Яндекс

Apache Spark

Егор Пахомов

Я.Субботник в Минске, 30.08.2014

О себе

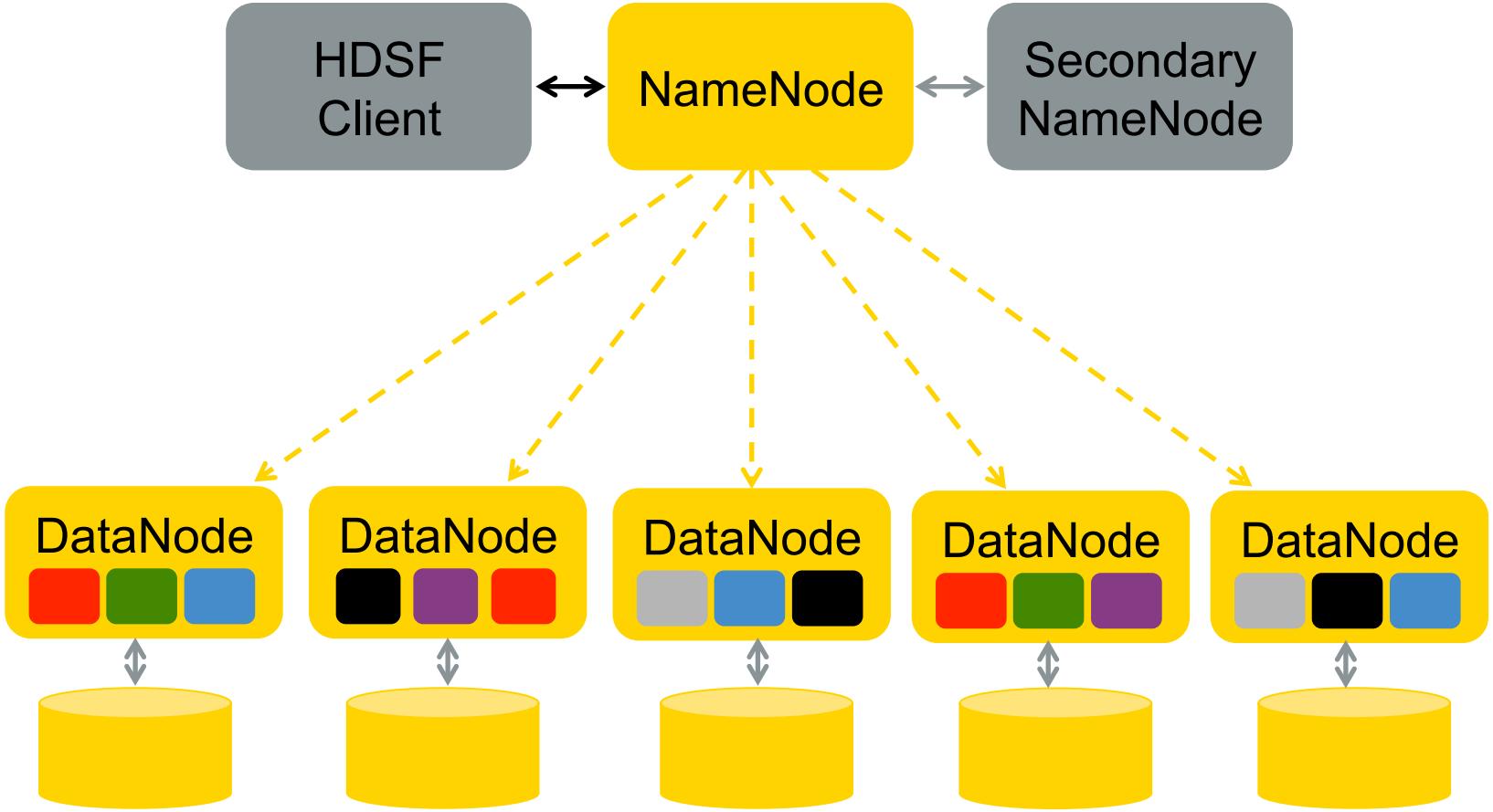
- Участвовал в создании Apache Solr based search engine для крупного западного e-commerce
- Разрабатывал Яндекс.Острова последний год
- Spark Contributor

