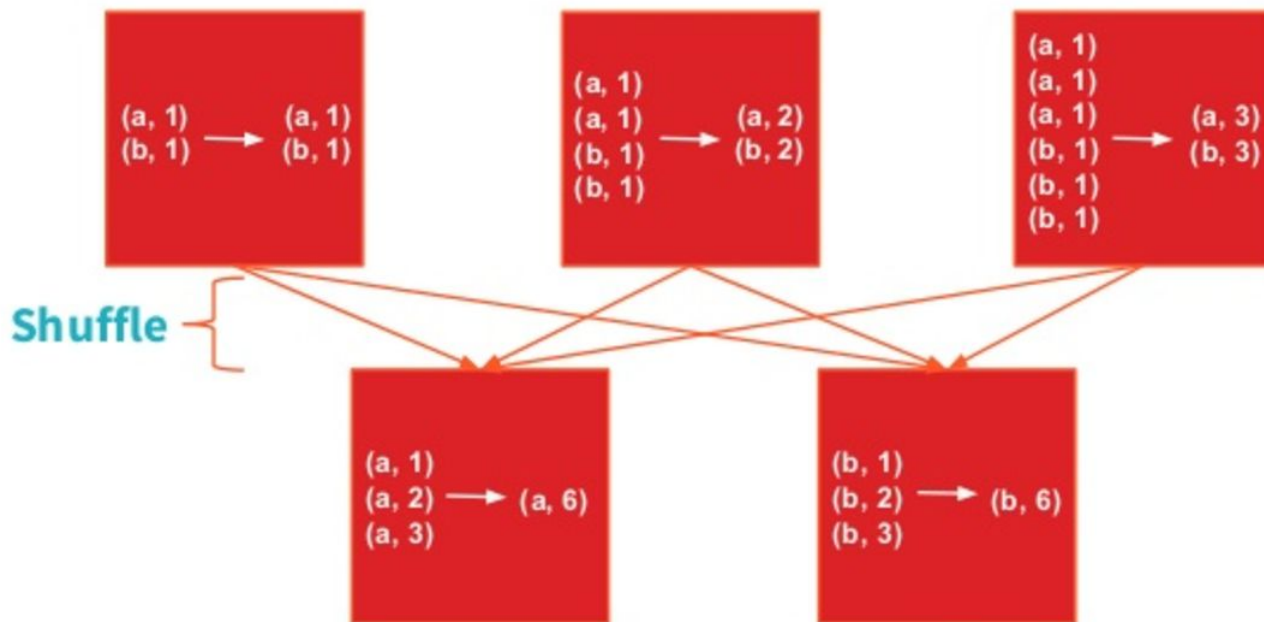
groupByKey



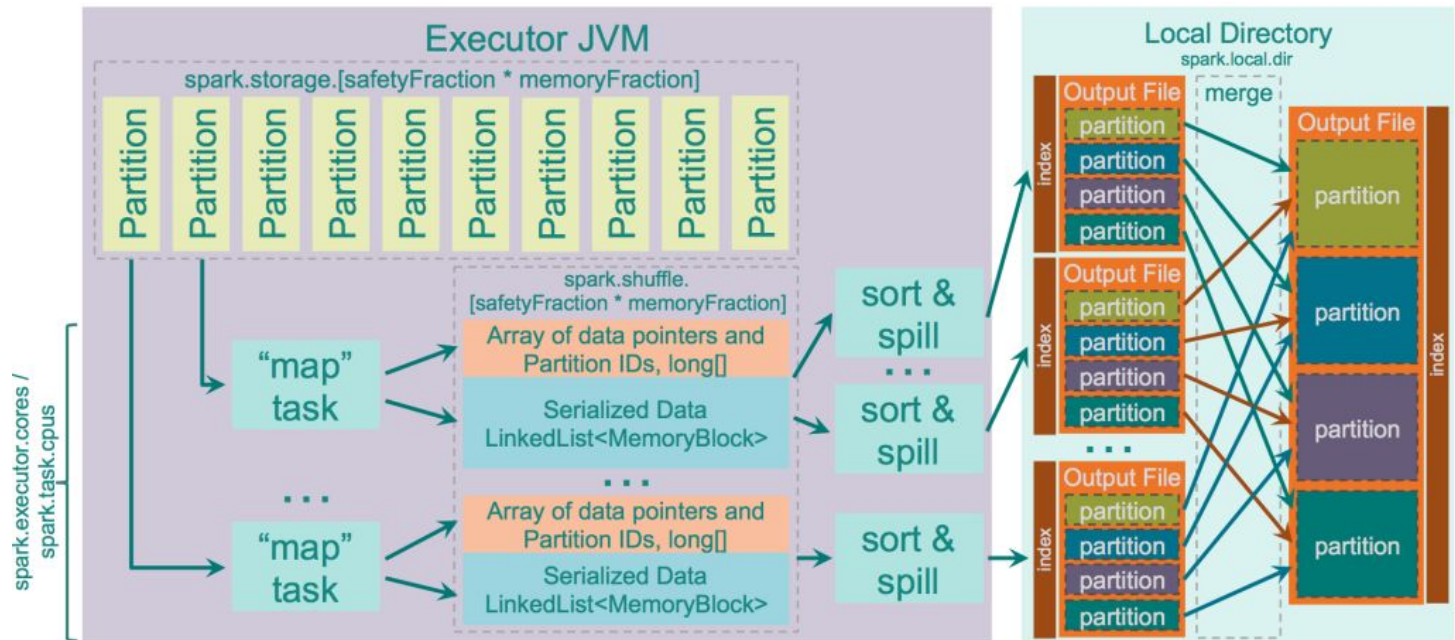
join with inputs not
co-partitioned

Операция Shuffle

ReduceByKey: Shuffle Step



Операция Shuffle



<https://issues.apache.org/jira/browse/SPARK-7081>

<https://0x0fff.com/spark-architecture-shuffle/>

Shared variables (общие переменные)

- В процессе обработки данных на нодах доступны переменные, инициализированные на клиенте (driver program)
- Изменение значений этих переменных на нодах не приводит к изменению значений на клиенте

Shared variables (общие переменные)

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
def increment_counter(x):
    global counter
    counter += x
rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Shared variables (общие переменные)

- Есть способа поддержки обновлений переменных на клиенте
