BIGDATA TEAM

MegaFon Course: Big Data

Модель вычислений Spark: RDD

Andrey Titov, andrey.titov@bigdatateam.org
Big Data Instructor @ BigData Team, http://bigdatateam.org/
Senior Spark Engineer @ NVIDIA

22.07.2019, Moscow, Russia

BIGDATA TEAM

На этом занятии

- обзор Apache Spark
- устройство RDD
- базовые операции с RDD
- PairRDD функции
- кеширование и персистентность
- бродкасты
- управление параллелизмом
- недостатки RDD
- RDD workshop

Обзор Apache Spark

Общие сведения

- платформа для распределенных вычислений
- эволюция MapReduce
- поддержка SQL
- Режимы работы batch и streaming
- API на Python и Scala

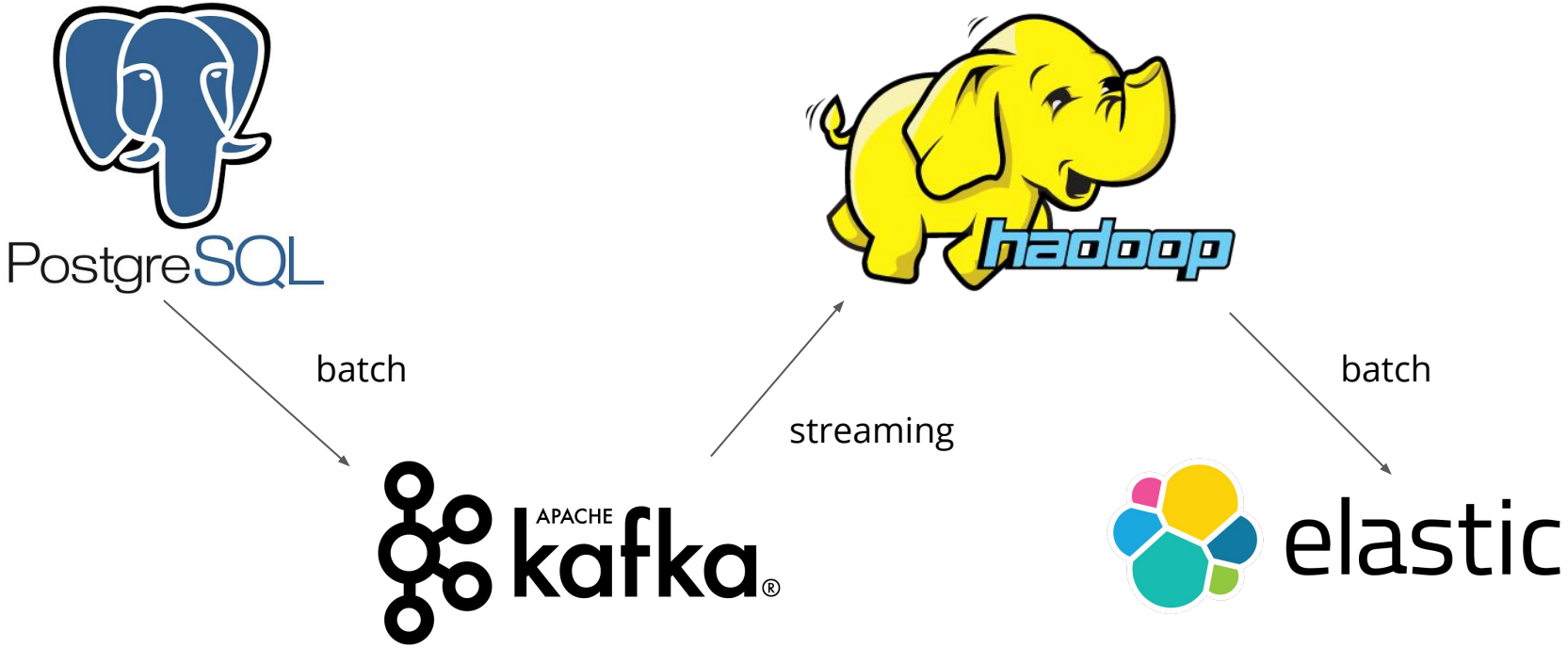
https://spark.apache.org

BIGDATA TEAM

Обработка больших данных

- распределенная обработка данных
- Чтение и запись в распределенные системы хранения
- отказоустойчивые вычисления
- масштабируемая производительность

