Лекция 8

Hadoop MapReduce

&

Apache Spark

Big Data

Определения (не верные):

- Big Data – это когда данных больше, чем 100Гб (500Гб, 1ТБ, кому что нравится)
- Big Data – это такие данные, которые невозможно обрабатывать в Excel
- Big Data – это такие данные, которые невозможно обработать на одном компьютере
- Big Data – это любые данные.

Определение:

Big Data - это данные, имеющие следующие характеристиками:

- ❑ большой объем;
- ❑ большую скорость поступления;
- ❑ разрозненность источников;
- ❑ не структурированность.

Big Data - примеры

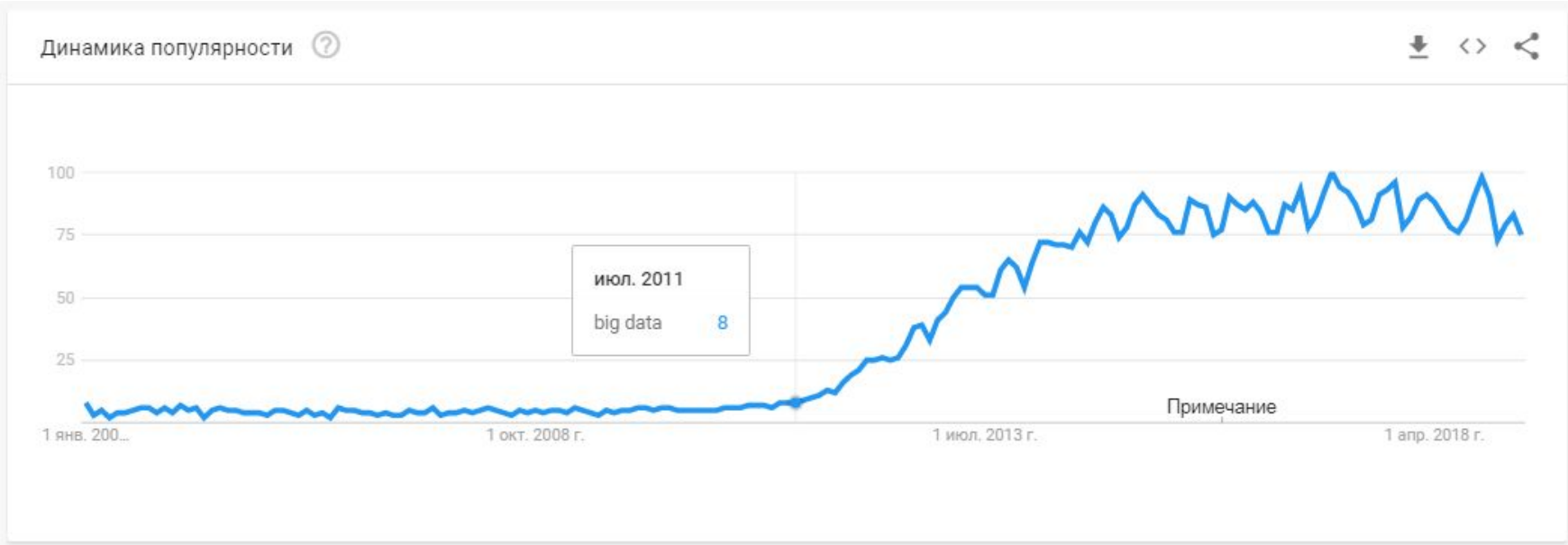
Большие данные:

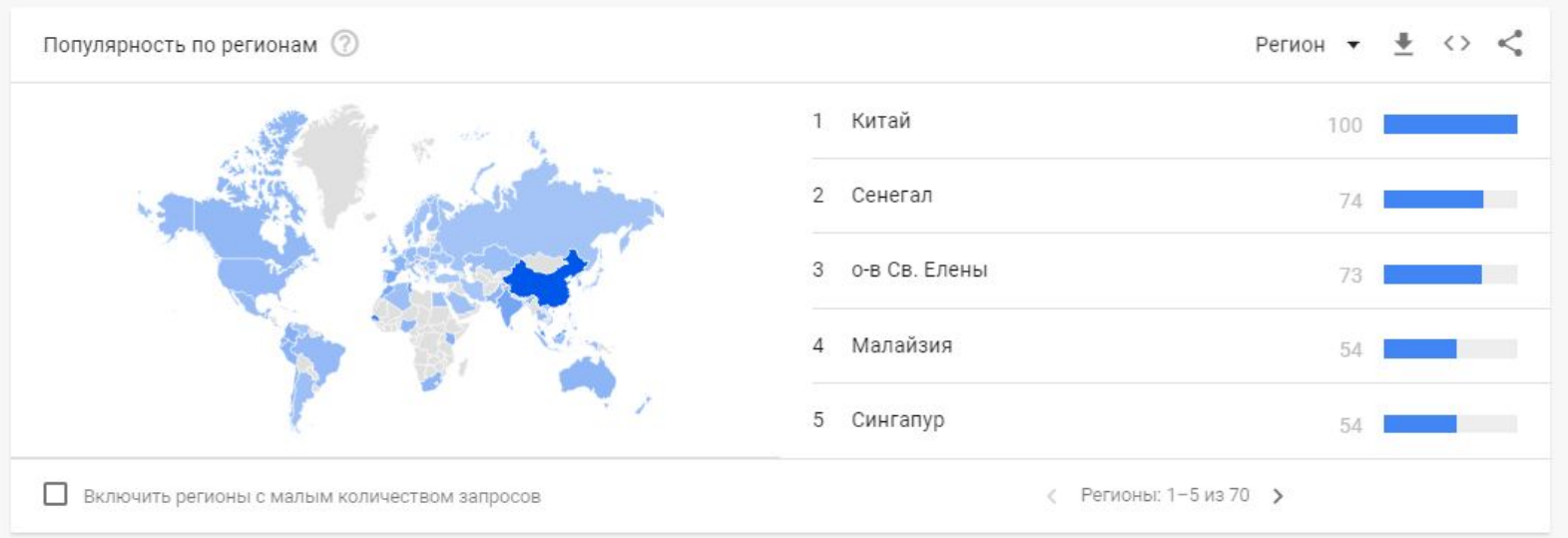
- Данные с медицинских носимых устройств;
- Данные с медицинских аппаратов;
- Данные из медицинских информационных систем

Данные большого объема:

- GPS-сигналы от автомобилей для транспортной компании
- Данные, снимаемые с датчиков в большом адронном коллайдере
- Оцифрованные книги в Российской Государственной Библиотеке
- Информация о транзакциях всех клиентов банка
- Информация о всех покупках в крупной ритейл сети и т.д.