 - **broadcast variables**
 - только чтение
 - имеет смысл если данные переменной используются на нескольких этапах обработки данных
 - полезно для пересылки большого объема данных
 - **accumulators**
 - на нодах чтение недоступно, можно только изменять значение
 - изменение доступно только в функциях-действиях (actions)
 - есть возможность определять пользовательские типы аккумуляторов

Shared variables (общие переменные)

```
>>> broadcastVar = sc.broadcast([1, 2, 3])  
<pyspark.broadcast.Broadcast object at 0x102789f10>  
  
>>> broadcastVar.value  
[1, 2, 3]
```

Shared variables (общие переменные)

```
>>> accum = sc.accumulator(0)
>>> accum
Accumulator<id=0, value=0>

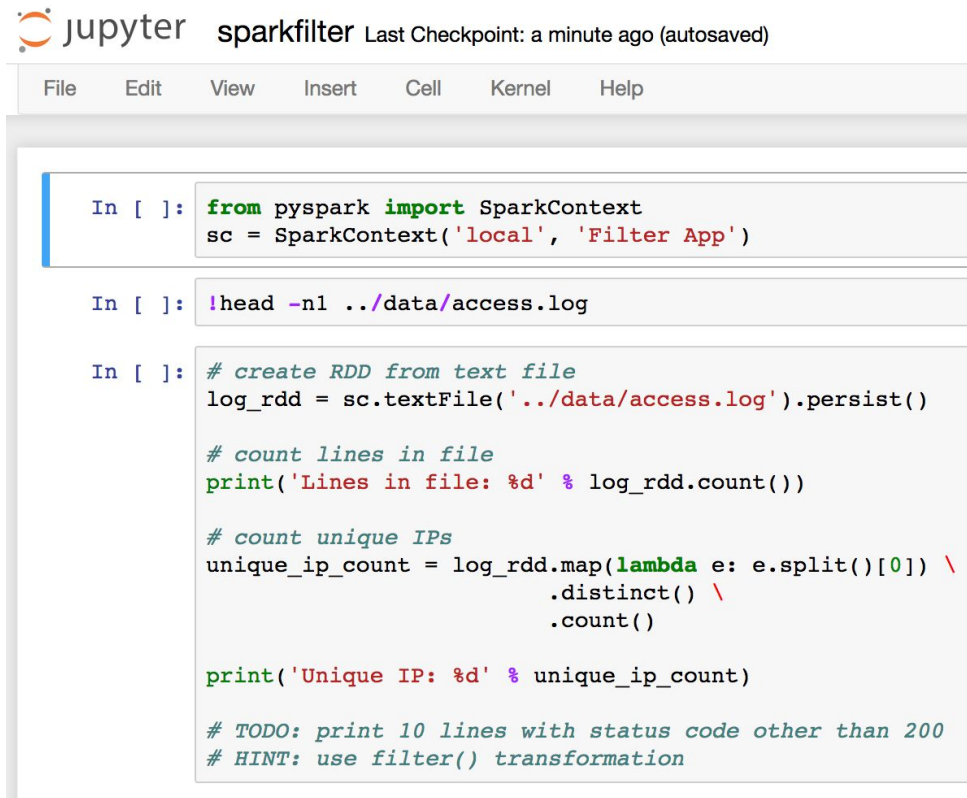
>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

>>> accum.value
10
```

