

Анализ больших данных с Apache Spark

Лекция 1. Парадигма MapReduce и концепция вычислений на Spark

Мурашкин Вячеслав
2017

<https://github.com/a4tunado/lectures-hse-spark/tree/master/express/001>

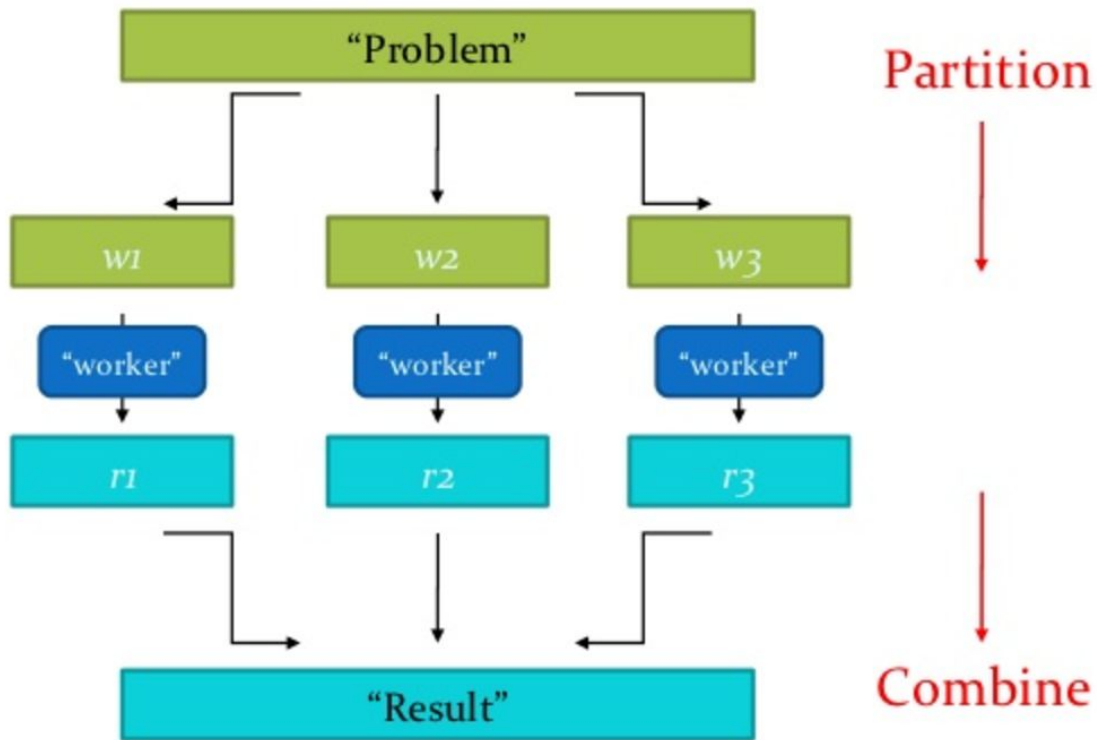
Лекция 1.1 Парадигма MapReduce

- Почему MapReduce?
- Парадигма MapReduce
- Распределенное хранилище: HDFS
- Экосистема приложений Hadoop

Почему MapReduce?

- Алгоритмы работающие на одном сервере сложно масштабировать при увеличении объема данных
- В случае разделения вычислительных ресурсов и хранилища возникает узкое место при переносе данных, решение: перенести вычисления к данным
- Сервер с мощным CPU, как правило, стоит дороже, чем эквивалентное число более слабых машин
- Возникла потребность в **отказоустойчивом** решении для обработки большого объема данных **дешево** с простым способ **масштабирования** без изменения логики работы алгоритмов

Парадигма MapReduce



Парадигма MapReduce

- Данные на входе и выходе алгоритма всегда передаются в виде **набора** пар **<key, value>**
- Для обработки данных необходимо реализовать **две** функции:

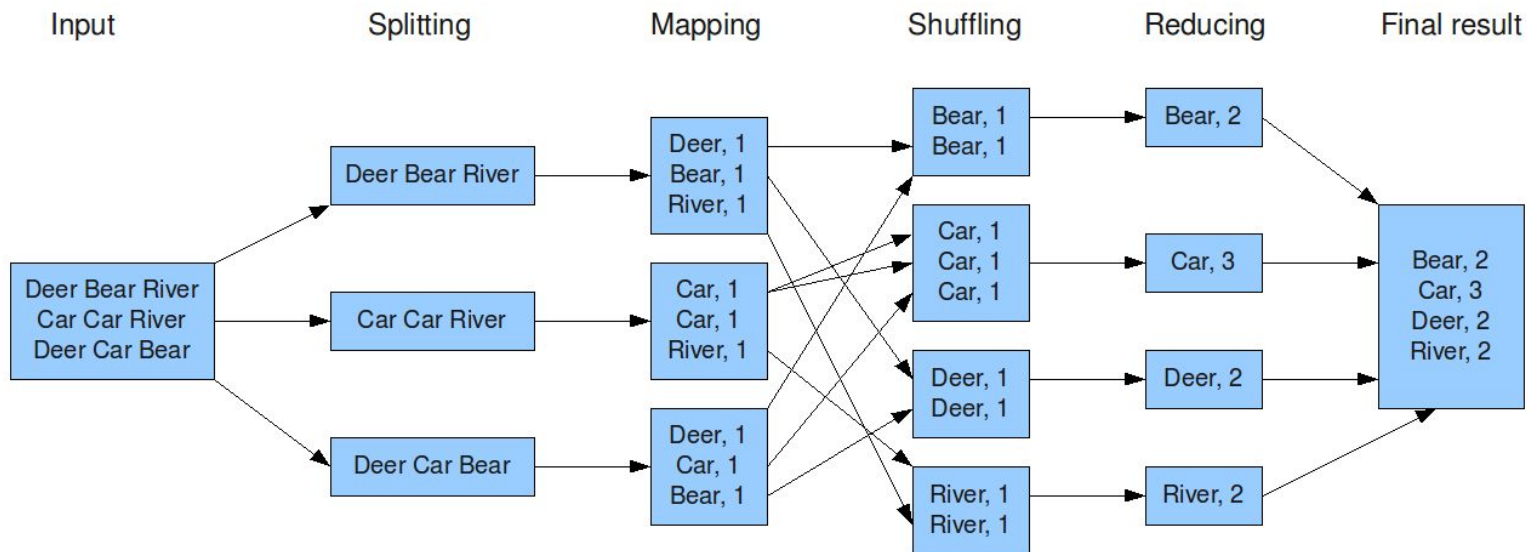
`map (in_key, in_value) -> list(out_key, intermediate_value)`

- принимает на входе пару `<in_key, in_value>`
- возвращает список пар с промежуточными значениями `<out_key, intermediate_value>`

`reduce (out_key, list(intermediate_value)) -> list(out_key, out_value)`

- обрабатывает промежуточные значения для ключа `out_key`
- возвращает результат обработки в виде списка (как правило из одного элемента)

Пример: подсчет частот слов в корпусе



Пример: подсчет частот слов в корпусе

```
map(String input_key, String input_value):
```

```
// input_key: document name; input_value: document contents
```

```
for each word w in input_value:
```

```
    EmitIntermediate(w, "1");
```

```
reduce(String output_key, Iterator intermediate_values):
```

```
// output_key: a word; output_values: a list of counts
```

```
int result = 0;
```

```
for each v in intermediate_values:
```

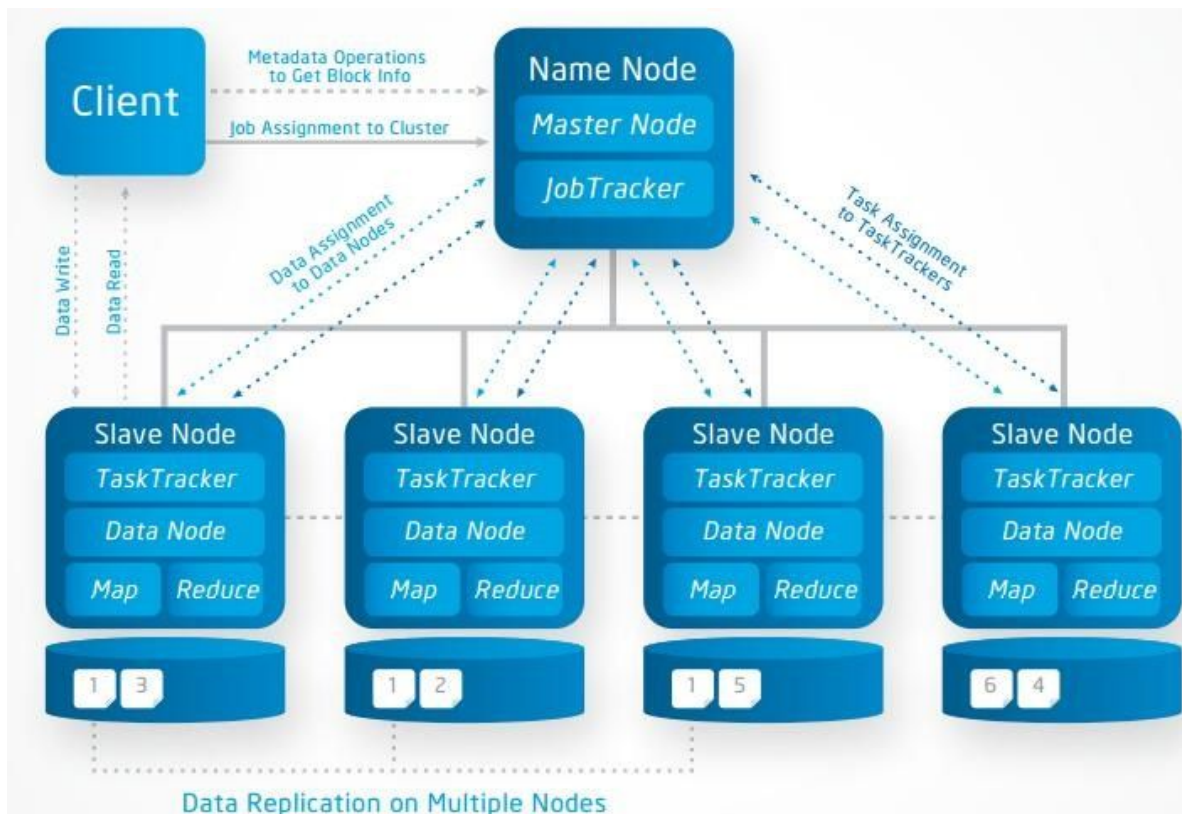
```
    result += ParseInt(v);
```

```
Emit(output_key, AsString(result));
```