Популярность термина



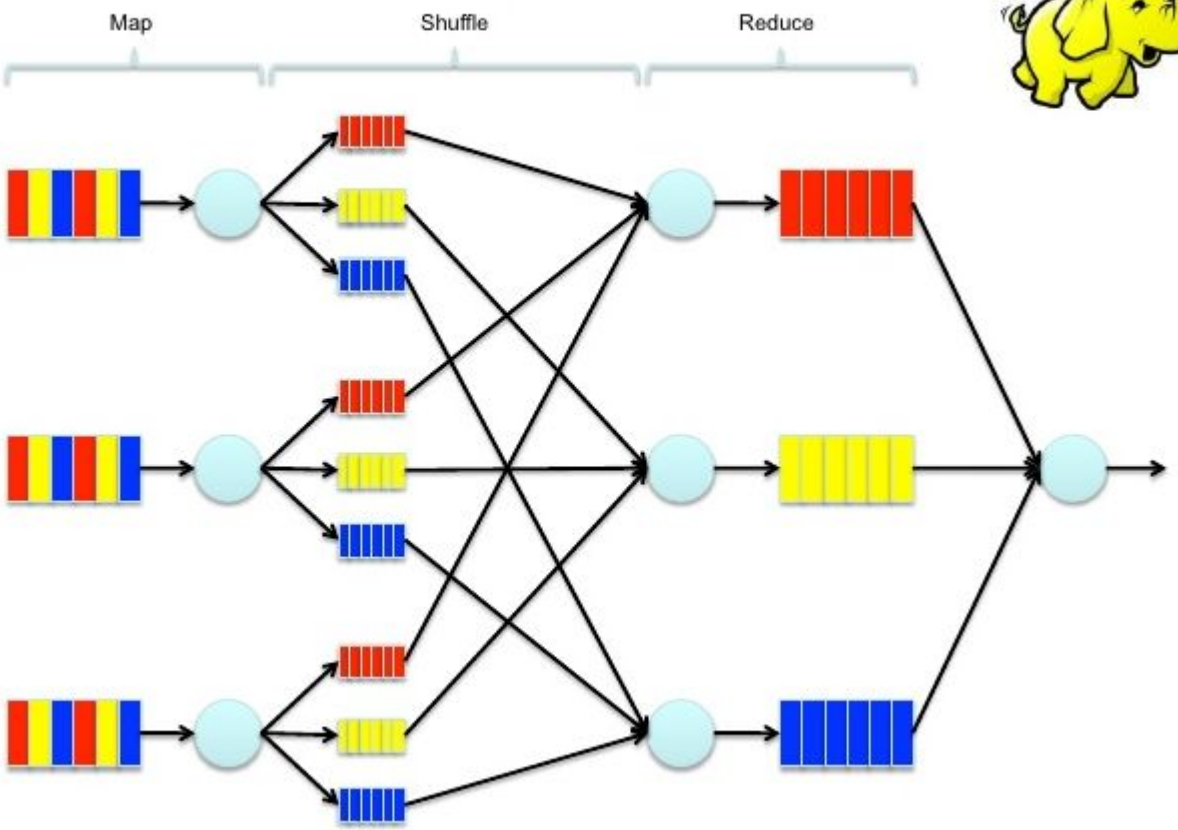


https://trends.google.com/trends/explore?q=big%20data

MapReduce

Парадигма MapReduce

MapReduce – это модель распределенной обработки данных, предложенная компанией Google для обработки больших объёмов данных на компьютерных кластерах.

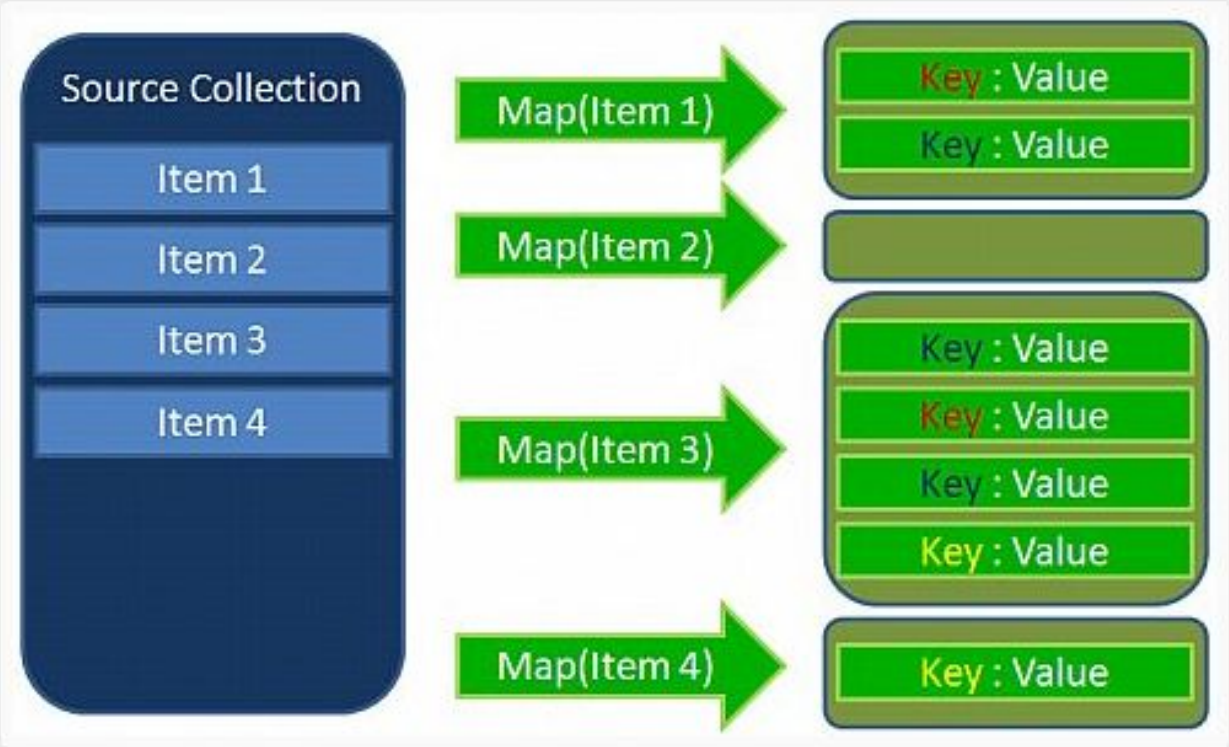


(adapted from http://code.google.com/p/mapreduce-framework/wiki/MapReduce)

Типичная реализация подхода MapReduce

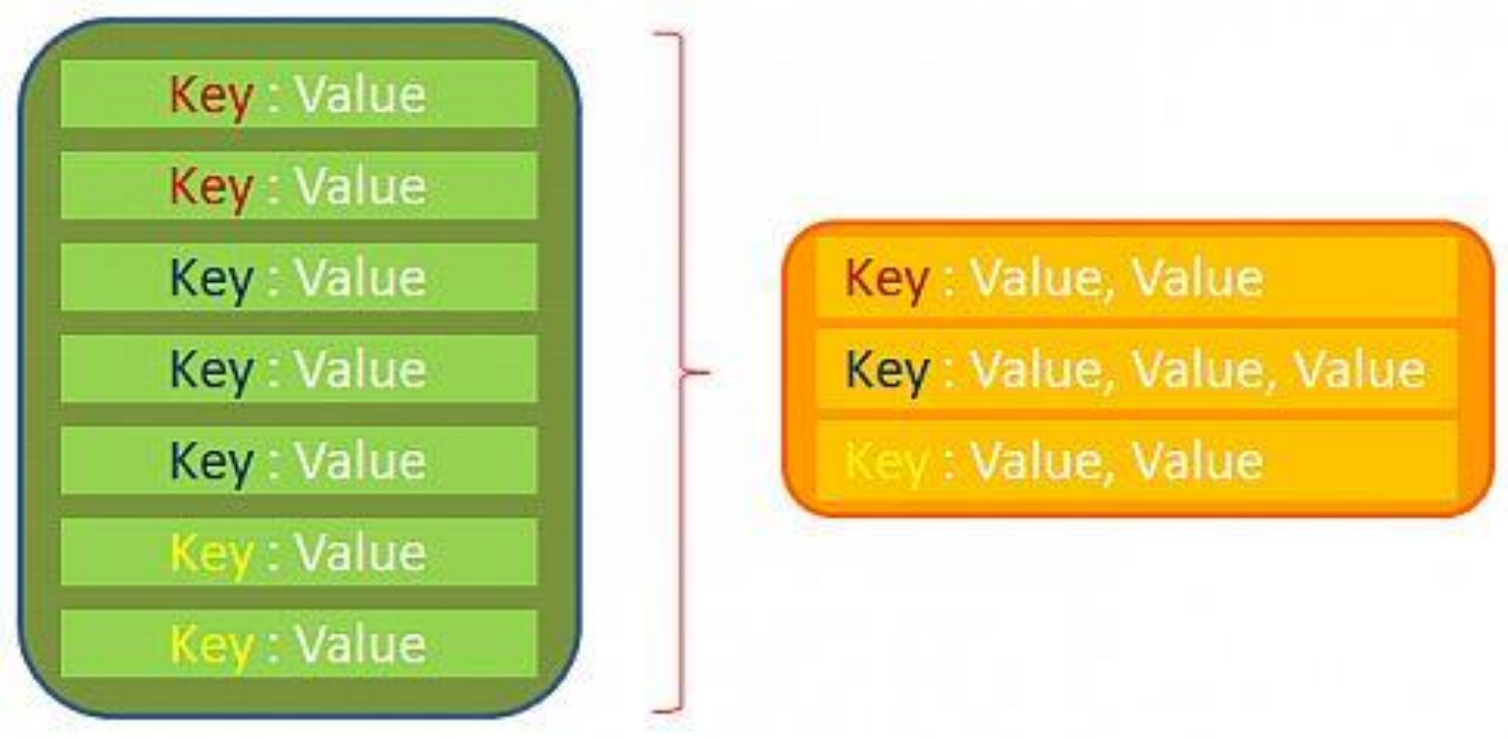
Подход состоит из нескольких стадий.