Сложности больших данных

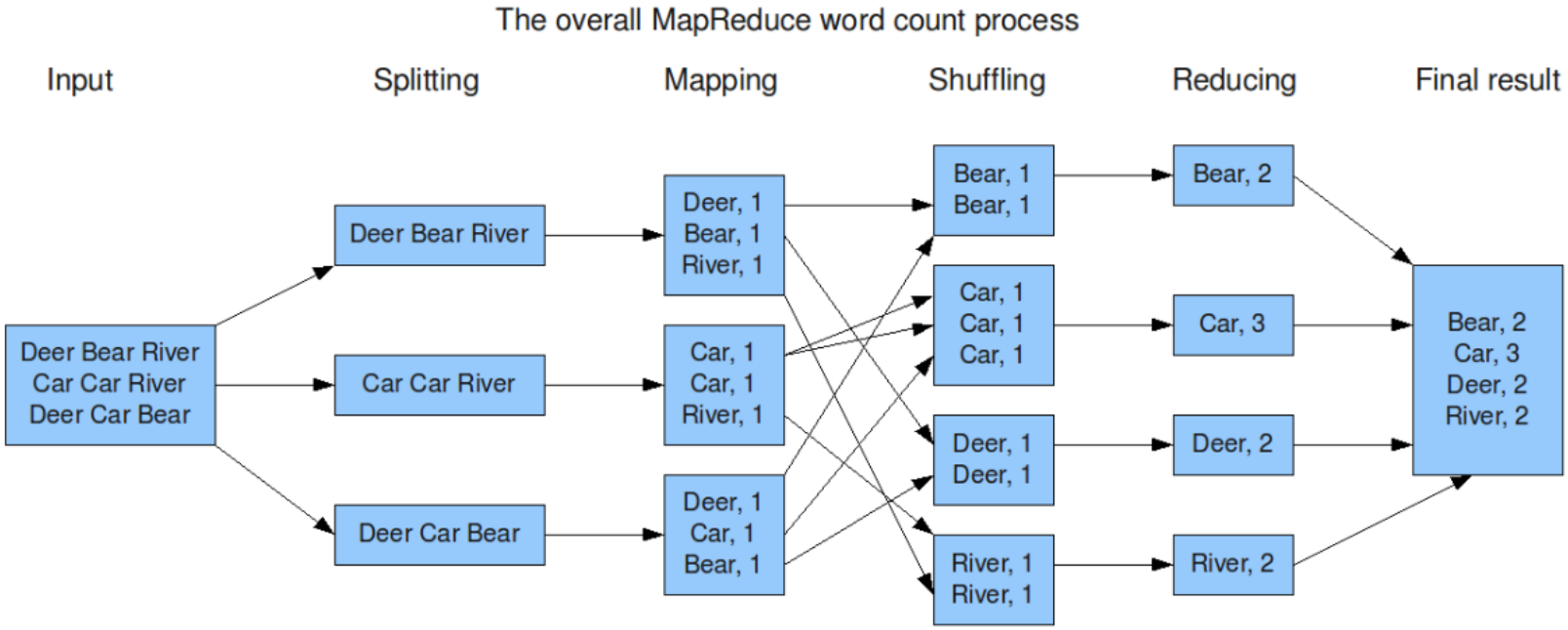
- Файлы слишком большие, чтобы разместить на одной машине
- Суперкомпьютеры дорогие и трубуют специальной экспертизы
- В случае поломки одной машины → перепрогонять весь алгоритм

Hadoop придет на помощь!