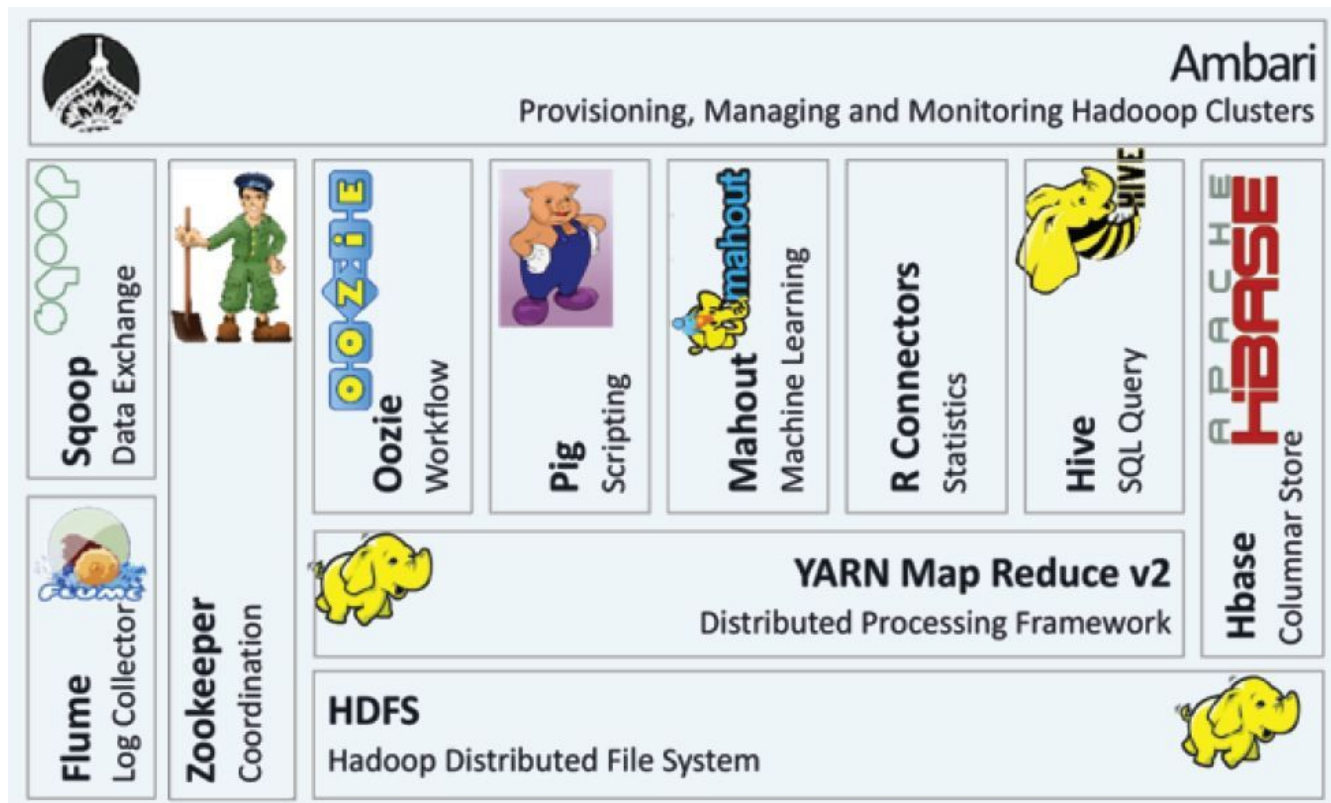
Плюсы и минусы

- + Map и reduce стадии легко параллелятся
- + Линейное масштабирование
- + Отказоустойчивость: независимая обработка данных, в случае падения одной машины не нужно перезапускать весь процесс заново
- Данные после каждого этапа сохраняются на диск (тратим время на запись/чтение данных)
- Необходимо копировать данные после map стадии (группировка по ключу)