1. Применение Map-функции к каждому элементу исходной коллекции. Map-функция вернет ноль либо создаст экземпляры коллекции Key/Value объектов.



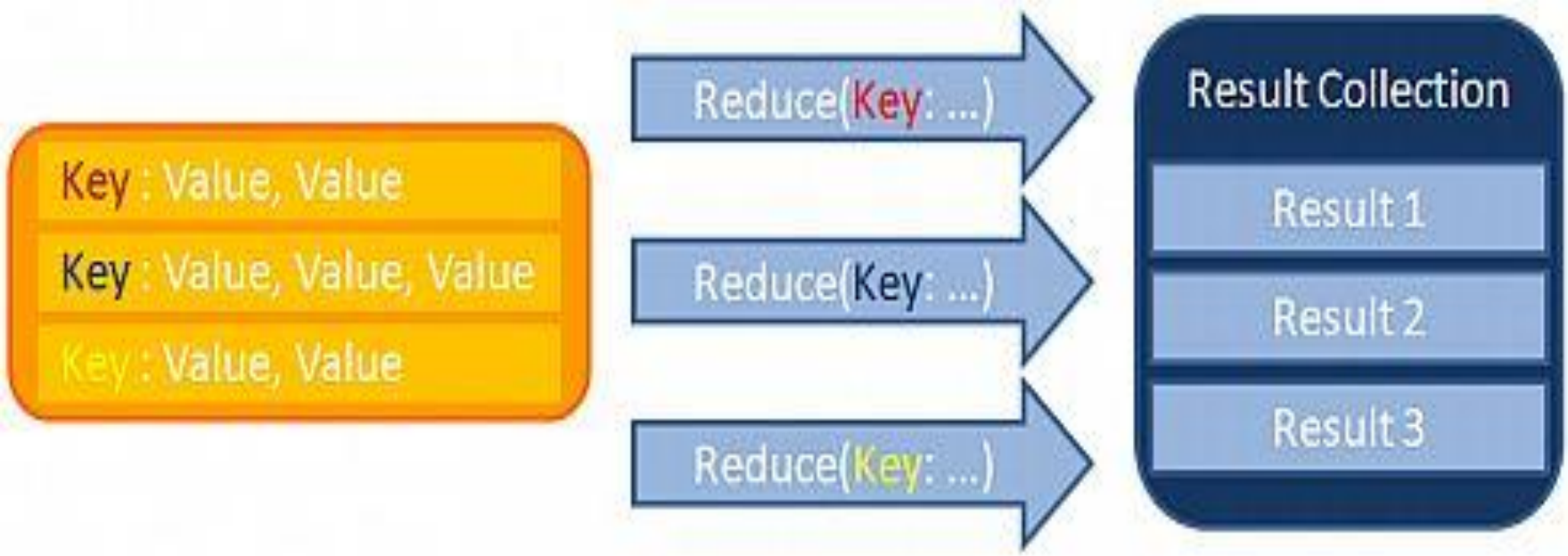
Типичная реализация подхода MapReduce

2. Сортировка всех пар `Key/Value` и создание новых экземпляров объектов, где все значения (`value`) будут сгруппированы по ключу.



Типичная реализация подхода MapReduce

3. Выполнение функции Reduce — для каждого сгруппированного экземпляра `Key/Value` объекта. Функция Reduce вернет новый экземпляр объекта, который будет включен в результирующую коллекцию.



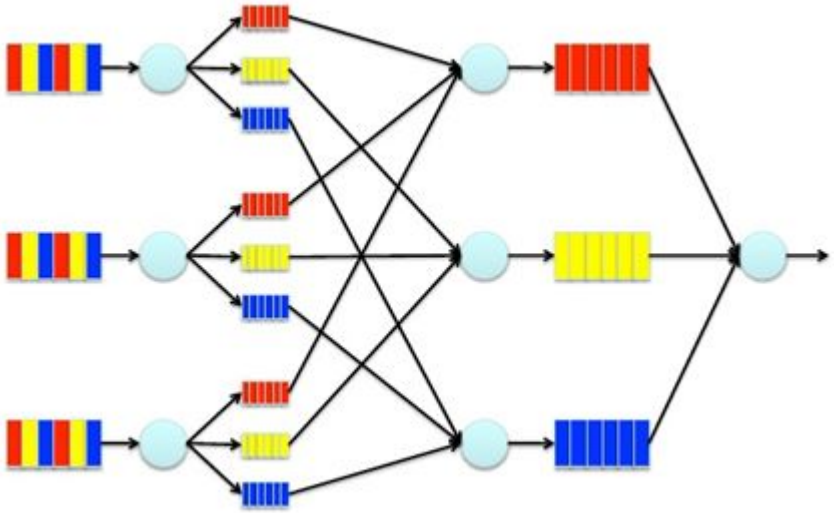
Дополнительные факты про MapReduce

1) Все запуски функции **map** работают независимо и могут работать параллельно, в том числе на разных машинах кластера.

2) Все запуски функции **reduce** работают независимо и могут работать параллельно, в том числе на разных машинах кластера.

3) Shuffle внутри себя представляет параллельную сортировку, поэтому также может работать на разных машинах кластера. **Пункты 1-3 позволяют выполнить принцип горизонтальной масштабируемости.**

4) Функция map, как правило, применяется на той же машине, на которой хранятся данные – это позволяет снизить передачу данных по сети (принцип локальности данных).



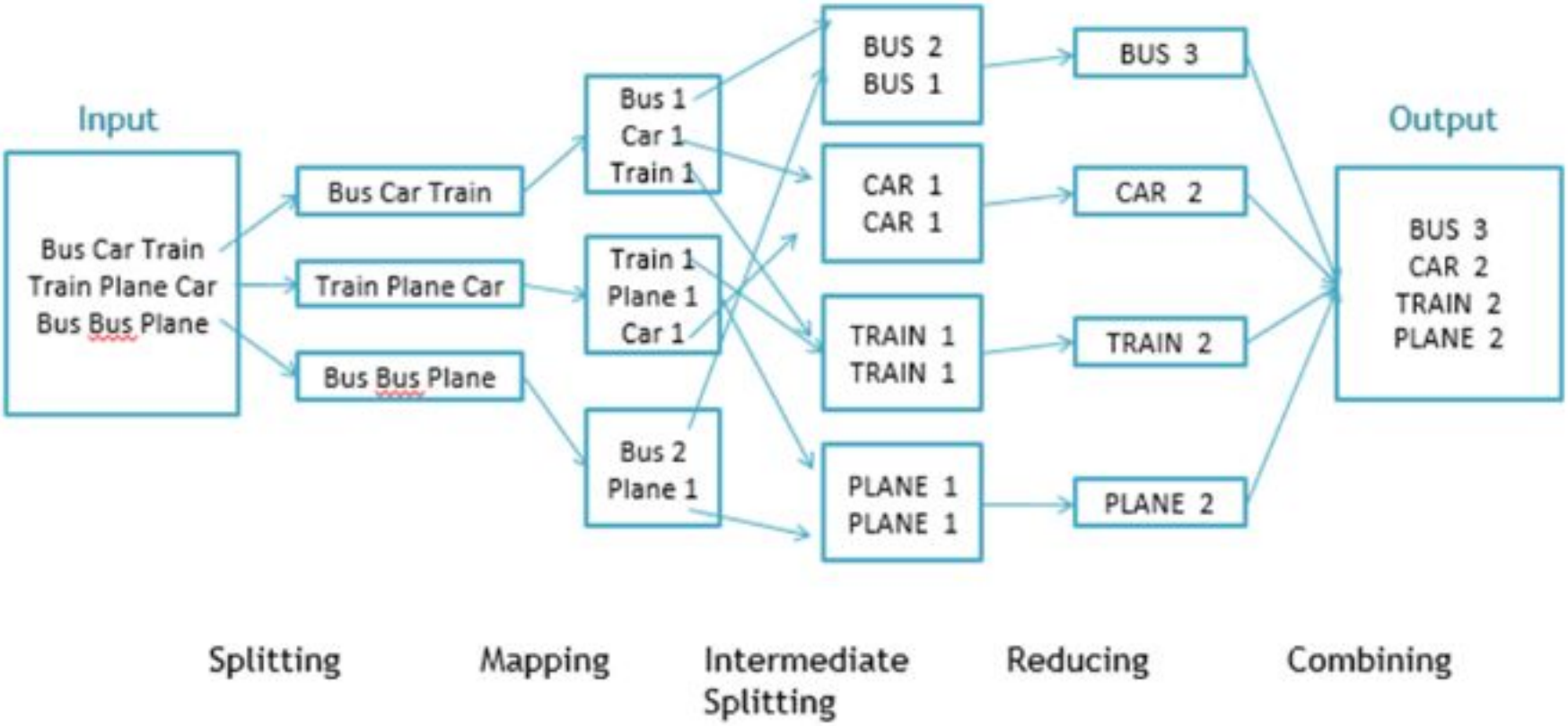
Парадигма MapReduce

Парадигма — это совокупность идей и понятий, определяющих подход к решению задач в определенной области.