HDFS как хранилище файлов



Модель map-reduce



```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException
        {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens())
            {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
    {
        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException
        {
            int sum = 0;
            for (IntWritable val : values)
            {
                sum += val.get();
            }
context.write(key, new IntWritable(sum));
        }
    }
```

Что сделал Hadoop

- Fault tolerance
- Data locality
- Новая вычислительная модель

Проблемы

- Тяжело реализовать алгоритм в map-reduce моделе
- Очень простой алгоритм требует как минимум 3 класса
- Частая запись на диск

RDD

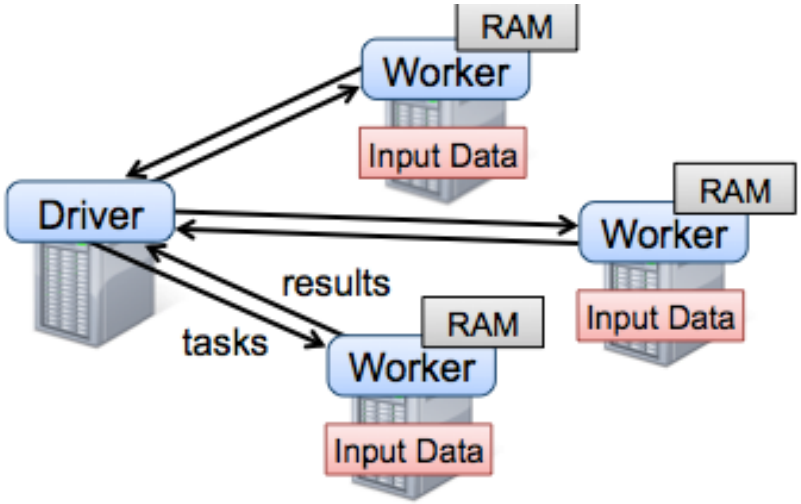


Figure 2: Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin,
Scott Shenker, Ion Stoica. NSDI 2012. April 2012.

Word count

```
val wordCounts = spark
                     .textFile("/user/usr/in")
                     .flatMap(line => line.split(" "))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
```

Whole program comparison

```
public class HadoopAnalyzer
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
        {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens())
            {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
    {
        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
        {
            int sum = 0;
            for (IntWritable val : values)
            {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception
    {
        JobConf conf = new JobConf(HadoopAnalyzer.class);
        conf.setJobName("wordcount");
        System.out.println("Executing job.");
        Job job = new Job(conf, "job");
        job.setInputFormatClass(InputFormat.class);
        job.setOutputFormatClass(OutputFormat.class);
        job.setJarByClass(HadoopAnalyzer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        TextInputFormat.addInputPath(job, new Path("/user/usr/in"));
        TextOutputFormat.setOutputPath(job, new Path("/user/usr/out"));
        job.setMapperClass(Mapper.class);
        job.setReducerClass(Reducer.class);
        job.waitForCompletion(true);
        System.out.println("Done.");
    }
```

Whole program comparison

```
package ru.yandex.spark.examples

import org.apache.spark.{SparkConf, SparkContext}
object WordCount {

  def main(args: Array[String]) {

    val conf = new SparkConf()
    conf.setAppName("serp-api")
    conf.setMaster("yarn-client")
    val spark = new SparkContext(conf)

    spark.addJar(SparkContext.jarOfClass(this.getClass).get)

    val wordCounts = spark
                       .textFile("/user/usr/in")
                       .flatMap(line => line.split(" "))
                       .map(word => (word, 1))
                       .reduceByKey(_ + _)

    wordCounts.saveAsTextFile("/user/usr/out")
    spark.stop()

  }

}
```

Actions

```
def collect(): Array[T]
def saveAsTextFile(path: String)
def toLocalIterator: Iterator[T]
```

