Introduction to Spark

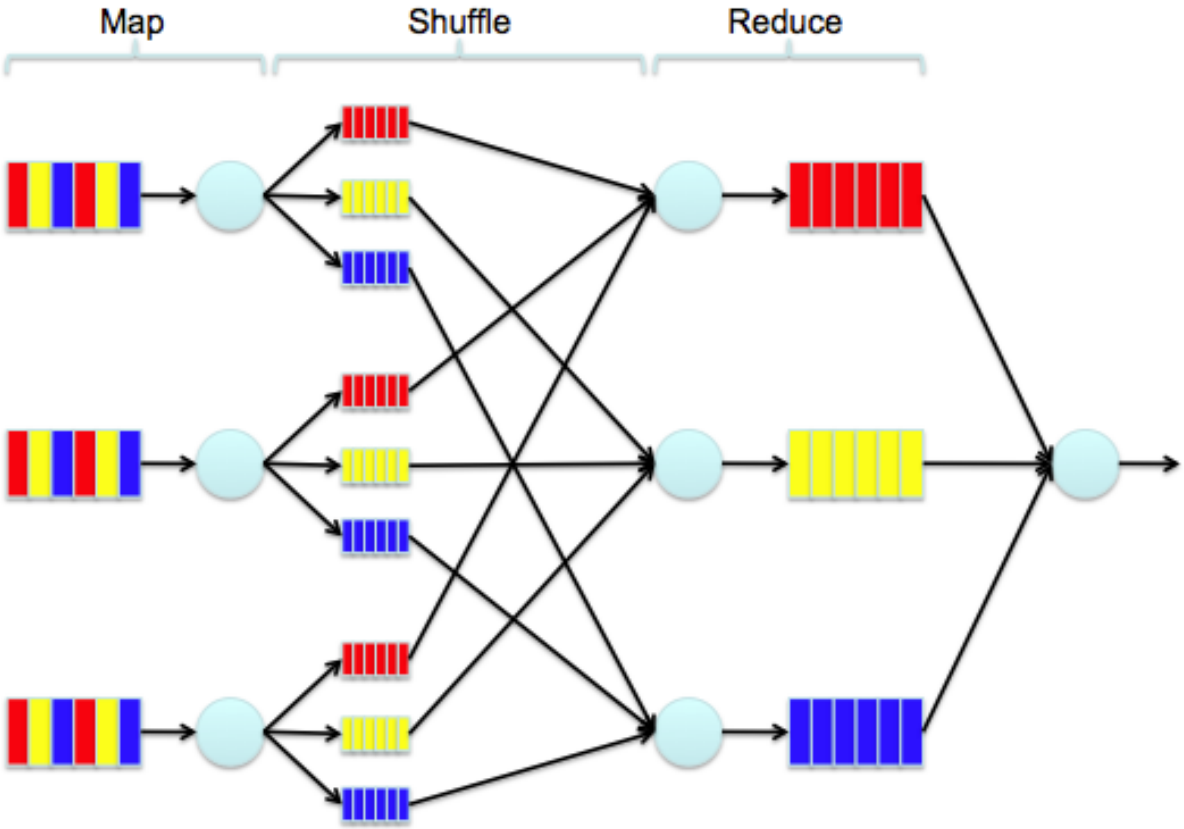
Andrey Ustyuzhanin, andrey.u@gmail.com
Maxim Borisyak, maxim.borisyak@gmail.com

Big Data

- Данные не помещаются на одну машину
- Распределенные вычисления и хранение как требование
- MPI (?), **Hadoop, Spark**

Map Reduce

- Map, Shuffle, Reduce
- `Map(f)` применяет `f` ко всем элементам коллекции
- `Reduce(g)` вычисляет свертку **ассоциативного** (**коммутативного**) оператора `g` над коллекцией



Hadoop Map Reduce

Немного FP

```
trait Collection[A] {
 def map[B](f: A => B): Collection[B]
 def flatMap[B](f: A => Collection[B]): Collection[B]
 def filter(f: A => Boolean): Collection[A]

 def foldLeft[B](zero: B)(f: (B, A) => B): B
 def foldRight[B](zero: B)(f: (A, B) => B): B
 def reduce[B](zero: B)(f: (A, B) => B): B
 def reduce(f: (A, A) => A): A
}
```

см. секцию A little bit of FP

- Распределенные вычисления
- Работает поверх Hadoop, Mesos, …
- Высокая скорость
- Функциональная парадигма
- Поддерживает Java, **Scala, Python**

Подсчет слов

```
file = spark.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Логистическая регрессия