Работа с разными источниками



BIGDATA TEAM

Распределенные SQL запросы

```
df = spark.read.json("vehicles.json")
df \
  .groupBy("brand", "model") \
  .count() \
  .filter(col("brand") == "dlc") \
  .join(sales, "model") \
  .select("brand", "model",
   "revenue", "sold") \
  .show(20, False)
```

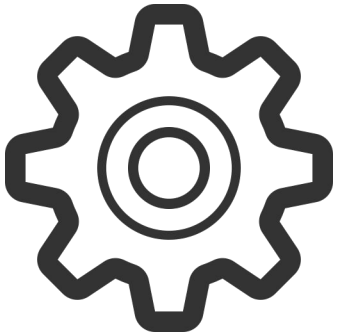
- поддержка batch и streaming
- join и window функции
- UDF функции
- оптимизатор запросов
- интеграция с источниками
- взаимозаменяемость API и SQL

BIGDATA TEAM

Поточная обработка данных

обогащение

сырые логи



приложение Spark

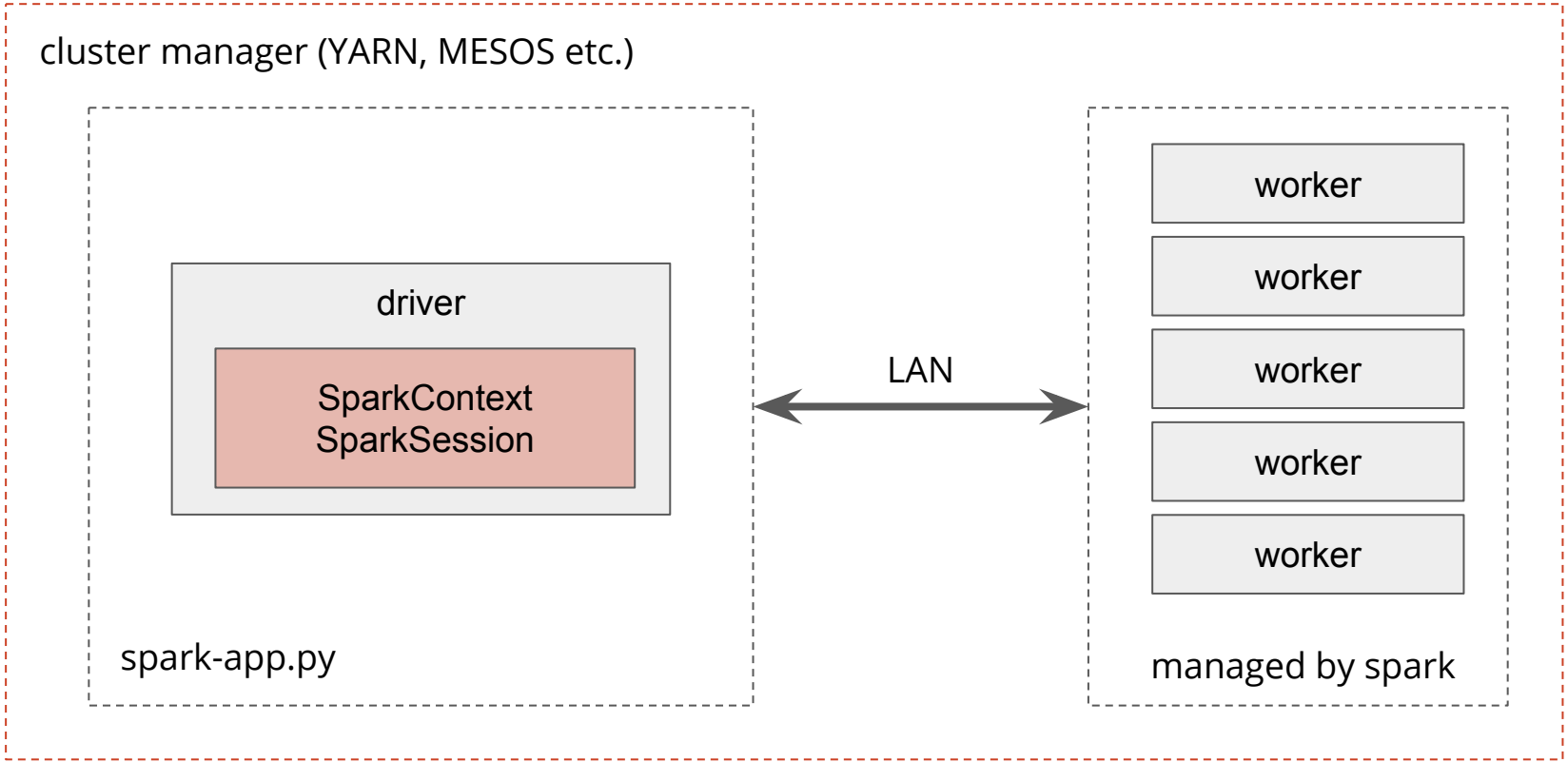
полезные метрики

BIGDATA TEAM

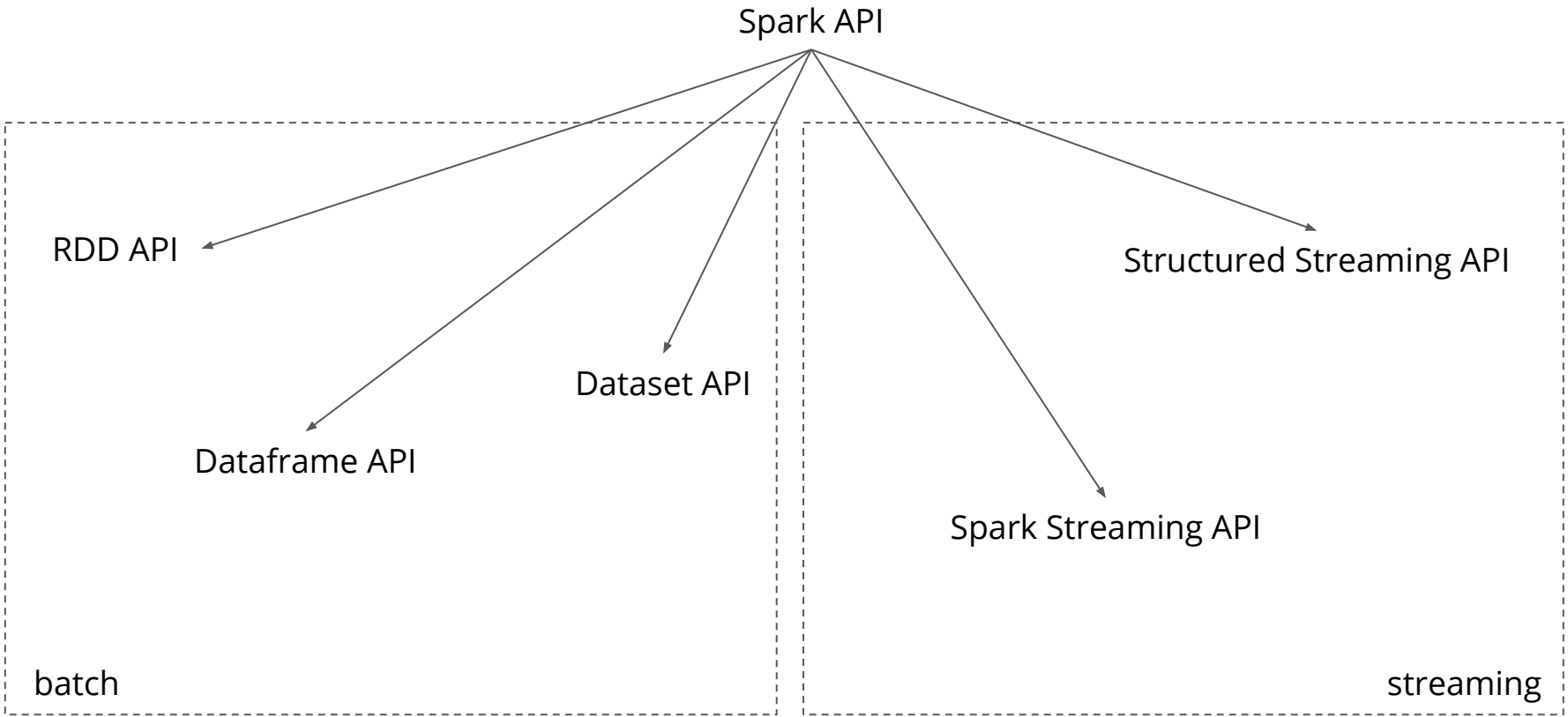
Ограничения применимости

- low latency
- business critical
- exactly once consistency
- ACID

Архитектура приложения



Spark API


Spark API
RDD API
Dataframe API
Dataset API
Structured Streaming API
Spark Streaming API
batch
streaming

BIGDATA TEAM

Полезная информация (QA)

Q: нужен ли мне кластер, чтобы начать писать код на Spark?
A: нет, не нужен - достаточно ноутбука и установленного дистрибутива Spark

Q: как запускаются Spark приложения?
A: в дистрибутив Spark входит утилита spark-submit - любое приложение, использующее spark, запускается с помощью нее