Реализации:

- Google (2004) – MapReduce: Simplified Data Processing on Large Clusters
- Hadoop MapReduce (Apache Software Foundation, 2005) – это бесплатная реализация MapReduce с открытыми исходными кодами на языке Java – https://hadoop.apache.org/
- GridGain (GridGain Systems, США, 2007) — это бесплатная реализация MapReduce с открытыми исходными кодами на языке Java – https://www.gridgain.com/
- Twister Iterative MapReduce (2008) – http://www.iterativemapreduce.org/
- Qt Concurrent (2009) — это упрощенная версия фреймворка, реализованная на C++, которая используется для распределения задачи между несколькими ядрами одного компьютера – http://labs.trolltech.com/page/Projects/Threads/QtConcurrent
- CouchDB (2008) – использует MapReduce для определения представлений поверх распределенных документов – https://couchdb.apache.org/
- MongoDB (2008) – также позволяет использовать MapReduce для параллельной обработки запросов на нескольких серверах – https://www.mongodb.com/
- Qizmt (2009) — это реализация MapReduce с открытым исходным кодом

Приме
р


Input
Bus Car Train
Train Plane Car
Bus Bus Plane
Bus Car Train
Train Plane Car
Bus Bus Plane
Bus 1
Car 1
Train 1
Train 1
Plane 1
Car 1
Bus 2
Plane 1
BUS 2
BUS 1
CAR 1
CAR 1
TRAIN 1
TRAIN 1
PLANE 1
PLANE 1
BUS 3
CAR 2
TRAIN 2
PLANE 2
Output
BUS 3
CAR 2
TRAIN 2
PLANE 2
Splitting
Mapping
Intermediate Splitting
Reducing
Combining

Hadoop

Hadoop MapReduce

Hadoop –программная платформа (software framework) построения распределенных приложений для массово-параллельной обработки (massive parallel processing, MPP) данных.

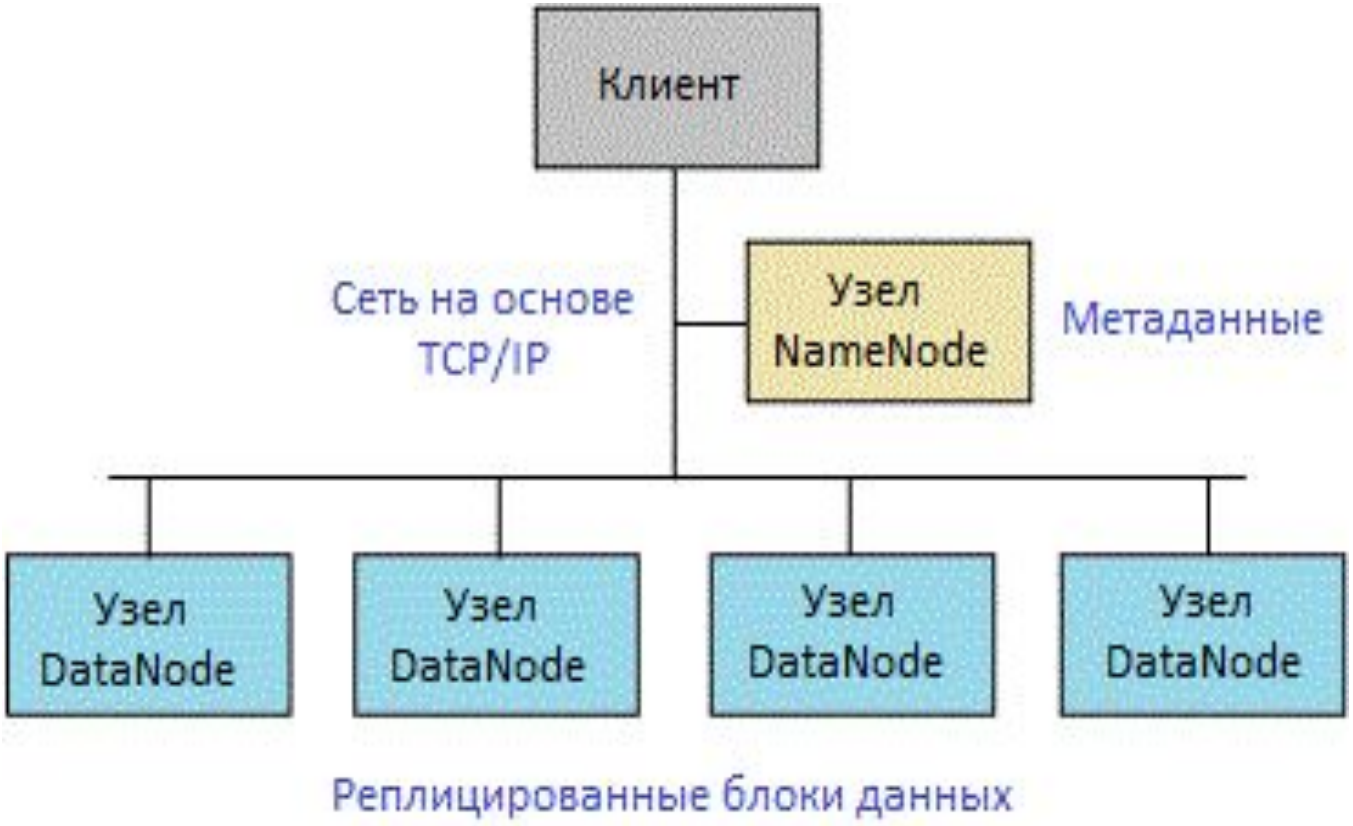
Hadoop включает в себе следующие компоненты:

- *HDFS* – распределенная файловая система;
- *Hadoop MapReduce* – программная модель (framework) выполнения распределенных вычислений для больших объемов данных в рамках парадигмы map/reduce.

Проекты, связанные с Hadoop, но не входящих в Hadoop core

- **Hive** – инструмент для SQL-like запросов над большими данными (превращает SQL-запросы в серию MapReduce–задач)
- **Pig** – язык программирования для анализа данных на высоком уровне. Одна строчка кода на этом языке может превратиться в последовательность MapReduce-задач
- **Hbase** – колоночная база данных, реализующая парадигму BigTable
- **Cassandra** – высокопроизводительная распределенная key-value база данных
- **ZooKeeper** – сервис для распределённого хранения конфигурации и синхронизации изменений этой конфигурации
- **Mahout** – библиотека и движок машинного обучения на больших данных.