Распределенное хранилище: HDFS



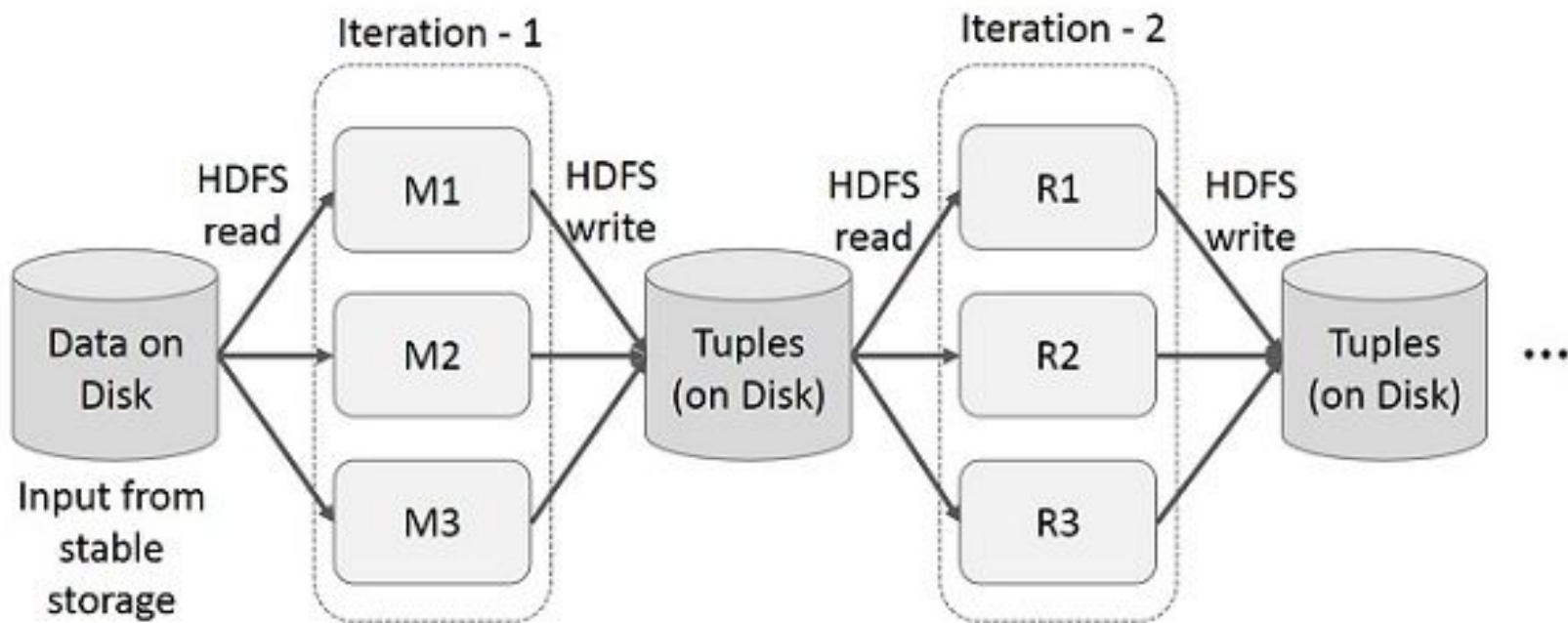
Экосистема Hadoop



Лекция 1.2 Концепция вычислений на Spark

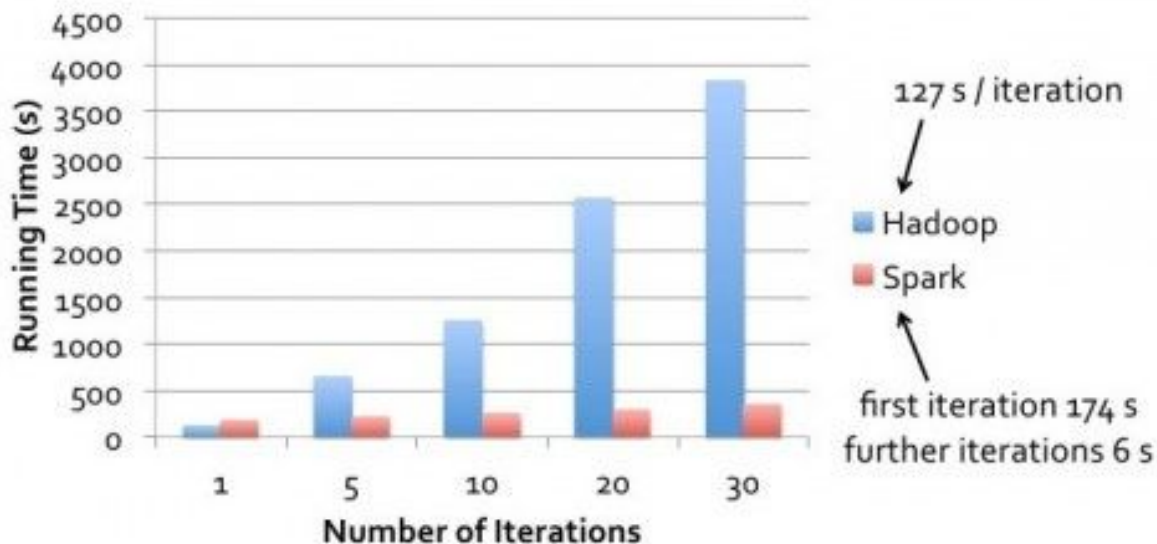
- Недостатки MapReduce
- Распределенные вычисления в памяти
- RDD (Resilient Distributed Datasets)
- Архитектура Spark
- Концепция вычислений на Spark
- Задание: реализовать обучение модели логистической регрессии на Spark

