**Московский государственный технический университет им. Н.Э.Баумана**

Защищено:
Гапанюк Ю.Е.

"__"_____2016 г.

# Отчет по домашнему заданию №1 по курсу РИП

8
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы **ИУ5-54** _____
(подпись)

**Наседкин И.А.**

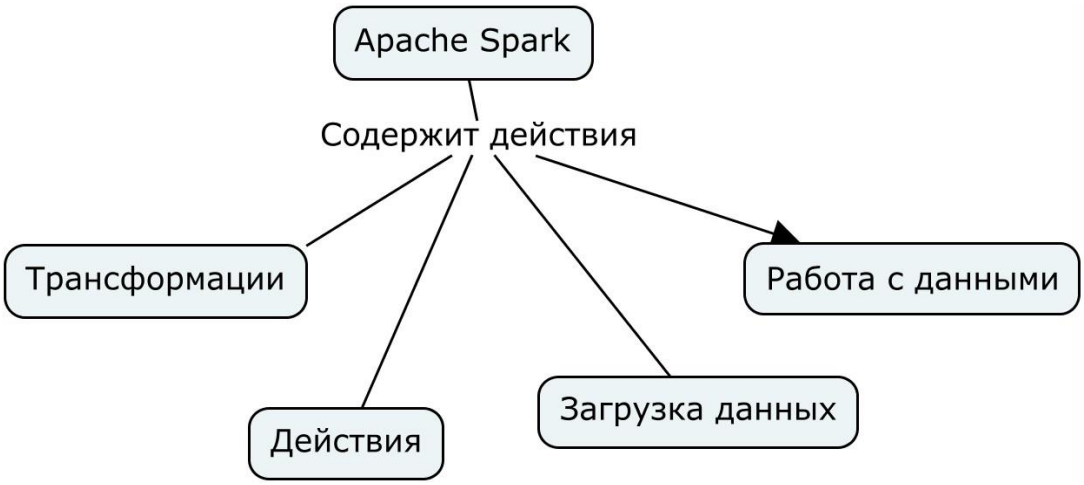
"__"_____2016 г.

Москва, МГТУ - 2016

**СОДЕРЖАНИЕ**

# Apache Spark

**Концептуальная карта**




## Введение

## Что такое Spark?

Когда исследователи университета Беркли начинали разрабатывать Spark, они держали в голове уже сложившуюся к тому времени картину мира больших данных – принцип параллельных вычислений на кластере компьютеров, способном к расширению в ширину, позволял работать с большими объемами информации, но жертвовал скоростью работы в пользу горизонтальной масштабируемости и способности к быстрому восстановлению после системных ошибок. Здесь стоит вспомнить о нашумевшей (и до сих пор популярной) технологии MapReduce, которая примерно до февраля этого года была основной движущей силой экосистемы Apache Hadoop. Действительно эффективная для пакетной обработки данных, эта модель, однако, обладает существенными недостатками.

- **Во-первых,** в угоду повышения отказоустойчивости, MapReduce предполагает хранение данных на

жестких дисках – а это непременно означает снижение скорости обработки информации.

- **Во-вторых,** MapReduce относится к классу однопроходных моделей вычисления, что делает ее не слишком пригодной, например, для итерационных вычислений и интерактивного анализа данных.

Все это имели в виду разработчики Spark – и их продукт на выходе получился настолько удачным и перспективным, что на данный момент именно это средство является одновременно флагманом фонда Apache и его главным козырем в рукаве.

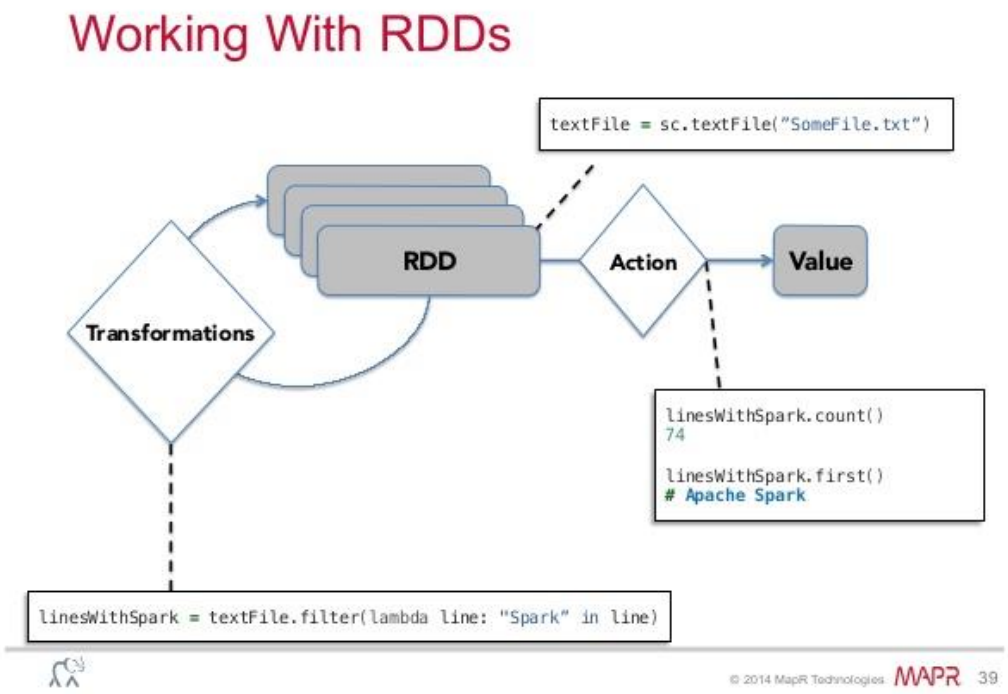
Строго говоря, Apache Spark – это высокопроизводительное средство обработки данных, работающее в кластере Hadoop – и призванное заменить (или дополнить – зависит от точки зрения) технологию MapReduce, путем обобщения и модификации использованных для ее функционирования технологий. Ключевых момента здесь два:

1. Повышение скорости обработки данных (до 100 раз по сравнению с MapReduce при условии работы в оперативной памяти и до 10 раз при условии взаимодействия системы с жестким диском) посредством уменьшения количества операций чтения и записи на жесткий диск – теперь существенная часть операций производится в реальном времени.
2. Такая технология обработки информации стала возможной благодаря хранению информации о каждом операторе в оперативной памяти. Это дает и еще одно преимущество – все процессы, связанные с данными (в том числе, обработка потоков и пакетов информации и машинное обучение) происходят на одном и том же кластере данных, в одном и том же приложении.

Нативно Spark поддерживает **Scala**, **Python** и **Java**.

Основным понятием в Spark'е является **RDD (Resilient Distributed Dataset)**, который представляет собой Dataset, над которым можно делать преобразования двух типов (и, соответственно, вся работа с этими

структурами заключается в последовательности этих двух действий - Трансформаций и Действий).




## Трансформации

Результатом применения данной операции к RDD является новый RDD. Как правило, это операции, которые каким-либо образом преобразовывают элементы данного датасета. Вот неполный самых распространенных преобразований, каждое из которых возвращает новый датасет (RDD):

**.map(function)** — применяет функцию function к каждому элементу датасета

**.filter(function)** — возвращает все элементы датасета, на которых функция function вернула истинное значение

**.distinct([numTasks])** — возвращает датасет, который содержит уникальные элементы исходного датасета

Также присутствуют операции над множествами, смысл которых понятен из названий:

**.union(otherDataset)**

**.intersection(otherDataset)**

**.cartesian(otherDataset)** — новый датасет содержит в себе всевозможные пары (A,B), где первый элемент принадлежит исходному датасету, а второй — датасету-аргументу

## Действия

Действия применяются тогда, когда необходимо материализовать результат — как правило, сохранить данные на диск, либо вывести часть данных в консоль. Вот список самых распространенных действий, которые можно применять над RDD:

**.saveAsTextFile(path)** — сохраняет данные в текстовый файл (в hdfs, на локальную машину или в любую другую поддерживаемую файловую систему — полный список можно посмотреть в документации)

**.collect()** — возвращает элементы датасета в виде массива. Как правило, это применяется в случаях, когда данных в датасете уже мало (применены различные фильтры и преобразования) — и необходима визуализация, либо дополнительный анализ данных, например средствами пакета Pandas

**.take(n)** — возвращает в виде массива первые n элементов датасета

**.count()** — возвращает количество элементов в датасете

**.reduce(function)** — знакомая операция для тех, кто знаком с **MapReduce**. Из механизма этой операции следует, что функция function (которая принимает на вход 2 аргумента возвращает одно значение) должна быть обязательно коммутативной и ассоциативной

Это основы, которые необходимо знать при работе с инструментом.

При запуске Spark, первое, что необходимо сделать — это создать **SparkContext** (если говорить простыми словами — это обьект, который отвечает за реализацию более низкоуровневых операций с кластером — подробнее — см. документацию), который при запуске **Spark-Shell** создается автоматически и доступен сразу (обьект **sc**)

## Загрузка данных

Загружать данные в Spark можно двумя путями:

**а).** Непосредственно из локальной программы с помощью функции **.parallelize(data)**

```
localData = [5,7,1,12,10,25]
ourFirstRDD = sc.parallelize(localData)
```

**б).** Из поддерживаемых хранилищ (например, hdfs) с помощью функции **.textFile(path)**

```
ourSecondRDD = sc.textFile("path to some data on the cluster")
```

В этом пункте важно отметить одну особенность хранения данных в Spark'e и в тоже время самую полезную функцию **.cache()** (отчасти благодаря которой Spark стал так популярен), которая позволяет закэшировать данные в оперативной памяти (с учетом доступности последней). Это позволяет производить итеративные вычисления в

оперативной памяти, тем самым избавившись от IO-overhead'a. Это особенно важно в контексте машинного обучения и вычислений на графах, т.к. большинство алгоритмов итеративные — начиная от градиентных методов, заканчивая такими алгоритмами, как **PageRank**

## Работа с данными

После загрузки данных в RDD мы можем делать над ним различные трансформации и действия, о которых говорилось выше. Например:

Посмотрим первые несколько элементов:

```
for item in ourRDD.top(10):
    print item
```

Либо сразу загрузим эти элементы в Pandas и будем работать с DataFrame'ом:

```
import pandas as pd
pd.DataFrame(ourRDD.map(lambda x: x.split(";")[:]).top(10))
```

Spark, как видно из примеров, очень удобен.

Еще один пример - пример трансформации, а именно, вычисление максимального и минимального элементов датасета. Сделать это можно, например, с помощью функции **.reduce()**:

```
localData = [5,7,1,12,10,25]
ourRDD = sc.parallelize(localData)
print ourRDD.reduce(max)
print ourRDD.reduce(min)
```

В работе были рассмотрены основные понятия, необходимые для работы с инструментом. Не была рассмотрена работа с SQL, работа с парами **<ключ, значение>** (что делается легко — для этого достаточно сначала применить к RDD, например, фильтр, чтобы выделить, ключ, а дальше — уже легко пользоваться встроенными функциями, вроде **sortByKey**, **countByKey**, **join** и др.)