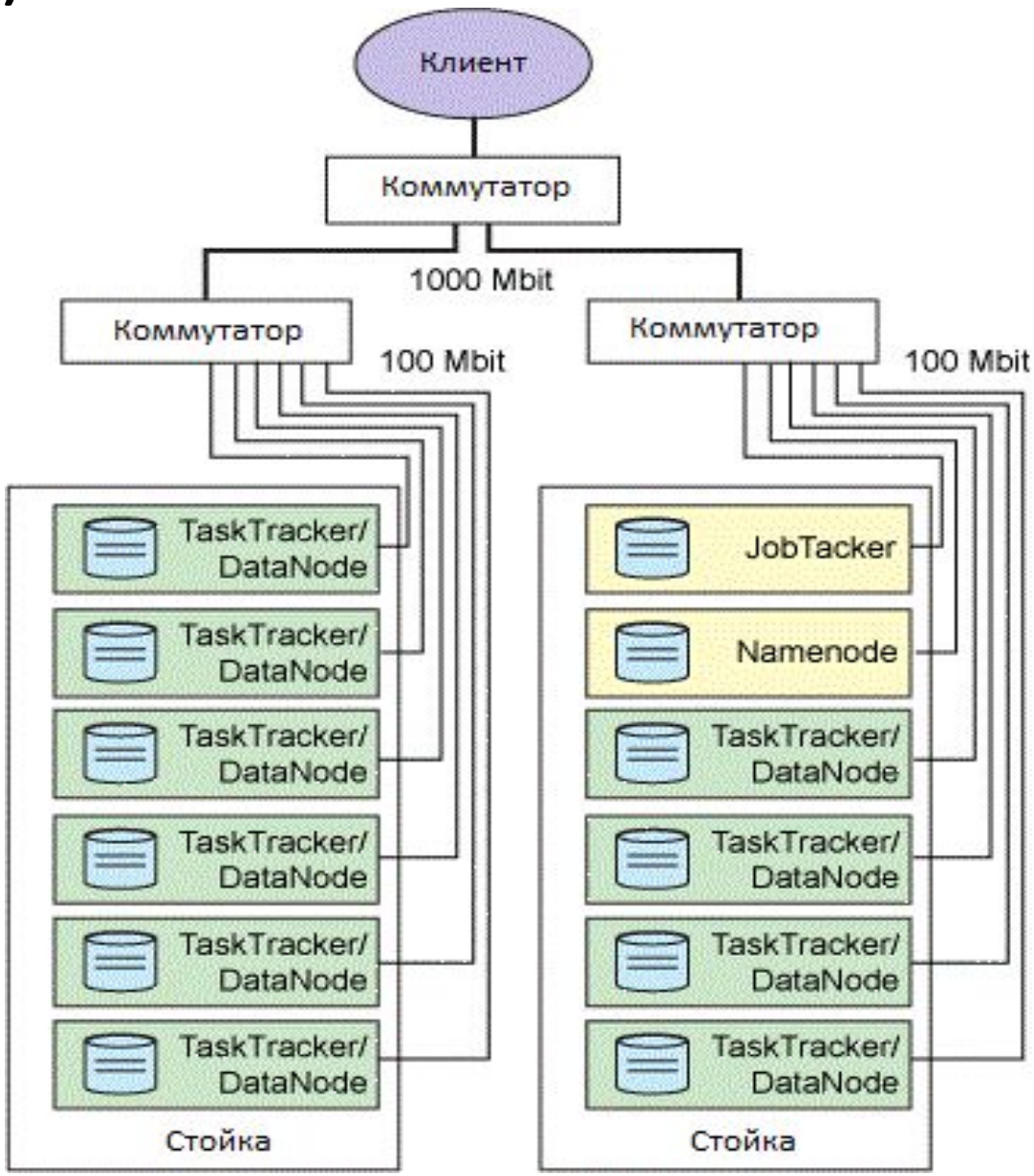
Упрощенный вид кластера Hadoop (уровень HDFS)



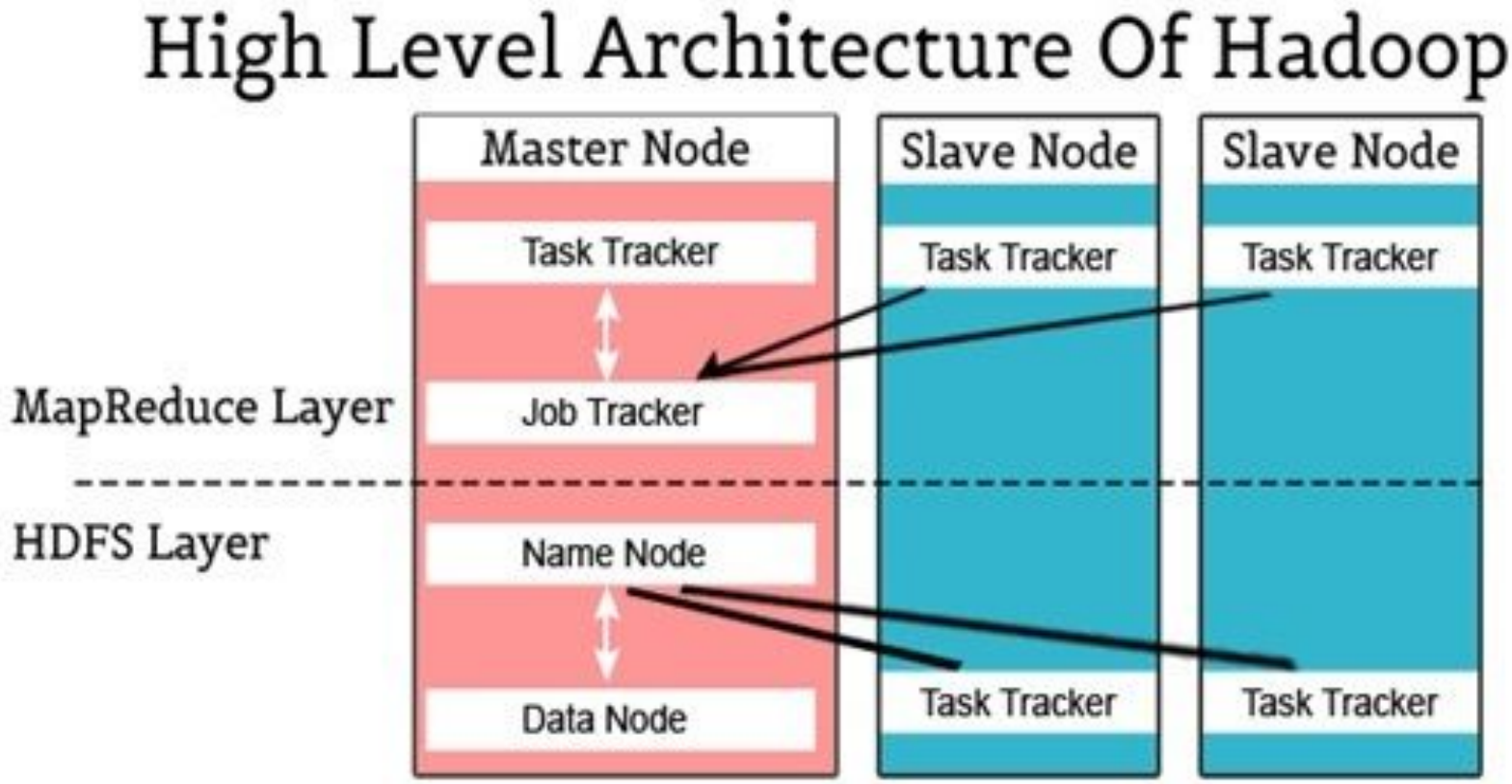
Характеристики HDFS

- ✔ hdfs лучше работает с небольшим числом больших файлов/блоков;
- ✔ один раз записали, много раз считали;
- ✔ можно только целиком считать, целиком очистить или дописать в конец (нельзя с середины);
- ✔ файлы бьются на блоки split (к примеру 64мб);
- ✔ все блоки реплицируются с фактором 3 по умолчанию (хранятся в 3 копиях на разных серверах)

Пример кластера Hadoop (уровень mapreduce)