Недостатки MapReduce



Распределенные вычисления в памяти

Logistic Regression Performance



RDD (Resilient Distributed Datasets)

- Основной способ представления данных в Spark
- Read-only
- Представляет собой набор объектов одного типа
- Каждый набор данных в RDD может быть разбит на логические части (chunk) и размещен на разных машинах кластера
- Может храниться как на жестком диске, так и в оперативной памяти
- Новый RDD можно получить загрузив данные из распределенного хранилища (HDFS), либо путем преобразования другого RDD
- https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf

RDD (Resilient Distributed Datasets)

- Управление настройками RDD
 - [partitioning](#) (repartition)
 - способ разбиения данных на логические части (chunk)
 - [persist](#) (MEMORY_ONLY, MEMORY_AND_DISK, ...)
 - определяет место хранения данных, полезно если все данные не помещаются в память
 - [unpersist](#)
 - удаляет данные и высвобождает память

RDD (Resilient Distributed Datasets)

- Поддерживаются два типа операций:
 - преобразования (map, reduceByKey, join, ...)
 - задают преобразование отдельно для каждой строки
 - возвращают ссылку на новый RDD
 - вычисления по требованию (lazy computations)
 - сохраняется последовательность преобразований для восстановления в случае отказа ноды кластера
 - действия (min, max, count, ...)
 - иницируют расчет и возвращают значения

Функции-преобразования (transformations) RDD

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

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*)

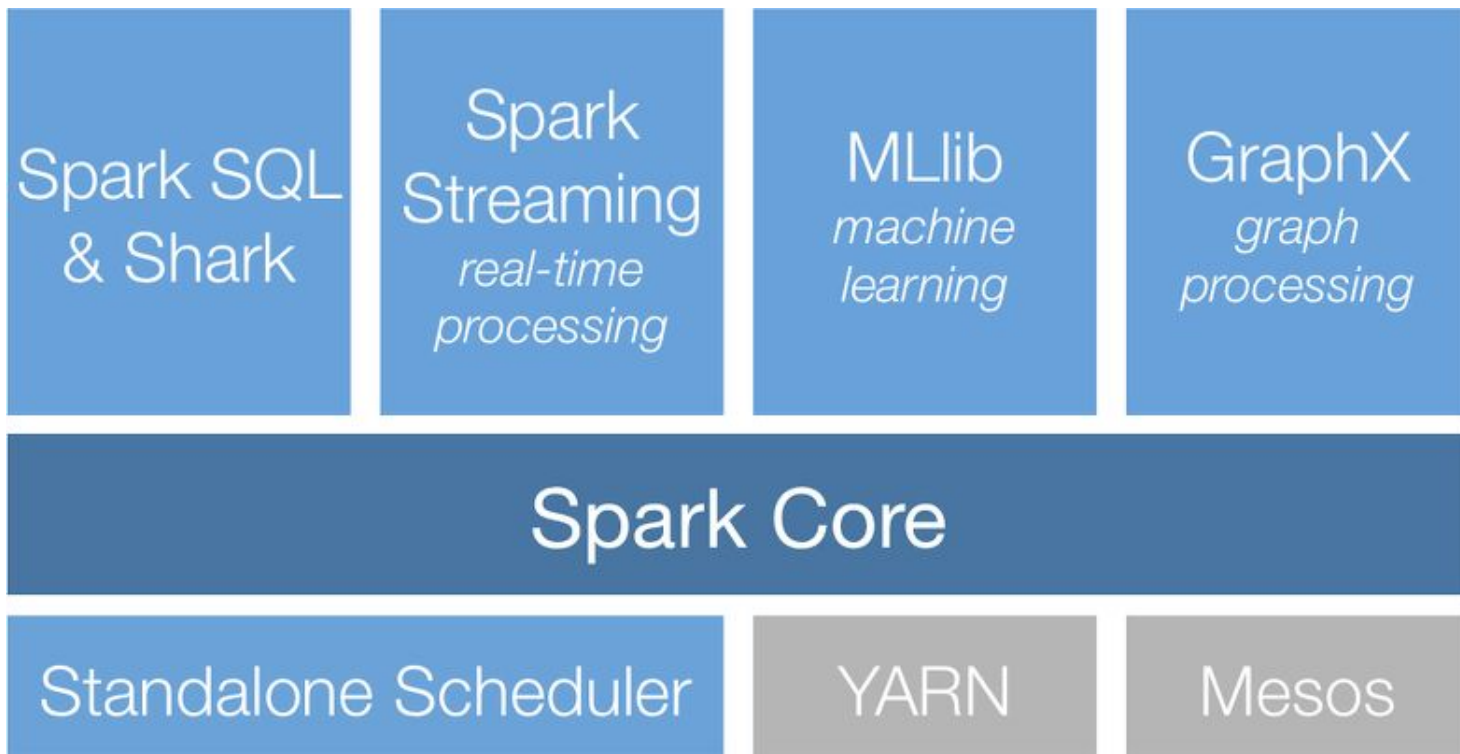
Run a function *func* on each element of the dataset. This is usually done for side effects such as updating an [Accumulator](#) or interacting with external storage systems.