Transformations

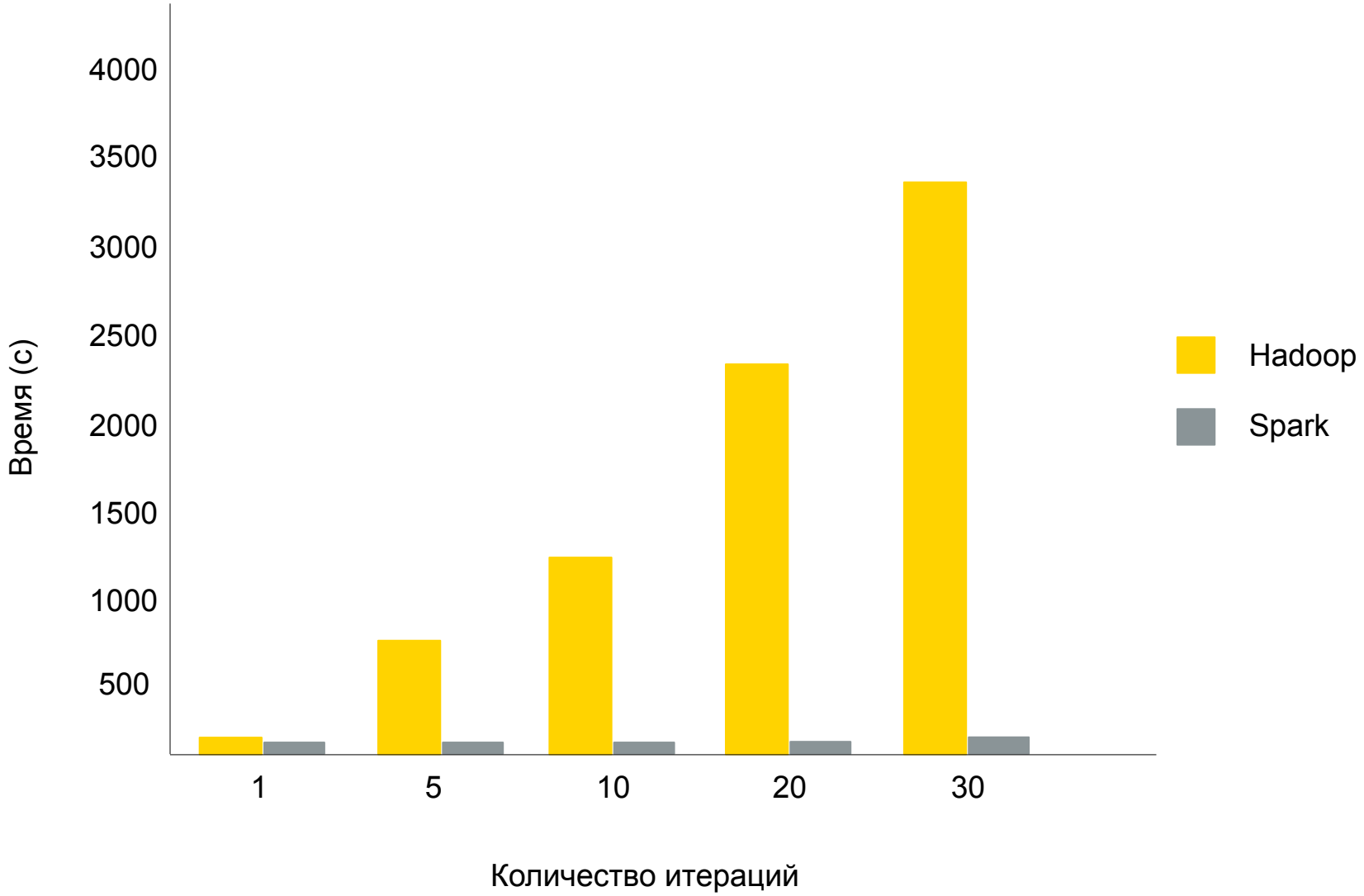
In one partition(narrow)

```
def filter(f: T => Boolean): RDD[T]
def map[U: ClassTag](f: T => U): RDD[U]
def foreachPartition(f: Iterator[T] => Unit)
def zipWithIndex(): RDD[(T, Long)]
```

Cross-partition(wide)

```
def join[W](other: RDD[(K, W)]): RDD[(K, (V, W))]
def reduceByKey(func: (V,V) => V,numPartitions: Int):RDD[(K, V)]
```

Скорость


4000
3500
3000
2500
2000
1500
1000
500
Время (с)
1
5
10
20
30
Количество итераций
Hadoop
Spark

Языки

Python

```
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://...")
```

Scala

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Языки

Java-8

```
JavaRDD<String> lines = sc.textFile("hdfs://log.txt");
JavaRDD<String> words =
    lines.flatMap(line -> Arrays.asList(line.split(" ")));
JavaPairRDD<String, Integer> counts =
    words.mapToPair(w -> new Tuple2<String, Integer>(w, 1))
         .reduceByKey((x, y) -> x + y);
counts.saveAsTextFile("hdfs://counts.txt");
```