Архитектура Hadoop



Пример WordCount

https://habr.com/ru/company/dca/blog/268277/

Суть реализации задачи WordCount

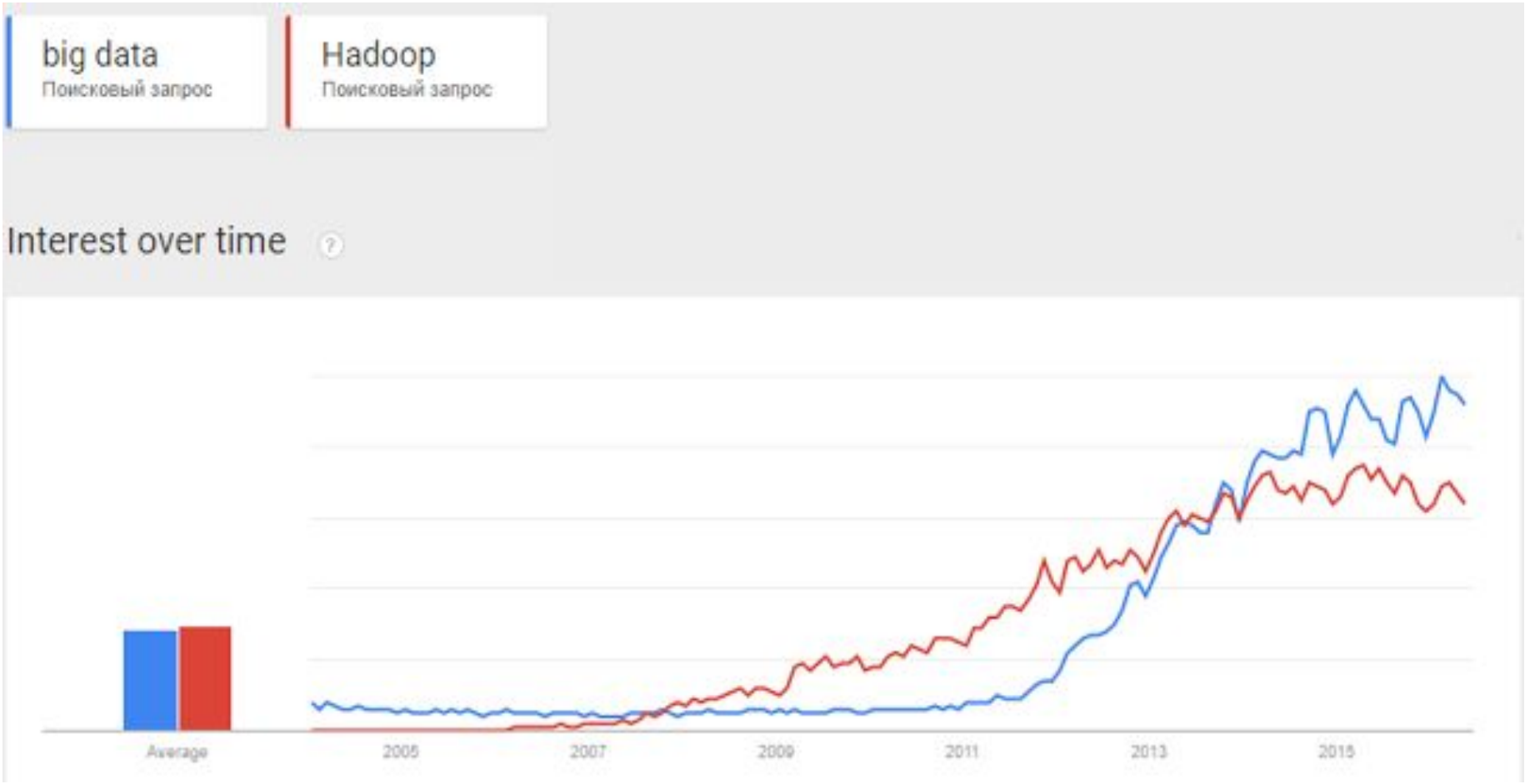
1. создать 2 класса, наследуемых от org.apache.hadoop.mapred.MapReduceBase.
   - реализация интерфейса org.apache.hadoop.mapred.Mapper (со своейmap-функцией);
   - реализация интерфейса org.apache.hadoop.mapred.Reducer (со своей reduce-функцией);
2. сконфигурировать MapReduce-задание, создав экземпляр класса org.apache.hadoop.mapred.JobConf и выставив с его помощью параметры:
   - путь к входному файлу на HDFS;
   - путь к директории, где будет лежать результат;
   - формат входных и выходных данных;
   - ваш класс с map-функцией;
   - ваш класс с reduce-функцией.
4. запустить задание на выполнение методом JobConf.runJob().

Пример WordCount

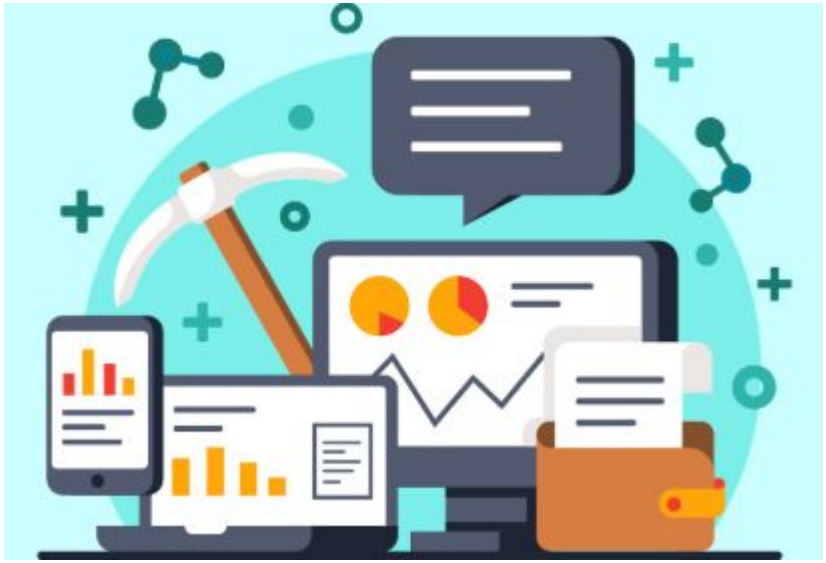
Hadoop сделает самостоятельно:

- ✔ копирование jar-файла с заданием;
- ✔ разбиение входных данных на части;
- ✔ назначение каждому рабочему узлу своей части на обработку;
- ✔ координация между узлами;
- ✔ сортировка и перетасовка промежуточных пар ключ/значение;
- ✔ перезапуск задач в случае ошибок;
- ✔ извещение клиента об окончании обработки.

Hadoop & big data



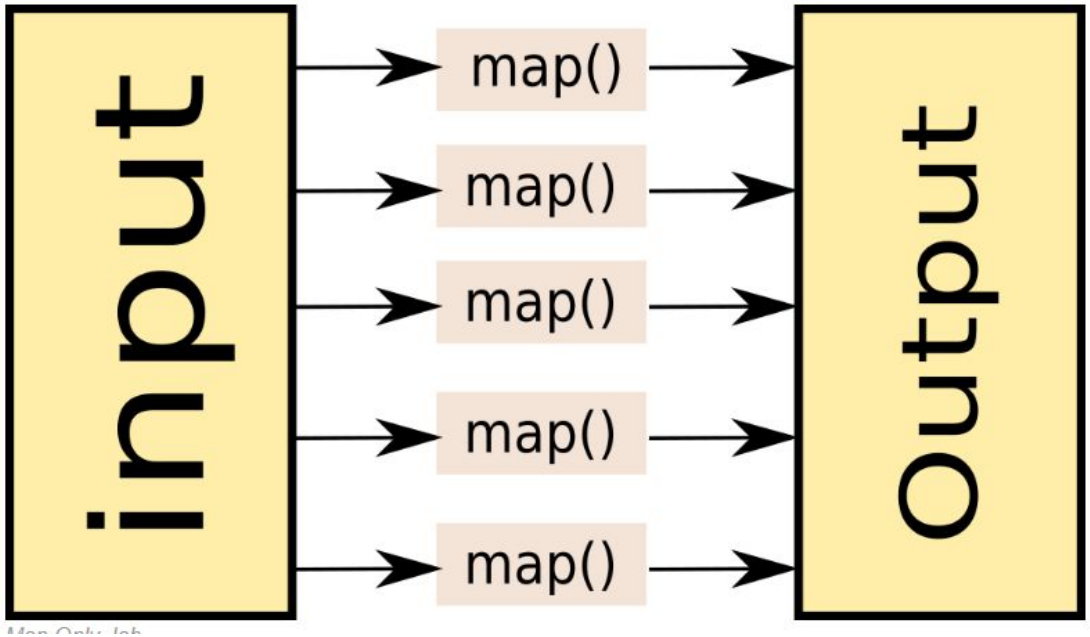
Приемы и стратегии разработки MapReduce-приложений



Map only job

Примеры задач только со стадией Map :

- Фильтрация данных (например, «Найти все записи с IP-адреса 123.123.123.123» в логах web-сервера);
- Преобразование данных («Удалить колонку в csv-логах»);
- Загрузка и выгрузка данных из внешнего источника («Вставить все записи из лога в базу данных»).