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

Apache Spark

- Высокопроизводительная система распределенных вычислений
- Основная часть реализована на *функциональном* языке Scala
- Поддерживает API для Python, Java, Scala и R
- Основные компоненты
 - SparkSQL
 - MLlib
 - Spark Streaming
 - GraphX


Apache Spark



Концепция вычислений на Spark - SparkContext

- SparkContext
 - один экземпляр на приложение
 - создание RDD и хранение мета информации
 - `sc.parallelize`
 - `sc.textFile`
 - `sc.wholeTextFiles`
 - отвечает за запуск приложений на Spark кластере
 - управление настройками и процессом выполнения приложений
 - получение информации о статусе выполнения приложений

Концепция вычислений на Spark

 jupyter sparkwordcount Last Checkpoint: 7 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

```
In [ ]: from pyspark import SparkContext
        # creating spark context
        sc = SparkContext('local', 'WordCount App')
```

```
In [ ]: # loading text from file
        with open('../data/idiot.txt') as src:
            text = src.readlines()

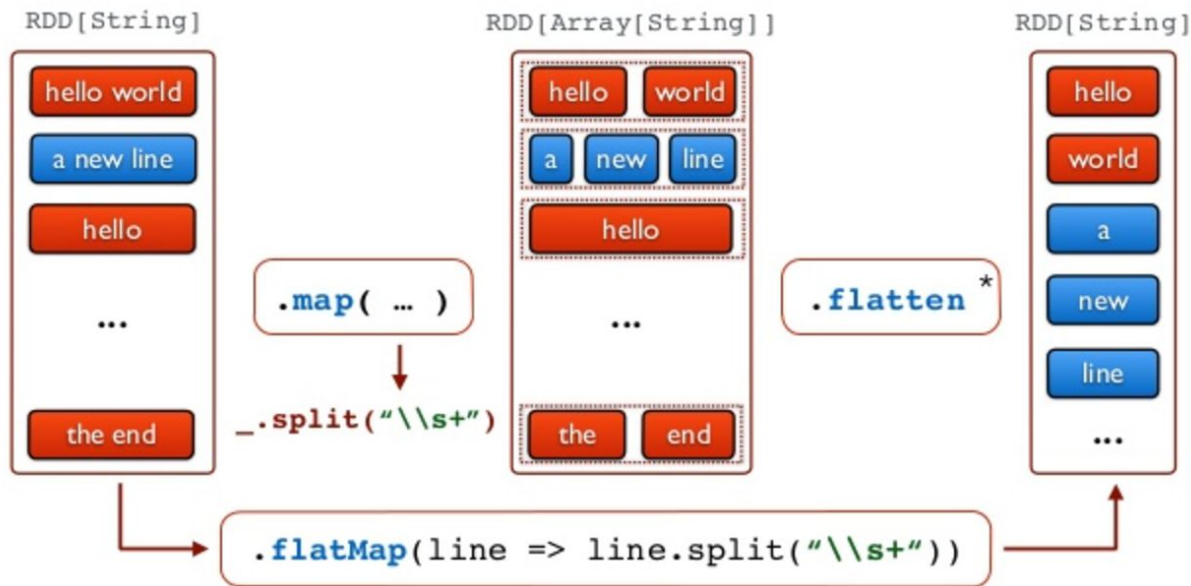
        # making RDD from text lines
        text_rdd = sc.parallelize(text)

        # counting word entries and storing result as RDD
        wc_rdd = text_rdd.flatMap(lambda line: line.split()) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda x, y: x + y)

        # getting result back to client
        wc = wc_rdd.collect()

        # output 10 most frequent words
        print sorted(wc, key=lambda e: -e[-1])[ :10]
```