Языки

Java-7

```
JavaRDD<String> file = spark.textFile("hdfs://...");
JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, String>() {
  public Iterable<String> call(String s) { return Arrays.asList(s.split(" "));  }
});
JavaPairRDD<String, Integer> pairs = words.map(new PairFunction<String, String, Integer>() {
  public Tuple2<String, Integer> call(String s) { return new Tuple2<String, Integer>(s, 1); }
});
JavaPairRDD<String, Integer> counts = pairs.reduceByKey(new Function2<Integer, Integer>() {
  public Integer call(Integer a, Integer b) { return a + b; }
});
counts.saveAsTextFile("hdfs://...");
```

SQL

Hadoop

Spark

Spark SQL


python
Scala
Java
HIVE QL
SQL-92
Spark SQL
SchemaRDD
Parquet
{JSON}
HIVE
cassandra

Hive начинает использовать Spark в качестве execution engine

Машинное обучение

Hadoop

Spark

SparkML

~ 24 algorithms

~ 13 algorithms

Легко писать

```
import org.apache.spark.mllib.clustering.KMeans
import org.apache.spark.mllib.linalg.Vectors

// Load and parse the data
val data = sc.textFile("data/kmeans_data.txt")
val parsedData = data.map(s => Vectors.dense(s.split(' ').map(_.toDouble)))

// Cluster the data into two classes using KMeans
val numClusters = 2
val numIterations = 20
val clusters = KMeans.train(parsedData, numClusters, numIterations)

// Evaluate clustering by computing Within Set Sum of Squared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)
```

Алгоритмы

- Classification and regression
  - linear support vector machine (SVM)
  - logistic regression
  - linear least squares, Lasso, and ridge regression
  - decision tree
  - naive Bayes
- Collaborative filtering
  - alternating least squares (ALS)

Алгоритмы

- Clustering
  - k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Optimization
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