Q: как мне запустить python shell с поддержкой Spark?
A: для этого следует использовать pyspark, входящий в дистрибутив Spark

Q: как писать тесты?
A: с помощью pytest-spark

Q: достаточно ли мне знать python, чтобы разрабатывать высокопроизводительные spark-приложения?
A: если вы используете RDD API, то нет. В python они работают значительно медленнее, чем на Scala или Java. Другое дело, если используется Dataframe API. В этом случае нет большой разницы на чем писать. Однако UDF функции все равно следует писать на Scala и Java

Q: что мне использовать для разработки - RDD API или Dataframe API?
A: по умолчанию всегда используйте Dataframe API. Используйте RDD API, только если вы на 100% уверены в этом и знаете, зачем вам это нужно. Не используйте RDD API в python.

Устройство RDD

Что такое RDD?

- неизменяемая распределенная ленивая партиционированная коллекция данных
- основной примитив, который используется “под капотом” всех Spark API
- создается на основе коллекции на драйвере или внешнего датасета

https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Как создавать RDD

```
# Обычный список
simple_list = ["Apple", "Banana", "Orange"]

# RDD, созданная из обычного списка
simple_rdd = sc.parallelize(simple_list)

# RDD, созданная из файла
file_rdd = sc.textFile("large_file.csv")

# Список, полученный из RDD
collected_list = simple_rdd.collect()

# Обыкновенный out of memory :)
common_mistake = file_rdd.collect()
```

Базовые операции с RDD

Операции над RDD

Существует два класса операций:

- трансформация (transformation)
- действие (action)

Трансформации:

- превращают одну RDD в другую RDD
- не запускают вычислений - ленивые

Действия:

- запускают вычисления
- результат действия возвращается на драйвер, либо во внешнее хранилище

Операции над RDD

```
# Создаем RDD
simple_list = ["Apple", "Banana", "Orange"]
rdd = sc.parallelize(simple_list)

# Пример трансформаций
lower_rdd = rdd.map(lambda x: x.lower())
m_rdd = rdd.filter(lambda x: x.startswith('A'))

# Примеры действий
two_elements = rdd.take(2)
local_list = rdd.collect()
count = rdd.count()
```

PairRDD функции

PairRDD функции

- PairRDD - RDD, состоящее из кортежей (key, value)
- PairRDD обладают набором дополнительных функций, например:
  - reduceByKey(f)
  - sortByKey()
  - join(rdd)
  - leftOuterJoin(rdd)
  - rightOuterJoin(rdd)
  - mapValues(f)