Концепция вычислений на Spark - flatMap



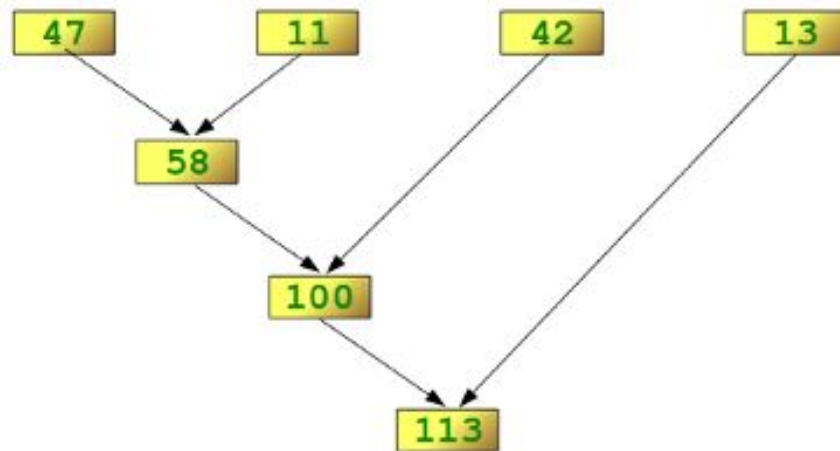
```
// Step 3 - Split lines into words  
val words = lower.flatMap(line => line.split("\\s+"))
```

Note: `flatten()` not available in spark, only `flatMap`

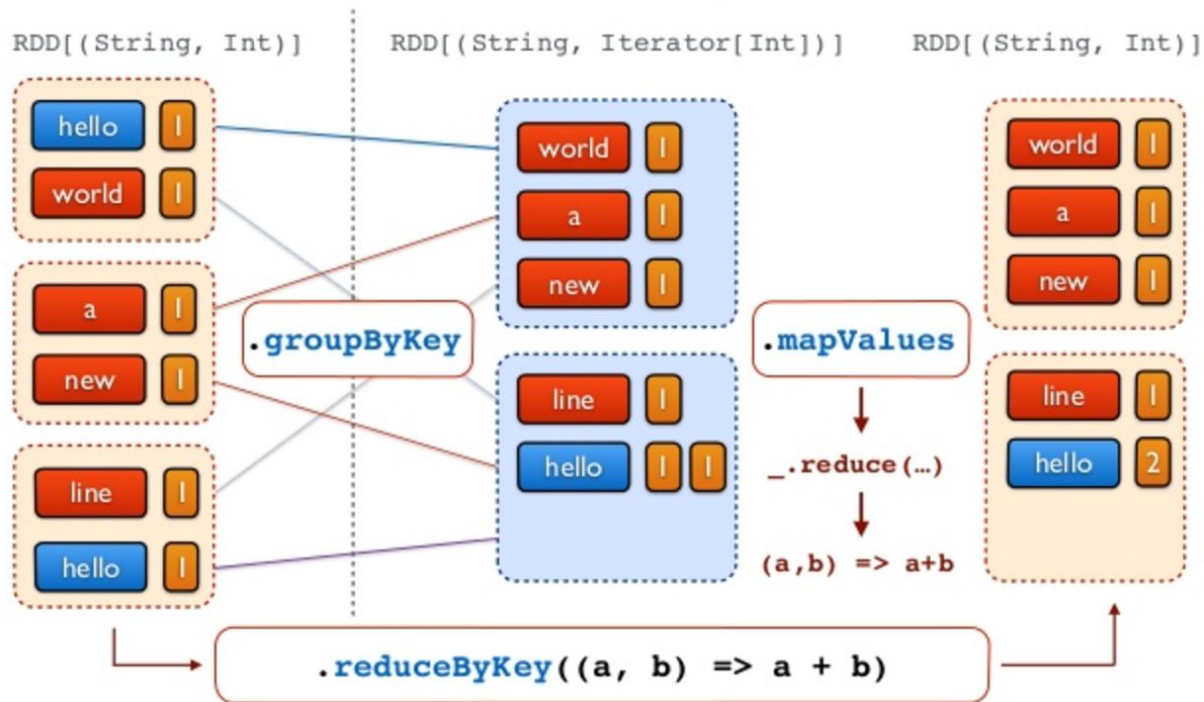
Концепция вычислений на Spark - reduceByKey

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

func(func(func(47, 11), 42), 13)



Концепция вычислений на Spark - reduceByKey

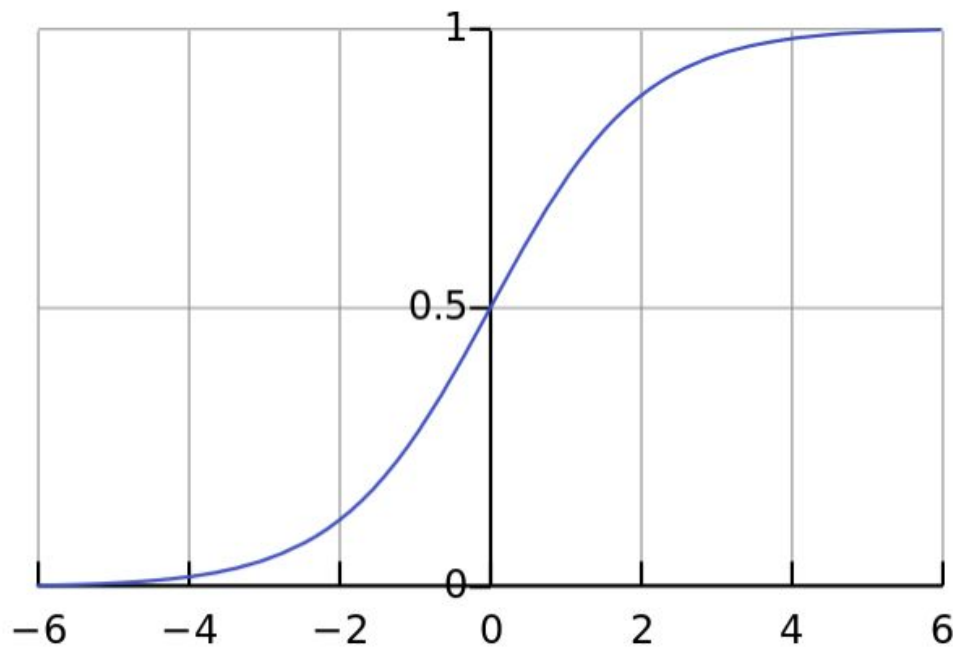


```
// Step 5 - Count all words  
val freq = counts.reduceByKey(_ + _)
```