Отладка программы

- Используйте часть данных для отладки, это позволит сократить время на поиск ошибок в программе
- Разбиение вычислений на части и получение промежуточных результатов вызовом функций-действий, например, `count()`
- В случае ошибок из-за нехватки памяти, попробуйте увеличить число партиций в данных
- Если ваша программа выполняется слишком медленно, попробуйте сократить число партиций
- Получение детальной информации из веб-интерфейса SparkUI

Пример: filter & count



The image shows a Jupyter Notebook interface with the title 'sparkfilter' and a status bar indicating 'Last Checkpoint: a minute ago (autosaved)'. The notebook has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. The main content area displays three code cells. The first cell imports SparkContext from pyspark and creates a local SparkContext named 'sc'. The second cell runs a shell command to view the first 10 lines of a log file. The third cell creates an RDD from the log file, counts the total lines, counts the unique IP addresses, and prints the results. It also includes TODO and HINT comments for further exploration.

```
In [ ]: from pyspark import SparkContext
        sc = SparkContext('local', 'Filter App')

In [ ]: !head -n1 ../data/access.log

In [ ]: # create RDD from text file
        log_rdd = sc.textFile('../data/access.log').persist()

        # count lines in file
        print('Lines in file: %d' % log_rdd.count())

        # count unique IPs
        unique_ip_count = log_rdd.map(lambda e: e.split()[0]) \
                                   .distinct() \
                                   .count()

        print('Unique IP: %d' % unique_ip_count)

        # TODO: print 10 lines with status code other than 200
        # HINT: use filter() transformation
```

Пример: accumulator

jupyter sparkaccumulator Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Help

Save + Copy Paste Undo Redo Code

```
In [ ]: from pyspark import SparkContext
sc = SparkContext('local', 'Accumulator App')

In [ ]: !head -n1 ../data/access.log

In [ ]: # create RDD from text file
log_rdd = sc.textFile('../data/access.log').persist()

# count lines with 'iphone' entry
accum_iphone = sc.accumulator(0)

log_rdd.foreach(lambda line: accum_iphone.add(line.lower().find('iphone') != -1))

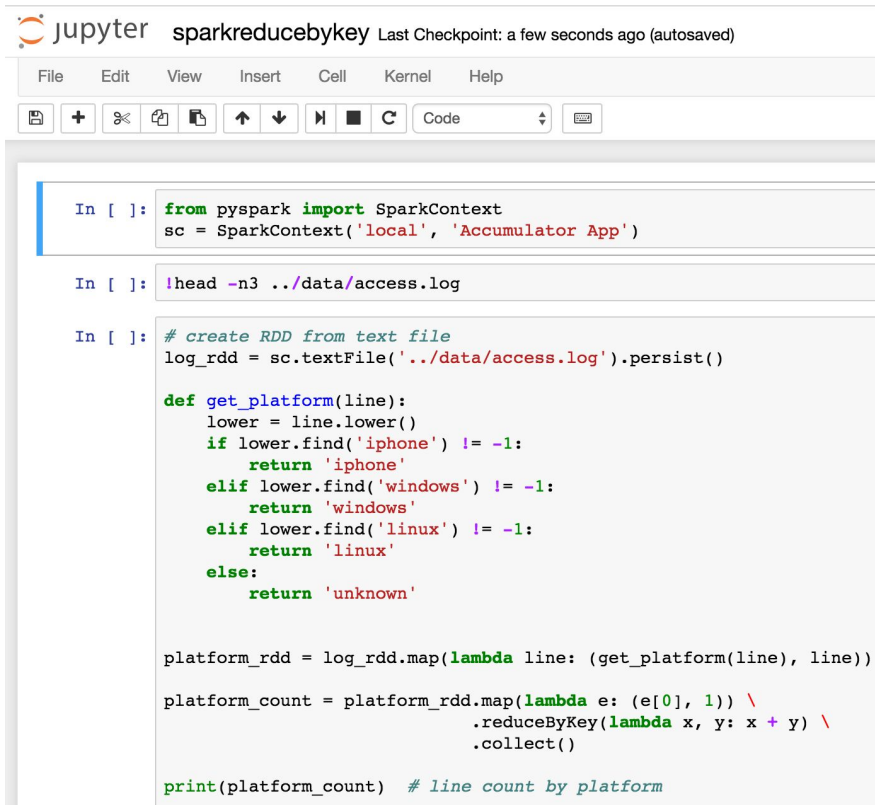
print('Number of lines with \'iphone\' phrase: %d' % accum_iphone.value)

# TODO: count lines with 'windows' phrase
# TODO: count lines with 'opera' phrase

# HINT: use separate accumulator for each task

# TODO: use global variables instead of accumulators and compare results
```