PairRDD функции

```
# Создаем RDD
simple_list = ["Apple", "Banana", "Orange", "Avocado"]
rdd = sc.parallelize(simple_list)

# Создаем PairRDD
pair_rdd = rdd.map(lambda x: (x[0], len(x)))

# Вычисляем что-нибудь :)
agg = pair_rdd.reduceByKey(lambda x,y : x + y)
max_agg = agg.max(key=lambda x: x[1])
```

Управление параллелизмом

Партиционирование

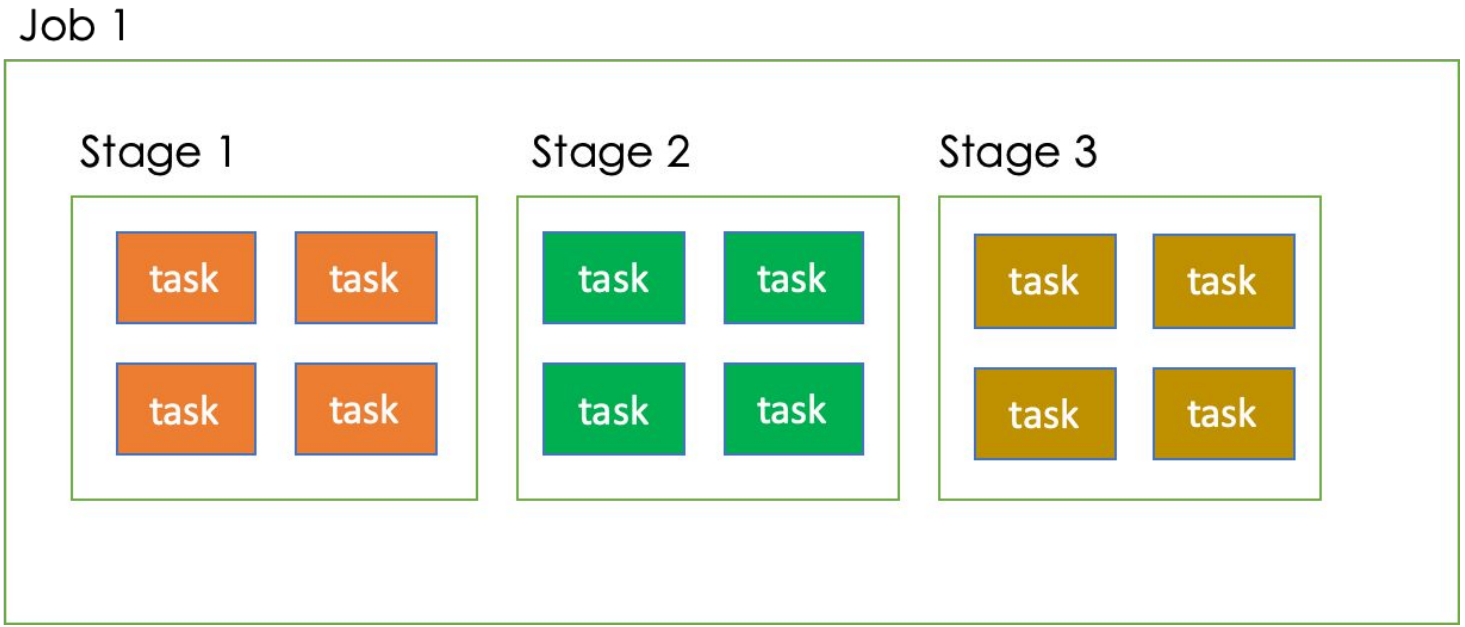
- любые RDD вне зависимости от источника и режима работы Spark состоят из партиций
- при создании RDD Spark управляет партициями автоматически и не всегда это происходит оптимально
- при чтении источника (например, базы данных) партиционирование определяется реализацией класса чтения данного источника
- при создании RDD с помощью sc.parallelize количество партиций определяется, исходя из количества ядер CPU, доступных Spark приложению