И Mahout начинает использовать Spark

Graph

Hadoop

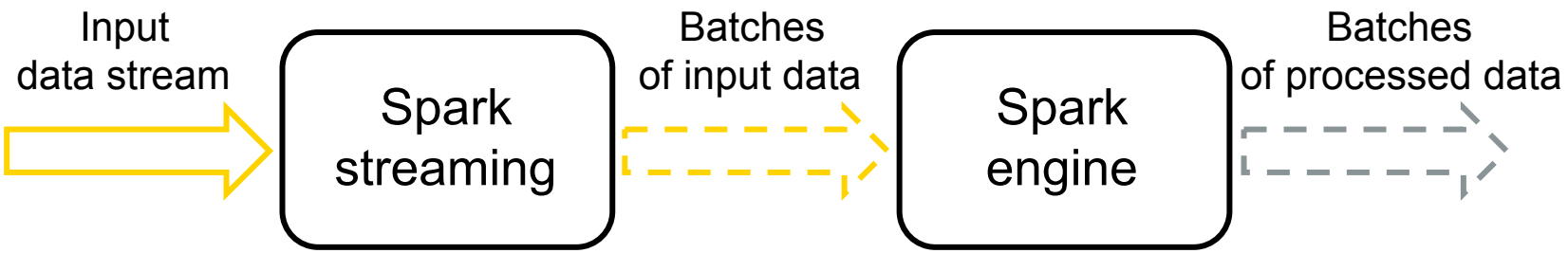
Spark

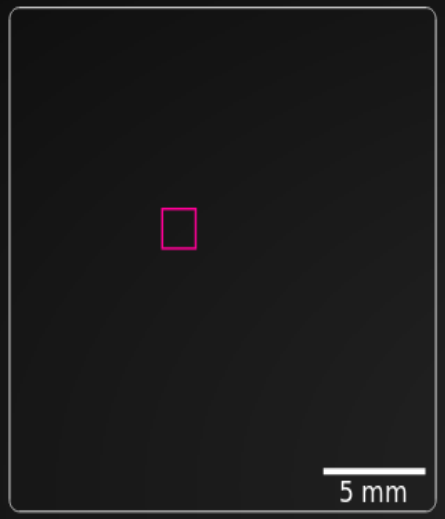
Streaming batch processing

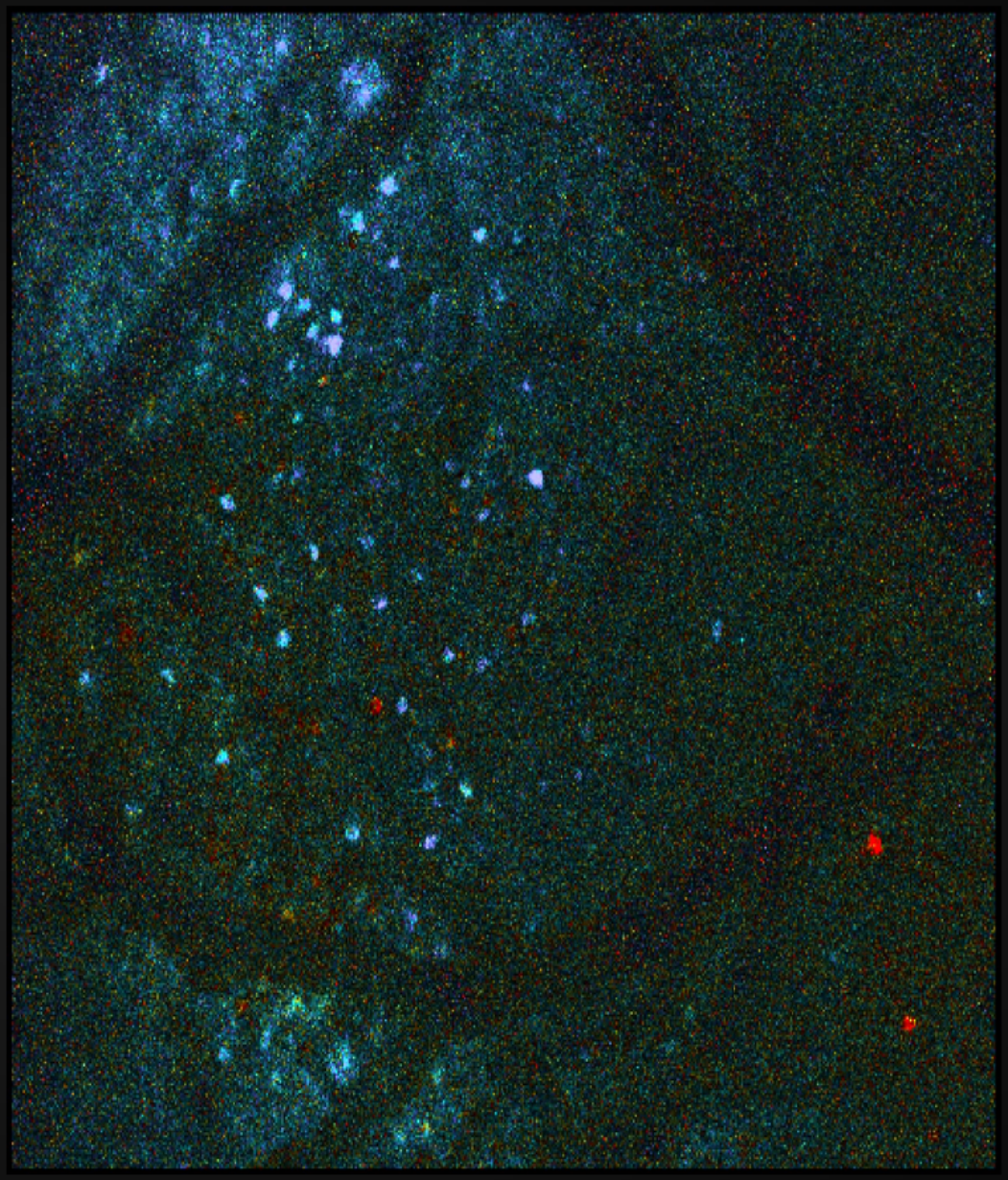
Hadoop

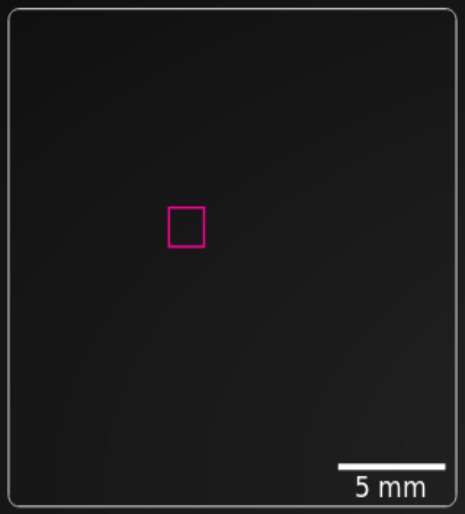
Spark