```
points = spark.textFile(...).map(parsePoint).cache()
w = numpy.random.ranf(size = D) # current separating plane
for i in range(ITERATIONS):
    gradient = points.map (lambda p:\
      (1 / (1 + exp( -p.y * (w.dot(p.x))) ) - 1) * p.y * p.x
    ).reduce(lambda a, b: a + b)
    w -= gradient
print "Final separating plane: %s" % w
```

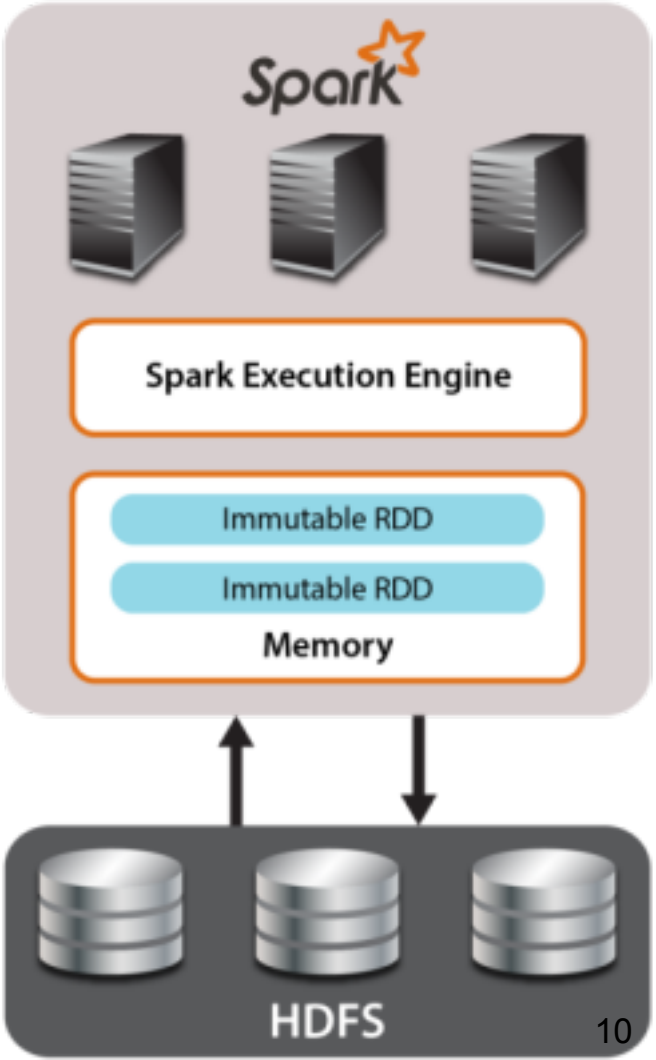
Логистическая регрессия

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.random(D) // current separating plane
for (i <- 1 to ITERATIONS) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  }.reduce { _ + _ }
  w -= gradient
}
println("Final separating plane: " + w)
```

Resilient Distributed Datasets

Базовая абстракция Spark:

- распределенная коллекция
- иммутабельна
- устойчива к отказам
- кэширование



Spark модель исполнения

- Два типа преобразований RDD:
  - action: RDD => значение
  - transformation: RDD => RDD
- Transformation - ленивы, вычисляются только при вызове action

Spark модель исполнения

- Driver / worker
- Driver формирует программу Spark в виде DAG преобразований RDD
- При вызове action вычисляются необходимые транформации.

Создание и сохранение RDD

```
trait SparkContext {
 def textFile(path: String): RDD[String]
 def objectFile[T](path: String): RDD[T]
 def parallelize[T](seq: Seq[T]): RDD[T]
 def union[T](rdds: Seq[RDD[T]]): RDD[T]
}
trait RDD[T] {
 def saveAsTextFile(path: String): Unit
 def saveAsObjectFile(path: String): Unit
}
```

см. секцию RDD creation

Создание и сохранение RDD

Python:

```
text = spark.textFile("hdfs://...")

nums = spark.parallelize(xrange(0, N))
```

Scala:

```
val text: RDD[String] = spark.textFile("hdfs://...")

val count: RDD[Int] = spark.parallelize(1 to N)
```

см. секцию RDD creation

Основные операции над RDD

Полный список см. в Spark API.

```
trait RDD[T] {
 def map[U](f: T => U): RDD[U]
 def flatMap[U](f: T => Seq[U]): RDD[U]
 def filter(p: T => Boolean): RDD[T]

 def aggregate[U](zero: U)(f: (U, T) => U, g: (U, U) => U): U
 def fold(zero: T)(f: (T, T) => T): T
 def reduce(op: (T, T) => T): T
}
```

см. секцию Simple RDD operations

Основные операции над RDD

```
trait RDD[T] {
 def count(): Long
 def max(): T
 def min(): T

 def sample(fraction: Double): RDD[T]
 def take(n: Int): Array[T]
 def collect(): Array[T]
}
```

см. секцию Simple RDD operations

Операции над PairRDD

```
trait PairRDD[K, V] extends RDD[(K, V)] {
 def aggregateByKey[U](zero: U)
                      (f: (U, V) => U, g: (U, U) => U): RDD[(K, U)]

 def foldByKey(zero: V)(f: (V, V) => V): RDD[(K, V)]
 def reduceByKey(op: (V, V) => V): RDD[(K, V)]
 def groupByKey(): RDD[(K, Iterable[V])]
}
```

см. секцию Pair RDD operations

Операции над RDD

```
trait PairRDD[K, V] extends RDD[(K, V)] {
 def cogroup[U](other: RDD[(K, U)]): RDD[(K, (Iterable[V], Iterable[U]))]

 def fullOuterJoin[U](other: RDD[(K, U)]): RDD[(K, (Option[V], Option[U]))]

 def leftOuterJoin[U](other: RDD[(K, U)]): RDD[(K, (V, Option[U]))]

 def join[U](other: RDD[(K, U)]): RDD[(K, (V, U))]
}
```

см. секцию RDD to RDD operations

Примеры

- Word count
- Zipf’s law
- Heap’s law

Machine Learning в Spark (MLlib)

- Базовые алгоритмы ML
  - распределенные по построению
  - вычислительно эффективные
  - production ready
- Активно развивается

Machine Learning в Spark (MLlib)

- Классификация/регрессия
  - SVM
  - decision trees, GBT, random forest
  - linear least squares, ridge regression, Lasso, logistic regression
- Кластеризация
  - k-means
- Снижение размерности
  - PCA via SVD

см. секцию MLlib

Примеры

- Титаник
- Logistic regression
- Gradient Boosted Trees

Практическая работа

- login.miptcloud.com:55010
- passwd: lambda03
- Получить копию IPython тетрадок:
  - Introduction to Spark
  - терадку с заданиями
- Ознакомиться с Introduction to Spark.
- Выполнить задания

Logistic regression