Перемешать данные между executor’ами можно с помощью:

- coalesce()
- repartition()
- repartitionAndSortWithinPartitions()

Структура DAG

```
sc.parallelize(x).map(y).reduceByKey().filter(z).mapPartitions(a).repartition(b).collect()
```



Структура DAG

- task’и выполняются внутри executor’а параллельно
- stage’ы внутри job выполняются последовательно
- stage’ы разных Job могут выполняться одновременно
- новый stage появляется при использовании shuffle
- shuffle - это «перемешивание» данных между executor’ами
  - map, filter - не приводят к shuffle
  - reduceByKey, groupByKey, repartition - приводят к shuffle
  - shuffle происходит по сети
  - shuffle - это медленно, но иногда полезно

Кеширование и персистентность

Кеширование и персистентность

```
# По умолчанию Spark не сохраняет промежуточные результаты задачи
simple_list = ["Apple", "Banana", "Orange", "Avocado"]
rdd = sc.parallelize(simple_list)
pair_rdd = rdd.map(lambda x: (x[0], len(x)))

# Каждое действие приведет к полному пересчету графа
action1 = pair_rdd.reduceByKey(lambda x,y : x + y).count()
action2 = pair_rdd.reduceByKey(lambda x,y : x + y).collect()
action3 = pair_rdd.reduceByKey(lambda x,y : x + y).filter(f).count()

# Для сохранения промежуточного результата следует использовать cache()
tmp_data = pair_rdd.reduceByKey(lambda x,y : x + y)
tmp_data.cache()

# общая часть графа будет выполнена только один раз
action1 = tmp_data.count()
action2 = tmp_data.collect()
action3 = tmp_data.filter(f).count()
```

Кеширование и персистентность

- в случае обновления данных на источнике закешированные данные не изменятся
- cache() - это persist() с использованием уровня кеширования по-умолчанию
- доступные уровни кеширования:
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - DISK_ONLY
  - MEMORY_ONLY_2, MEMORY_AND_DISK_2, DISK_ONLY_2

Бродкасты

Бродкасты

- бродкаст - объект, скопированный вручную на каждый executor
- удобно использовать для обогащения данных на executor’ах (aka map-side join)
- если бродкаст случайно удалится - сам по себе не восстановится :)
- размер бродкаста ограничен размером памяти executor’а
- без бродкаста локальные объекты с драйвера будут постоянно передаваться по сети на executor’ы

Бродкасты

```
# Создаем RDD и бродкаст объект
simple_list = ["Apple", "Banana", "Orange", "Avocado"]
some_dict = {"A": 0, "B": 1, "C": 2}
bc_dict = sc.broadcast(some_dict)
rdd = sc.parallelize(simple_list)
pair_rdd = rdd.map(lambda x: (x[0], len(x)))


# Используем бродкаст объект для map-side join
mapped_rdd = pair_rdd.map(lambda x: bc_dict.value.get(x[0],0))
mapped_rdd.take(2)
```

Недостатки RDD

Недостатки RDD

- нет встроенной поддержки SQL
- ручная очистка данных
- нет оптимизации вычислений
- недостаточная поддержка источников
- неудобный Streaming API
- недостаточная поддержка источников
- низкая скорость работы

Workshop

Workshop

1. Создайте RDD из файла на HDFS (файл расположен на HDFS по пути hdfs://user/atitov/data1.json)
2. Получите список всех ключей
3. Исключите дубликаты из RDD (дубликатом считаются записи с одинаковыми значениями полей name и country)
4. Удалите нулевые элементы
5. Найдите город с самым большим населением
6. Посчитайте топ-2 континентов по населению
7. Добавьте к каждой записи новое поле
8. Измените партиционирование RDD по значению поля continent

Thank you! Questions?

Feedback: http://rebrand.ly/mf2019q2_feedback_06_sparkrdd

Andrey Titov, andrey.titov@bigdatateam.org
Big Data Instructor @ BigData Team, http://bigdatateam.org/
Senior Spark Engineer @ NVIDIA