Пример: reduceByKey



The image shows a Jupyter Notebook interface with the title "sparkreducebykey" and a status bar indicating "Last Checkpoint: a few seconds ago (autosaved)". The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for saving, adding cells, deleting, copying, pasting, undo, redo, and a dropdown menu currently set to "Code".

```
In [ ]: from pyspark import SparkContext
        sc = SparkContext('local', 'Accumulator App')

In [ ]: !head -n3 ../data/access.log

In [ ]: # create RDD from text file
        log_rdd = sc.textFile('../data/access.log').persist()


        def get_platform(line):
            lower = line.lower()
            if lower.find('iphone') != -1:
                return 'iphone'
            elif lower.find('windows') != -1:
                return 'windows'
            elif lower.find('linux') != -1:
                return 'linux'
            else:
                return 'unknown'

        platform_rdd = log_rdd.map(lambda line: (get_platform(line), line))












        platform_count = platform_rdd.map(lambda e: (e[0], 1)) \
            .reduceByKey(lambda x, y: x + y) \
            .collect()

        print(platform_count) # line count by platform
```

Пример: join

 jupyter sparkjoin Last Checkpoint: a few seconds ago (autosaved)

File Edit View Insert Cell Kernel Help

          Code 

```
In [ ]: from pyspark import SparkContext
        sc = SparkContext('local', 'Join App')

In [ ]: !head -n1 ../data/access.log

In [ ]: !head -n5 ../data/ip.tsv

In [ ]: # create RDD from text file
        log_rdd = sc.textFile('../data/access.log').persist()
        ip_rdd = sc.textFile('../data/ip.tsv').persist()

        # TODO: count requests per city

        # HINT: make (k, v) pairs from source files and join by ip
```

Задание: K-means кластеризация

1. Initialize **cluster centroids** $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

For every i , set


$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

For each j , set











$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

}

Задание: K-means кластеризация

 jupyter sparkkmeans Last Checkpoint: 2 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