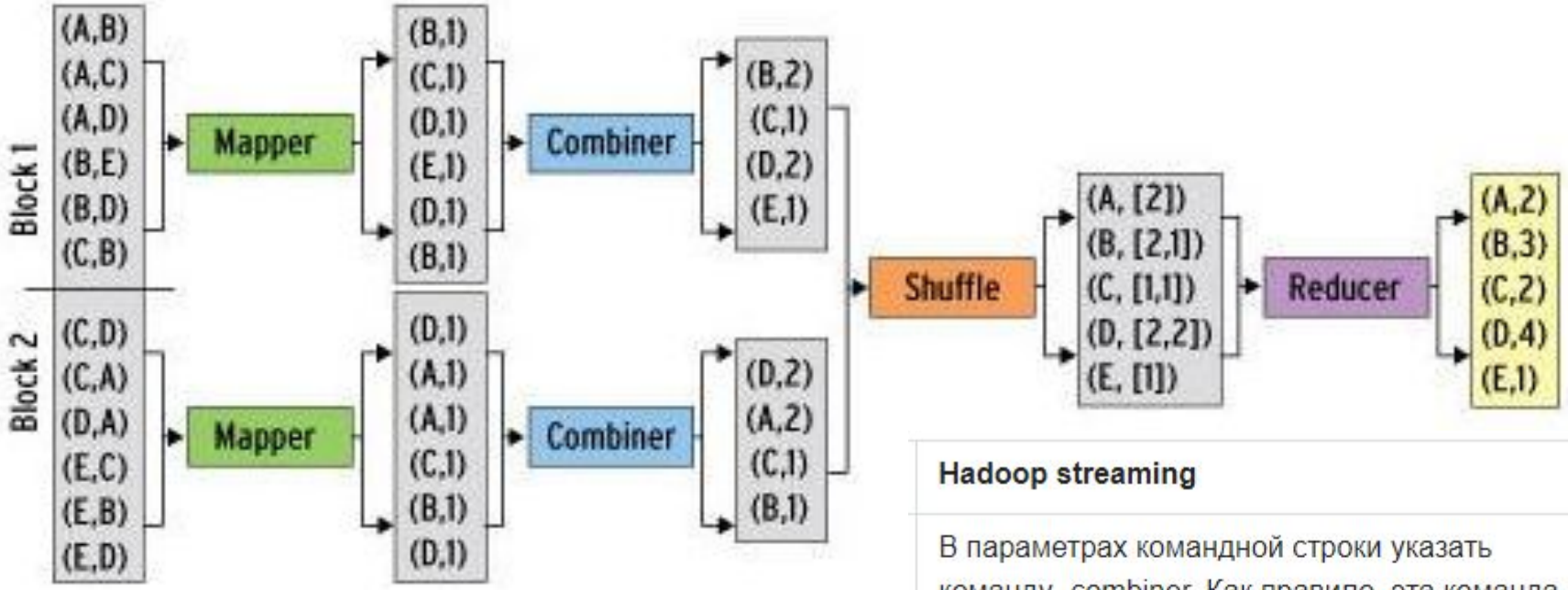
Map only job

Hadoop Streaming Interface

Не указываем редьюсер и указываем нулевое количество редьюсеров Пример:

```
hadoop jar hadoop-streaming.jar \
 -D mapred.reduce.tasks=0\
-input input_dir\
-output output_dir\
-mapper "python mapper.py"\
-file "mapper.py"
```

Combine



Hadoop streaming

В параметрах командной строки указать команду -combiner. Как правило, эта команда совпадает с командой reducer'а. Пример:

```
hadoop jar hadoop-streaming.jar \
-input input_dir\
-output output_dir\
-mapper "python mapper.py"\
-reducer "python reducer.py"\
-combiner "python reducer.py"\
-file "mapper.py"\
-file "reducer.py"\
```

Цепочки MapReduce-задач

Задача: имеется набор текстовых документов, необходимо посчитать, сколько слов встретилось от 1 до 1000 раз в наборе, сколько слов от 1001 до 2000, сколько от 2001 до 3000 и так далее

Решение на псевдокоде:

```
#map1
def map(doc):
        for word in doc:
                yield word, 1
```

```
#reduce1
def reduce(word, values):
        yield int(sum(values)/1000), 1
```

```
#map2
def map(doc):
        interval, cnt = doc.split()
        yield interval, cnt
```

```
#reduce2
def reduce(interval, values):
        yield interval*1000, sum(value
s)
```

https://habr.com/ru/company/dca/blog/270453/

Map-Reduce на примере MongoDB

Входные коллекции:

```
{
        name : "John",
        age : 23,
        interests : ["football", "IT", "women"]
}
```

Выходная коллекция:

```
{
        key: "football",
        value: 1349
},
{
        key: "MongoDB",
        value: 58
},
//...
```

Map-Reduce на примере MongoDB

Функцию **map:**

```
function map(){
        for(var i in this.interests) {
                emit(this.interests[i], 1);
        }
}
```

Функция **reduce:**

```
function reduce(key, values) {
        var sum = 0;
        for(var i in values) {
                sum += values[i];
        }
        return sum;
}
```

Map-Reduce на примере MongoDB

Запуск:

```
db.users.mapReduce(map, reduce,{out:"interests"})
```

Требования на реализацию функции reduce:

- Тип возвращаемого значения функции **reduce** должен совпадать с типом значения, которое выдается функцией **map** (второй параметр функции **emit**)
- Должно выполняться равенство: **reduce(key, [A, reduce(key, [B, C])]) == reduce(key, [A, B, C])**
- Повторное применение операции **Reduce** к полученной паре *<ключ, значение>* не должно влиять на результат (идемпотентность)
- Порядок значений, передаваемых функции **reduce**, не должен влиять на результат

Пример 2. Map-Reduce на примере MongoDB

Задача: Найти среднее количество интересов у людей разных возрастов

```
function map(){
        emit(this.age, {interests_count: this.interests.length, count: 1});
}
```

Пример 2. Map-Reduce на примере MongoDB

```
function reduce(key, values) {
        var sum = 0;
        var count = 0;
        for(var i in values){
                count += values[i].count;
                sum += values[i].interests_count;
        }
        return {interests_count: sum, count: count};
}
```

Пример 2. Map-Reduce на примере MongoDB

```
function finalize(key, reducedValue) {
        return reducedValue.interests_count / reducedValue.count;
}
```

Пример 2. Map-Reduce на примере MongoDB

Команда для вызова:

```
db.users.mapReduce(map, reduce, {finalize: finalize, out:"interests_by_age"})
```

Немного дегтя

Инструмент:	Объект:	Комментарии:
		
		
		
		

Немного дегтя

Инструмент:	Характеристики:	Комментарии:
	Имеет встроенный прицел	Теперь его можно метать
	Имеет встроенный нож	Это позволит более эффективно рубить деревья

Немного дегтя

Инструмент:	Характеристики:	Комментарии:
	Имеет встроенную косу	Теперь им можно косить траву

ЗАЧЕМ ?