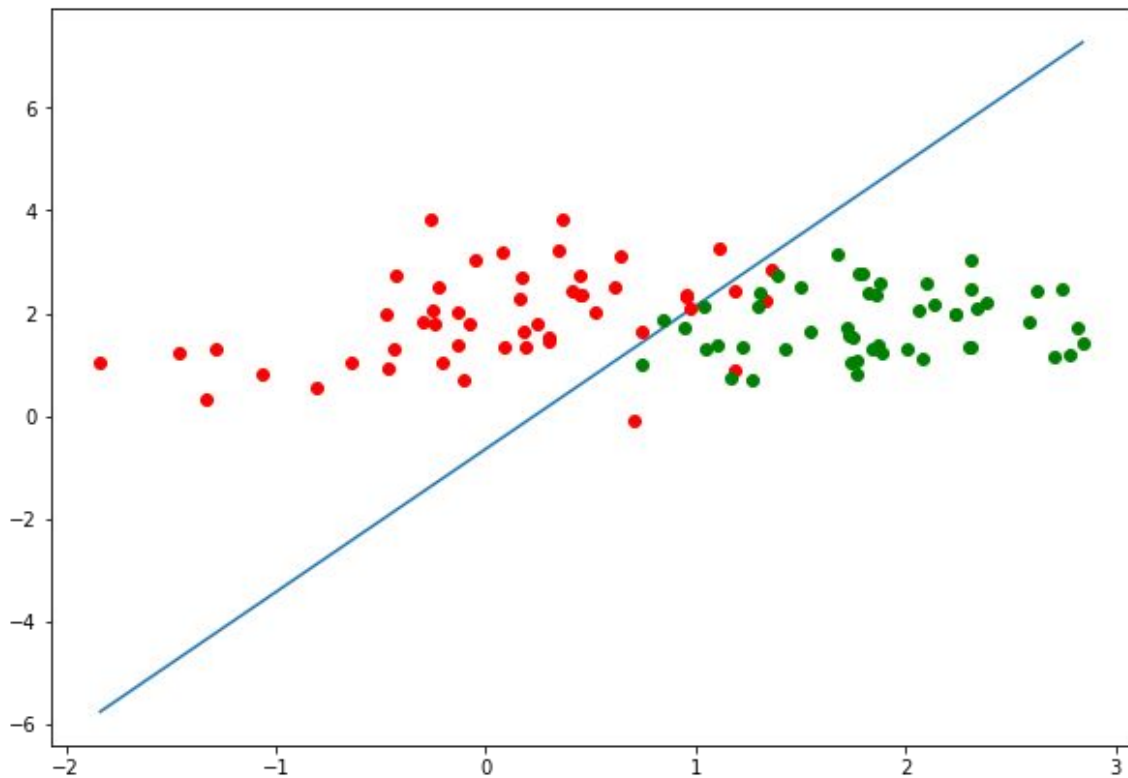
Задание: логистическая регрессия

$$y(x, w) = \frac{1}{1 + \exp(-xw^T)}$$

$$w = w - \lambda \frac{1}{N} \sum_{i=1}^N (y(x_i) - t_i) x_i$$



Задание: логистическая регрессия



Полезные материалы

- Advanced Analytics with Spark
<http://shop.oreilly.com/product/0636920035091.do>
- Mastering Apache Spark 2
<https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details>
- Spark Overview
<http://spark.apache.org/>
- Cluster Mode Overview
<http://spark.apache.org/docs/latest/cluster-overview.html>
- Tuning Spark
<http://spark.apache.org/docs/latest/tuning.html>

Полезные материалы

- Databricks Blog
<https://databricks.com/blog>
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
https://cs.stanford.edu/~matei/papers/2012/nsdi_spark.pdf
- Spark Misconceptions
<https://0x0fff.com/spark-misconceptions/>
- Spark Architecture: Shuffle
<https://0x0fff.com/spark-architecture-shuffle/>