         Code 

```
In [ ]: import numpy as np

n_clusters = 3
n_features = 2

np.random.seed(123456)

def parse_line(line):
    return np.array([float(v) for v in line.split(',')])

def random_cluster(features):
    return np.random.randint(0, n_clusters), features

def nearest_cluster(features, centroids):
    distances = np.linalg.norm(centroids - features, axis=1)
    return np.argmin(distances)

# create features RDD from text file
features_rdd = sc.textFile('../data/kmeans.csv').map(parse_line)

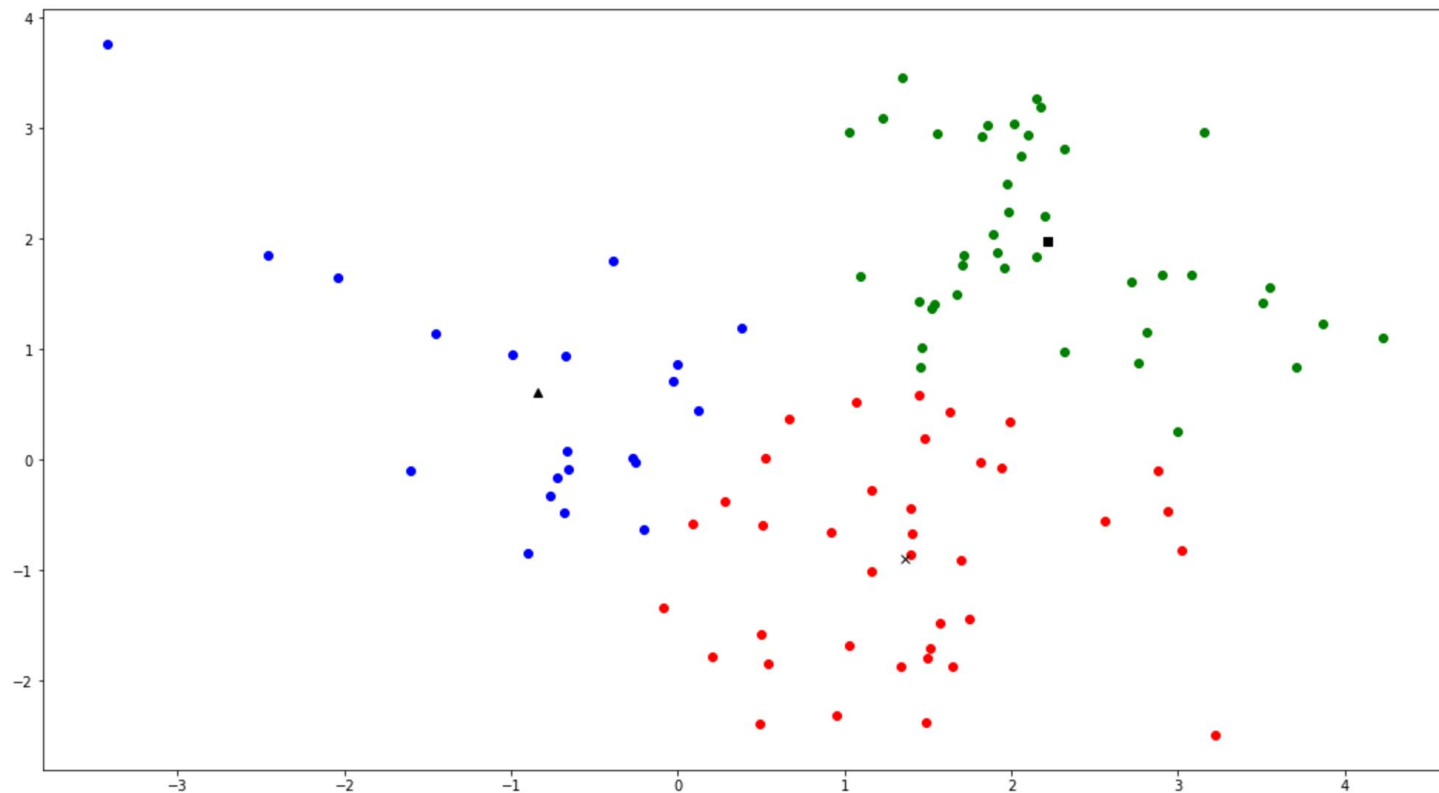
# assign random cluster
labels_rdd = features_rdd.map(random_cluster)

for i in range(10):

    # TODO: update centroids by averaging appropriate object coordinates
    # HINT: use reduceByKey and countByKey

    # TODO: update object labels
    # HINT: use nearest_cluster() here
```

Задание: K-means кластеризация



Полезные материалы

- Spark Programming Guide
<http://spark.apache.org/docs/latest/programming-guide.html>
- Functional programming
https://en.wikipedia.org/wiki/Functional_programming
- Spark Architecture: Shuffle
<https://0x0fff.com/spark-architecture-shuffle/>
- 7 Tips on debugging
<https://databricks.com/blog/2016/10/18/7-tips-to-debug-apache-spark-code-faster-with-databricks.html>
- Understanding your Apache Spark Application Through Visualization
<https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html>