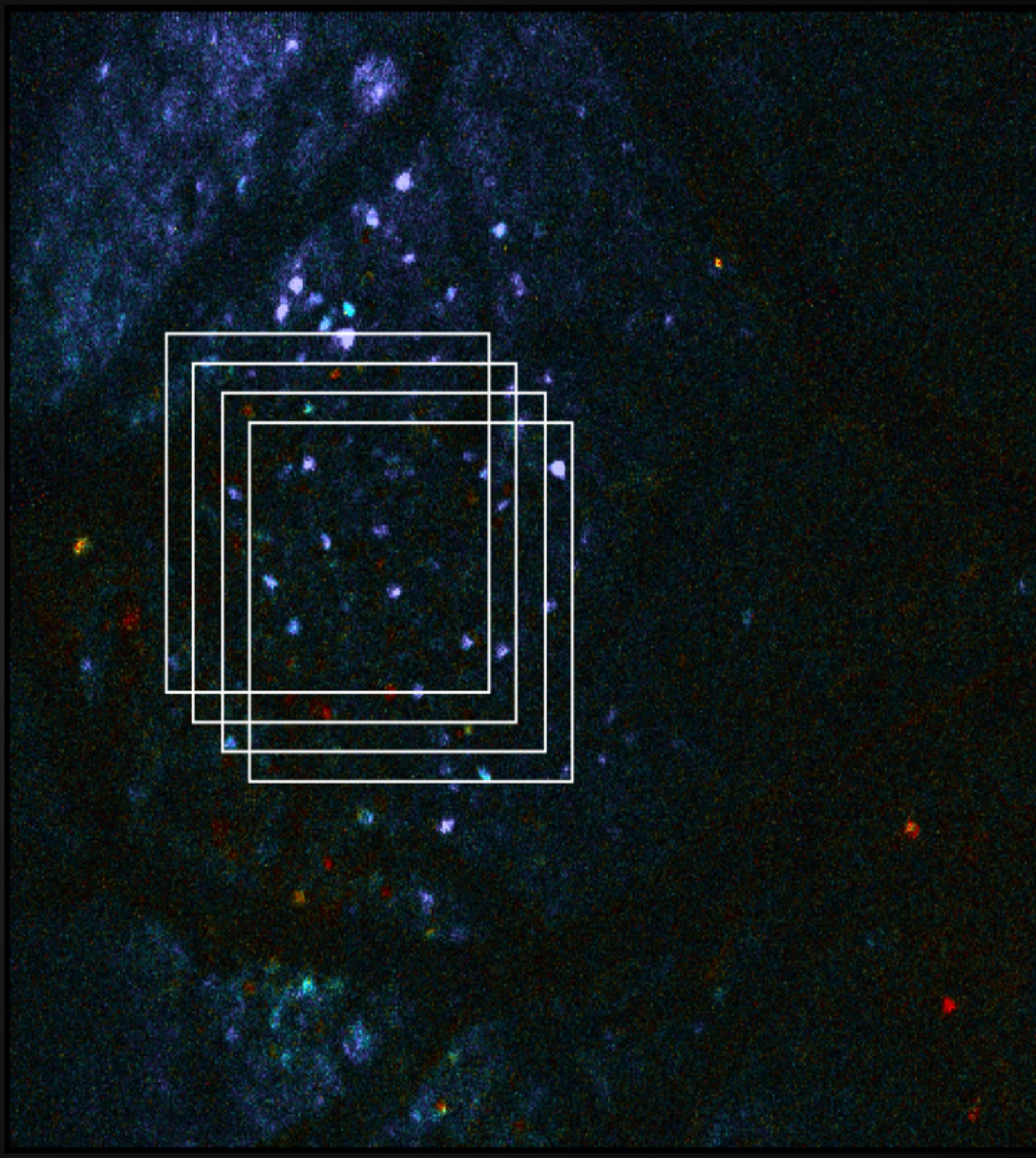
Streaming batch processing




5 mm




5 mm

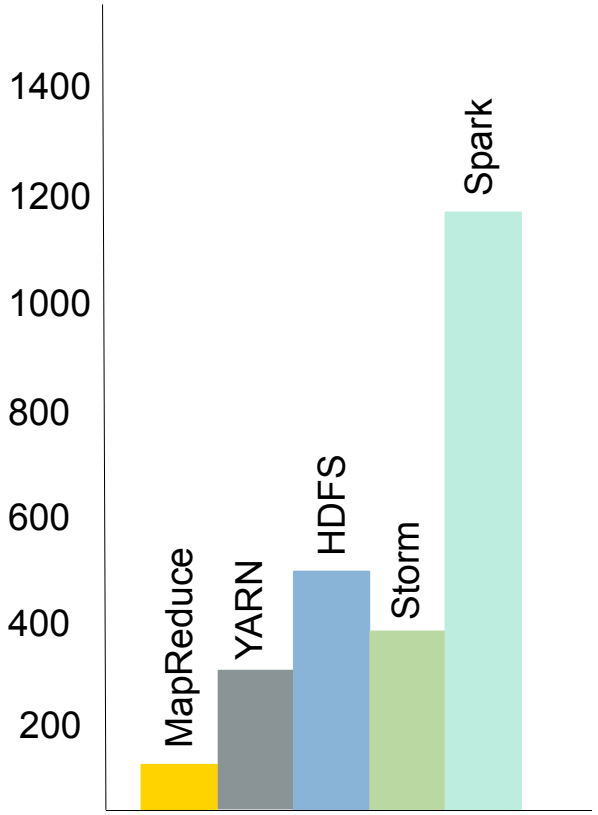


Единая платформа для приложений Big Data

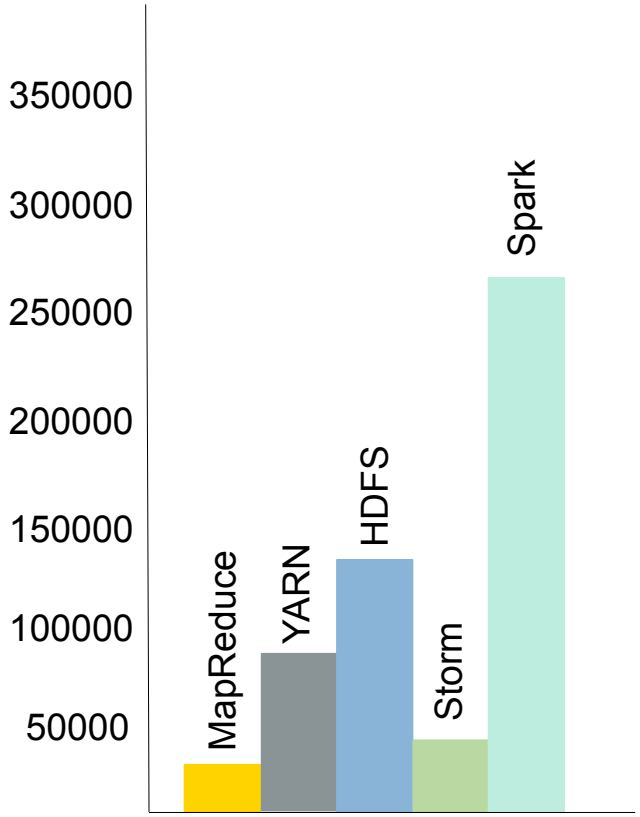
Единый API для выполнения различных задач в различных типах хранилищ и средах выполнения

Контрибьютеры

Сравнение с другими проектами



Коммиты



Измененные строчки кода

Активность за последние 6 месяцев

Спасибо за внимание!

Егор Пахомов

Разработчик

Группа технологий работы
с большими данными

epahomov@yandex-team.ru