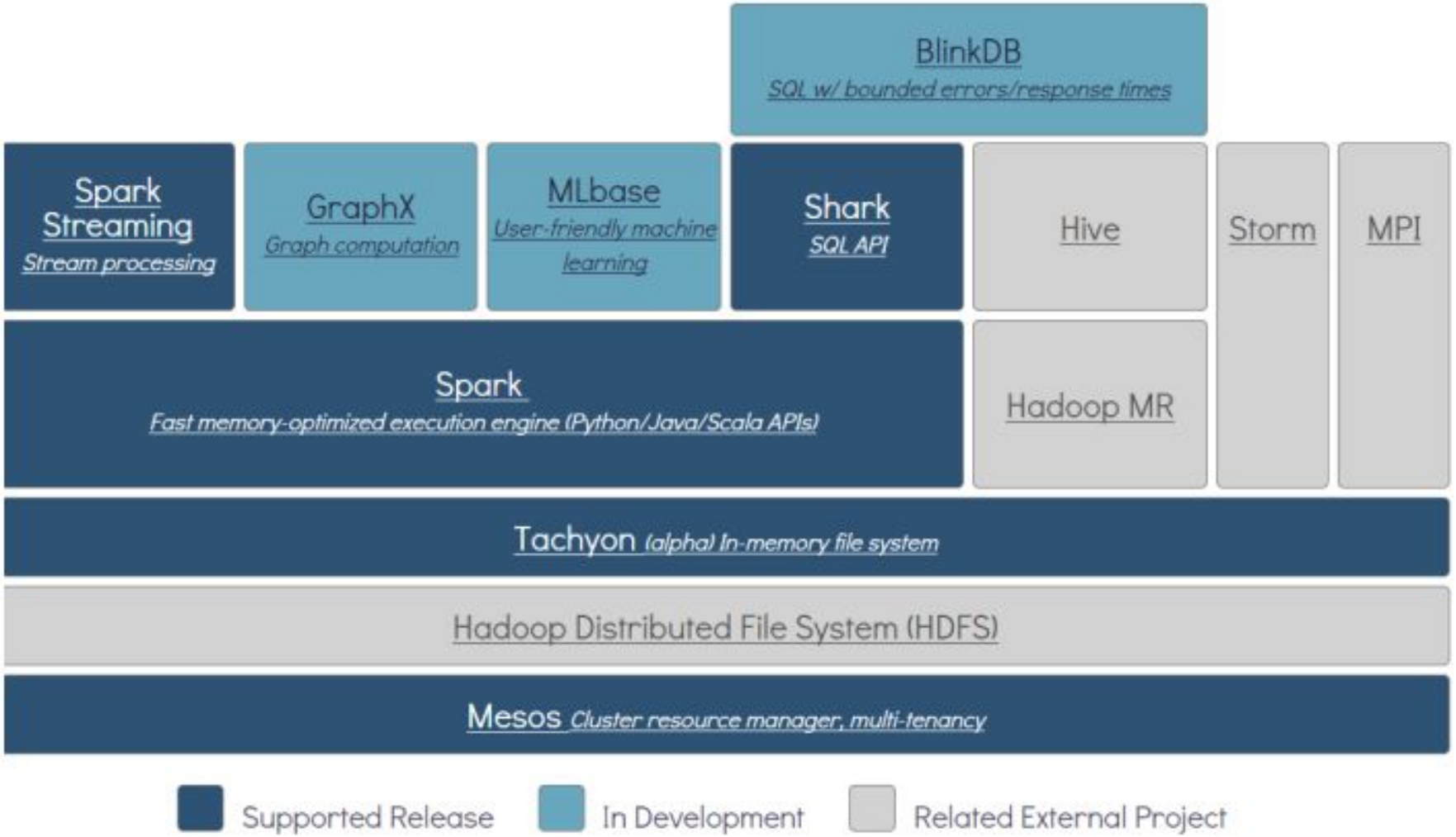
Что не так с MapReduce

- Данные должны уже «быть» до старта
- Пакетная обработка
- Связь между задачами через HDFS
- Не поддерживаются рекурсивные и итеративные задачи
- Долгая планировка задачи

Что есть Apache Spark

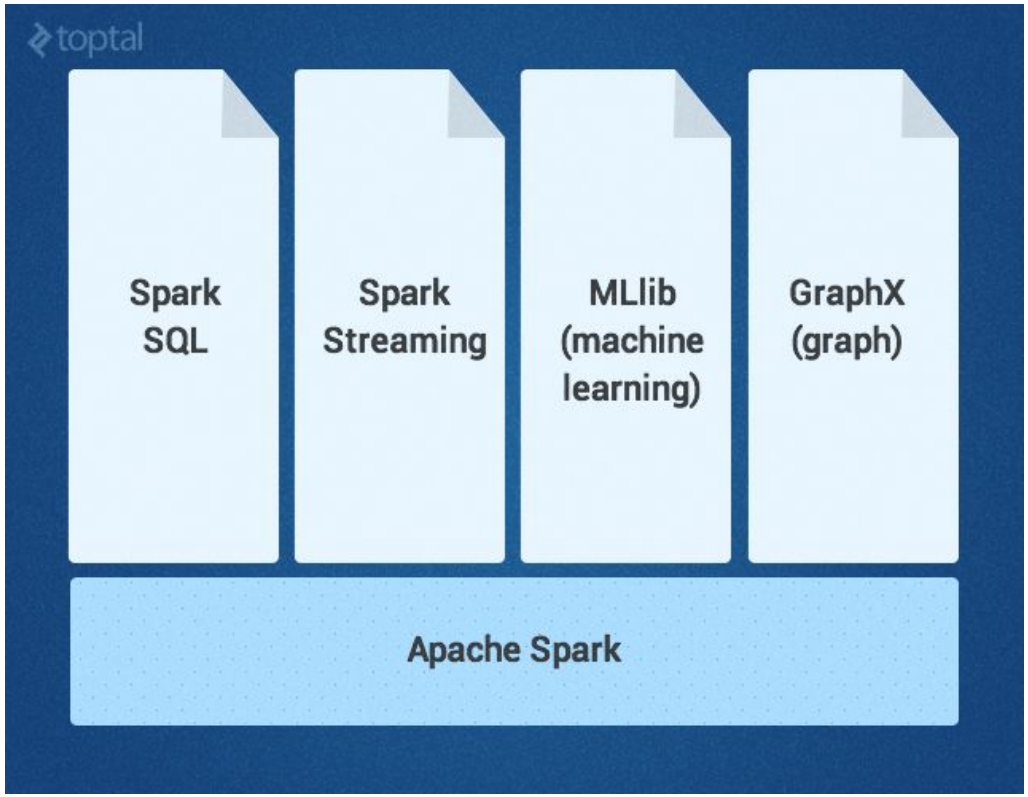
- Быстрая и универсальная система кластерных вычислений
- + набор стандартных расширений
  - Spark SQL (обработка SQL и структурированных данных)
  - MLlib (машинное обучение)
  - GraphX (обработка графов)
  - Spark Streaming (потоковая обработка)
- Высокоуровневое API для Java, Scala, Python

Spark & Hadoop



Ядро Spark

Ядро Spark – это базовый движок для крупномасштабной параллельной и распределенной обработки данных.



Ядро отвечает за:

- управление памятью и восстановление после отказов
- планирование, распределение и отслеживание заданий кластере
- взаимодействие с системами хранения данных

RDD (Resilient Distributed Dataset)

RDD (устойчивый распределенный набор данных) – неизменяемая отказоустойчивая распределенная коллекция объектов, которые можно обрабатывать параллельно.

В RDD поддерживаются операции двух типов:

- Трансформации – это операции, совершаемые над RDD; результатом трансформации становится новый RDD, содержащий ее результат.

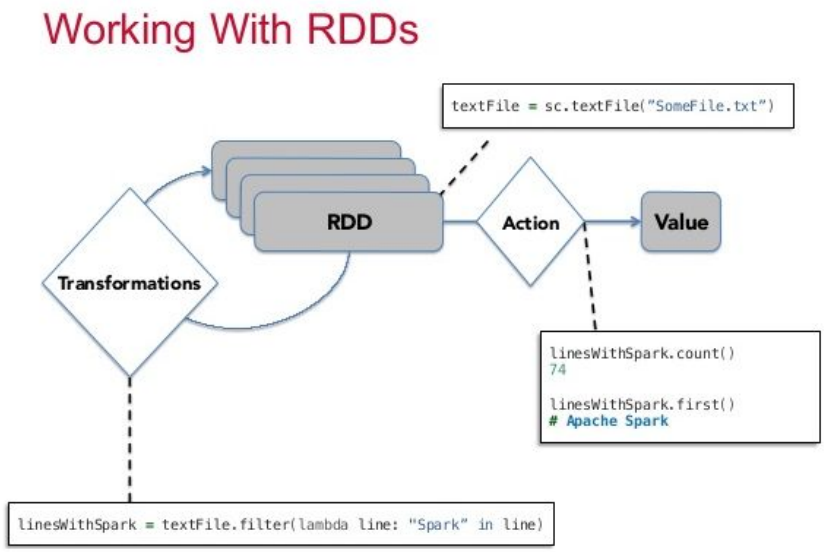
- Действия – это операции, возвращающие значение, получаемое в результате некоторых вычислений в RDD.

Распространенные **преобразования**:

- **.map(function)** — применяет функцию function к каждому элементу RDD;
- **.filter(function)** — возвращает все элементы RDD, на которых функция function вернула истинное значение
- **.distinct([numTasks])** — возвращает RDD, который содержит уникальные элементы исходного RDD;
- **.union(otherDataset); .intersection(otherDataset);**
- **.cartesian(otherDataset)** ;

Примеры **действий**:

- **.saveAsTextFile(path)**;
- **.collect()** — возвращает элементы RDD в виде массива; **.take(n)** — возвращает в виде массива первые n элементов RDD;
- **.count()** — возвращает количество элементов в RDD;
- **.reduce(function)**



https://habr.com/ru/company/mlclass/blog/250811/

Spark. Примеры

Пример 1. 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")
```

Пример 2. Просмотр первых 10 элементов:

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

Пример 3. Трансформация: поиск максимального и минимального элементов RDD.

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

Combine

Команда для вызова:

Combine

Команда для вызова:

Combine

Команда для вызова: