Уральский федеральный университет

Параллельные вычисления

Apache Spark

Созыкин Андрей Владимирович

к.т.н.

зав.кафедрой высокопроизводительных компьютерных технологий

Достоинства и недостатки Hadoop

- Достоинства:
  - Простая модель программирования Map Reduce
  - Работа с большим объемом данных
  - Автоматизация распараллеливания
  - Защита от сбоев
- Недостатки
  - Длительный цикл разработки (реализация Mapper, Reducer, Driver, компиляция, упаковка в архив jar, копирование на кластер, запуск и т.д.)
  - Фиксированная последовательность обработки, нет workflow
  - Обязательная запись данных на диск после обработки
  - Сложно реализовать join
  - Только пакетная обработка
  - Чтение всех данных

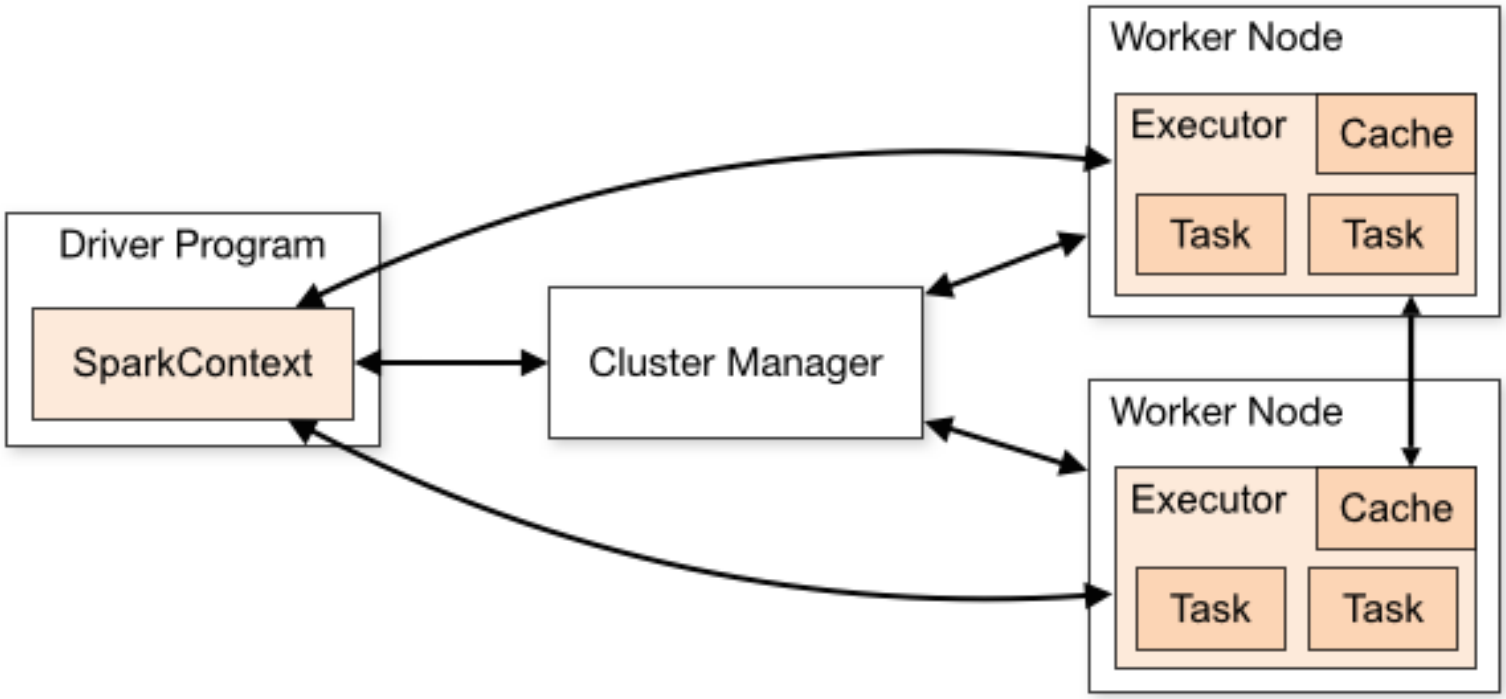
Экосистема Hadoop

- Средства аналитики:
  - Apache Pig – язык потоков данных
  - Apache Hive – SQL-like язык
- Базы данных с произвольным доступом
  - Apache HBase
  - Cassandra
- Обработка данных в памяти
  - Apache Spark
- Обработка потоковых данных
  - Apache Storm
- Другие проекты
  - Sqoop, Flume, Mahout, Kafka, Oozie, Zookeeper и др.

Apache Spark

- Создан в лаборатории AMPLab университета Berkeley
  - В 2010 году открыты исходные коды
  - Сейчас top-level проект apache (http://spark.apache.org/)
- Отличительные особенности
  - Распределенная обработка данных в оперативной памяти
  - Нет жесткой привязки к MapReduce
  - Интеграция с Hadoop (HDFS, InputFormats и т.п.), системами управления кластерами (Mesos, YARN)
- Написан на Scala, API для языков
  - Scala
  - Java
  - Python

Компоненты Spark



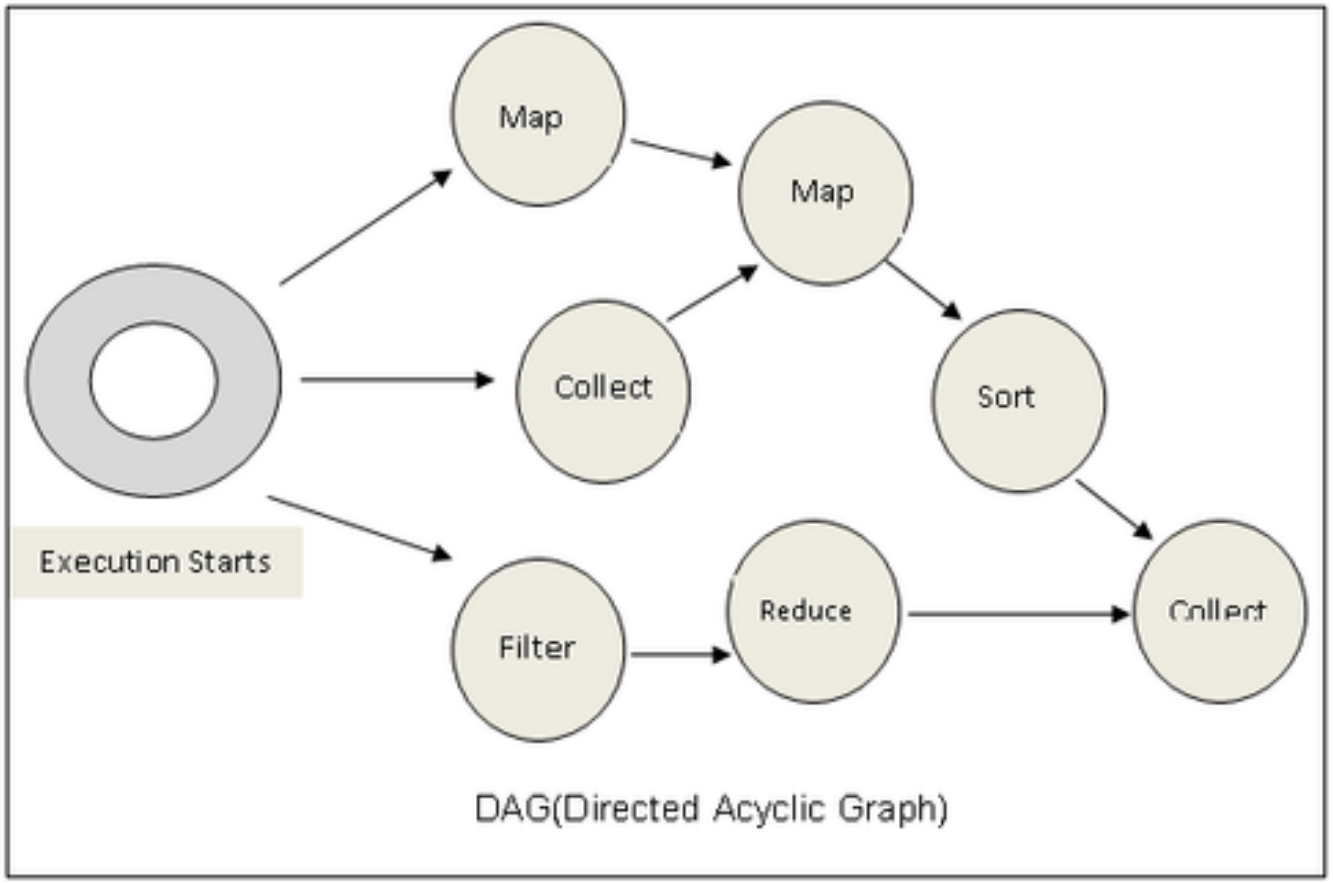
Нужен ли MapReduce?

- Google предложил программную модель MapReduce в целях
  - Автоматизация распараллеливания
  - Защита от сбоев
- Фиксированный алгоритм MapReduce
  - Единственно возможный вариант для таких целей?
  - Возможны ли другие варианты?
  - Какими свойствами они должны обладать?

Обработка данных в Spark

- Абстракции Spark
  - Трансформация (Transformation) – создание нового набора данных из существующего
  - Действие (Action) – вычисление над набором данных, возвращение значения в контекст
- Примеры трансформаций:
  - map(function) (flatMap, mapPartitions)
  - filter(function)
  - join(otherDataset)
- Примеры действий:
  - reduce(function)
  - collect()
  - saveAsTextFile(path)

Spark DAG


Map
Map
Collect
Sort
Execution Starts
Filter
Reduce
Collect
DAG(Directed Acyclic Graph)

Данные в Spark

- Какими должны быть данные?
  - Распределенное хранение в памяти серверов кластера
  - Возможность выполнения Action и Transformation
  - Итеративная обработка

Данные в Spark

- Какими должны быть данные?
  - Распределенное хранение в памяти серверов кластера
  - Возможность выполнения Action и Transformation
  - Итеративная обработка
  - Надежность?

Resilient Distributed Datasets (RDD)

- Spark использует абстракцию Resilient Distributed Datasets (RDD)
  - 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. Technical Report UCB/EECS-2011-82. July 2011.
- Определение RDD:
  - Разделенная на части коллекция записей, доступная только для чтения
  - Read-only, partitioned collection of records
- RDD может быть создана только *детерминистическими* операциями:
  - Загрузка данных с диска
  - Преобразование существующего RDD

Resilient Distributed Datasets (RDD)

• Почему важно использовать детерминистические операции?

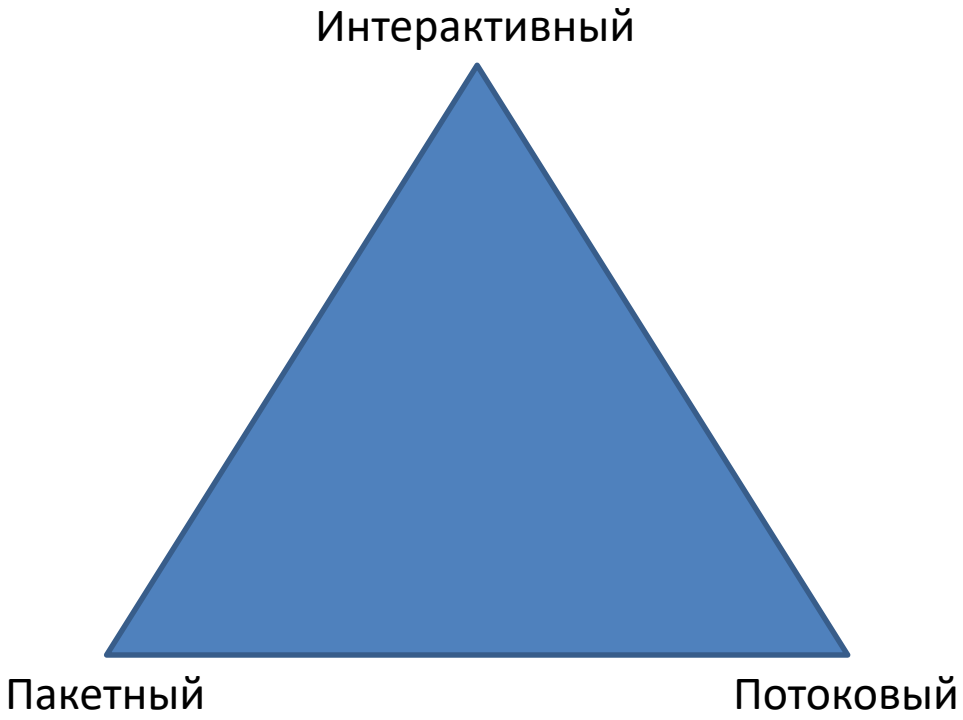
Resilient Distributed Datasets (RDD)

- Почему важно использовать детерминистические операции?
- Надежность:
  - При выходе из строя узла кластера партиция RRD может быть пересчитана на основе данных с диска

Программа в Spark

- Данные
  - RDD
- DAG
  - Трансформации
  - Действия
- Ленивый режим
  - Новые RDD не создаются только тогда, когда они используются первый раз

Режимы работы Spark



Режимы работы Spark

- Локальный режим
  - Spark работает на локальной машине, есть возможность использовать несколько процессов
- Кластерный режим
  - Spark Standalone
  - Apache Mesos
  - Apache YARN (Hadoop v2)

Интерактивный режим

- Командная строка Spark Shell
  - spark-shell (Scala)
  - pyspark (Python)
- SparkContext
  - Представляет соединение с кластером Spark (или локальным Spark)
  - Специальная версия SparkContext для интерактивного режима
  - Переменная sc в Spark Shell

Интерактивный режим

```
$ pyspark
>>> textFile = sc.textFile("hdfs://data/wiki/en/articles-part")
>>> textFile.count()
3755
>>> filteredFile = textFile.filter(lambda s: 'London' in s)
>>> filteredFile.saveAsTextFile("hdfs://user/u1213/london")
```

Результаты в HDFS

```
$ hadoop fs -ls /user/u1213/london
Found 3 items
-rw-r--r--   3 u1213 supergroup          0 2015-05-24 12:18
/user/u1213/london/_SUCCESS
-rw-r--r--   3 u1213 supergroup   15303635 2015-05-24 12:18
/user/u1213/london/part-00000
-rw-r--r--   3 u1213 supergroup   14101956 2015-05-24 12:18
/user/u1213/london/part-00001
```

Как выполняется?

```
SparkContext: Starting job: saveAsTextFile at
NativeMethodAccessorImpl.java:-2
DAGScheduler: Got job 1 (saveAsTextFile at
NativeMethodAccessorImpl.java:-2) with 2 output partitions
DAGScheduler: Submitting 2 missing tasks from Stage 1 (MappedRDD[11] at
saveAsTextFile at NativeMethodAccessorImpl.java:-2)
TaskSetManager: Starting task 0.0 in stage 1.0 (TID 2, localhost, ANY,
1212 bytes)
TaskSetManager: Starting task 1.0 in stage 1.0 (TID 3, localhost, ANY,
1212 bytes)
HadoopRDD: Input split: hdfs://data/wiki/en/articles-
part:37297703+37297704
HadoopRDD: Input split: hdfs://data/wiki/en/articles-part:0+37297703
```

Как выполняется?

```
FileOutputCommitter: Saved output of task
'attempt_201505241218_0001_m_000001_3' to
hdfs:///user/u1213/london/_temporary/0/task_201505241218_0001_m_000001
SparkHadoopWriter: attempt_201505241218_0001_m_000001_3: Committed
Executor: Finished task 1.0 in stage 1.0 (TID 3). 1768 bytes result sent
to driver
TaskSetManager: Finished task 1.0 in stage 1.0 (TID 3) in 1757 ms on
localhost (1/2)
FileOutputCommitter: Saved output of task
'attempt_201505241218_0001_m_000000_2' to
hdfs://user/u1213/london/_temporary/0/task_201505241218_0001_m_000000
SparkHadoopWriter: attempt_201505241218_0001_m_000000_2: Committed
Executor: Finished task 0.0 in stage 1.0 (TID 2). 1768 bytes result sent
to driver
TaskSetManager: Finished task 0.0 in stage 1.0 (TID 2) in 1788 ms on
localhost (2/2)
```

Wordcount в Spark

```
wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word:
(word, 1)).reduceByKey(lambda a, b: a+b)
```

Кэширование в Spark

- RRD можно сохранить в памяти
  - `filteredFile.cache()`
  - Эффективно, если RRD будет часто использоваться
- Метод cache – только подсказка для Spark:
  - Данные могут не поместится в память, кэширования не будет
  - Смешанный режим: часть партиций будет в кэш, часть будет рассчитываться на основе данных с диска
- Удаление данных из памяти:
  - `filteredFile.unpersist()`
  - Spark автоматически удаляет неиспользуемые RRD из памяти с помощью LRU алгоритма

Пакетный режим в Spark. Задача

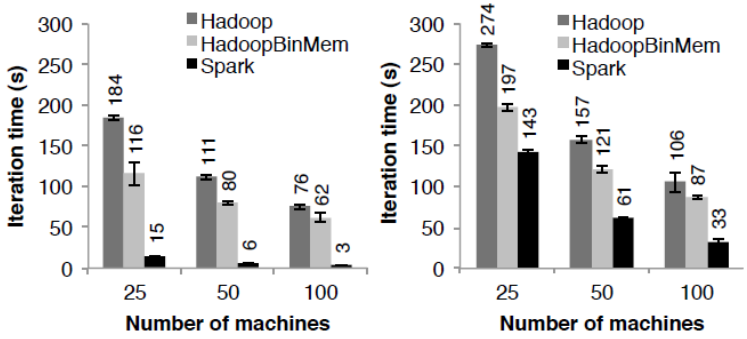
```
# spark-wordcount.py
from pyspark import SparkContext
# Создаем контекст
sc = SparkContext("local", “WordCount")
# Загружаем данные из HDFS
textFile = sc.textFile("hdfs://data/wiki/en/articles-part")
# Считаем WordCount
wordCounts = textFile.flatMap(lambda line:
line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a,
b: a+b)
# Записываем результаты в HDFS
wordCounts.saveAsTextFile("hdfs://user/u1213/spark-wordcount")
```

Пакетный режим в Spark. Запуск задачи

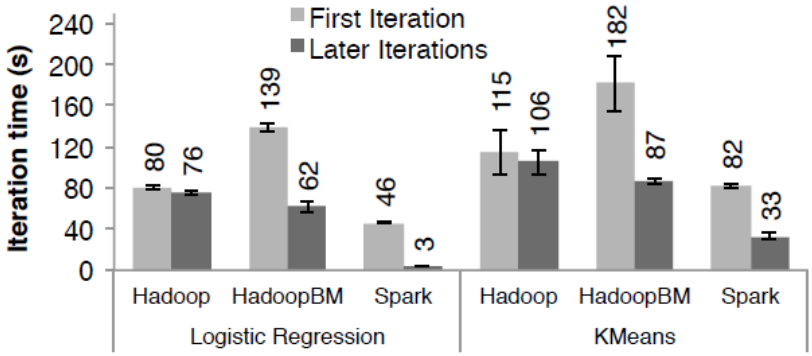
- Скрипт для запуска задач Spark
  - `spark-submit`
  - Используется для локального режима и любого кластера (Spark standalone, Mesos, Yarn)
- Пример для локального Spark:
  - `spark-submit --master local[*] spark-wordcount.py`
- Пример для Spark standalone:
  - `spark-submit --master spark://umu30.imm.uran.ru:7077 spark-wordcount.py`
- Пример для YARN:
  - `spark-submit --master yarn-cluster spark-wordcount.py`
  - `spark-submit --master yarn-client spark-wordcount.py`
  - Адрес YARN ResourseManager должен быть в опциях Hadoop (переменные окружения или конфигурационный файл)

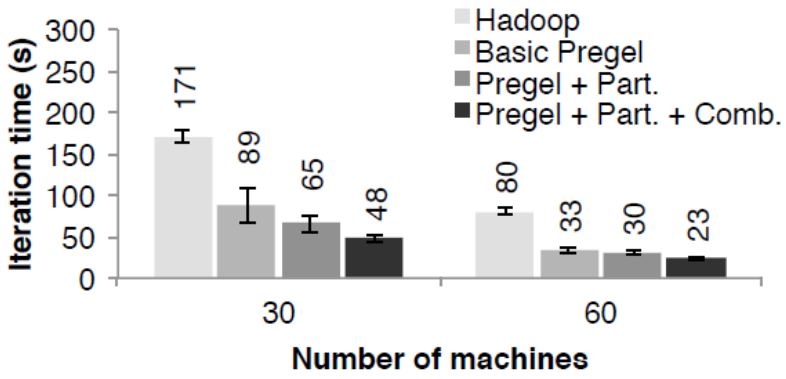
Почему это лучше?

Application	Data Description	Size
Logistic Regression	1 billion 9-dimensional points	100 GB
K-means	1 billion 10-dimensional points (k = 10 clusters)	100 GB
PageRank	Link graph from 4 million Wikipedia articles	49 GB
Interactive Data Mining	Wikipedia page view logs from October, 2008 to April, 2009	1 TB

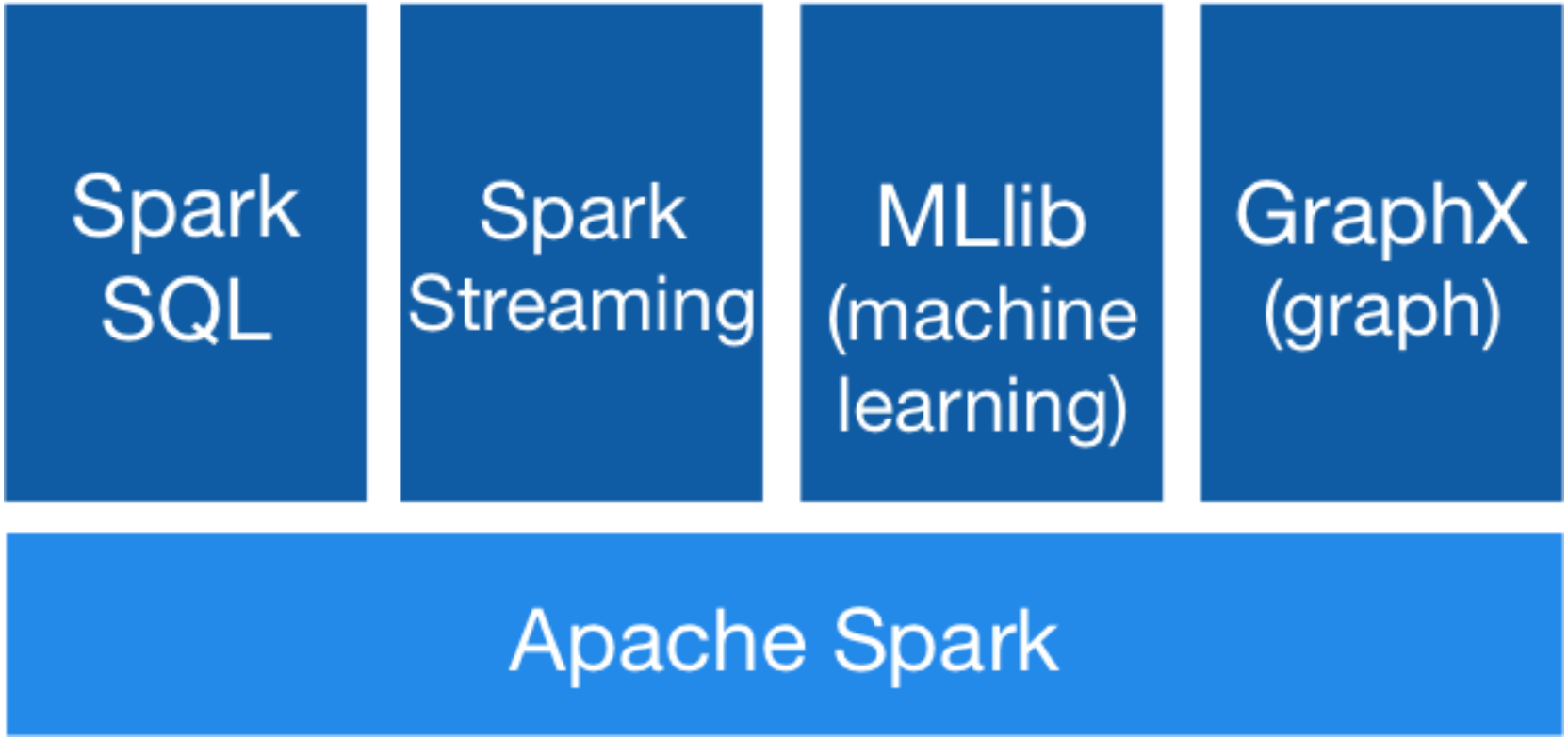


(a) Logistic Regression (b) K-Means

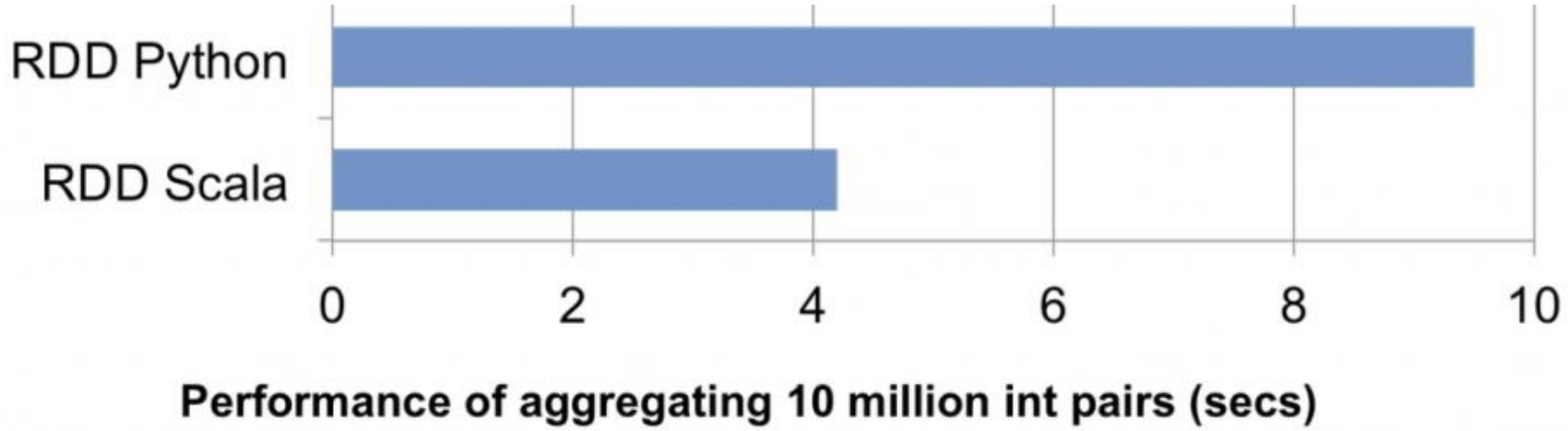




Экосистема Apache Spark



Производительность RDD

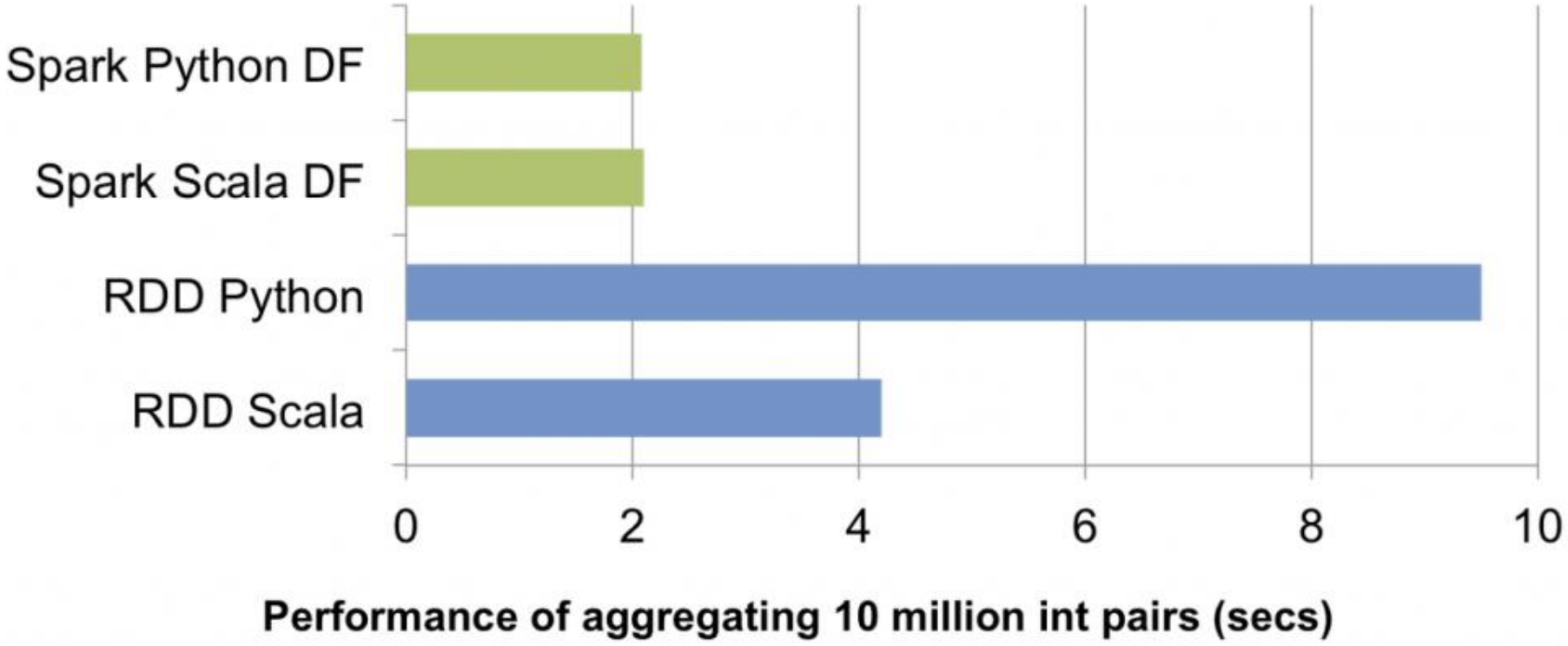


Источник - https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

Spark DataFrame API

- Высокоуровневый интерфейс программирования
  - Аналог DataFrame в Pandas и R
  - Работа со структурированными данными
- Выполнение
  - Трансформации и действия над RDD
- Оптимизация (Catalyst optimizer)
  - Логические оптимизации
  - Генерация кода JVM в рантайме

Производительность DataFrame



Источник - https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html

Использование DataFrame

```
Граф пользователей Twitter:
user_id \t follower_id

t_graph = sqlContext.read.csv("twitter.csv")

t_graph.printSchema()

t_graph.count()

t_graph.show(5)
```

Использование DataFrame

```
t_graph.select('user_id').show(5)

t_graph.filter(t_graph.user_id == "206754").show()

t_graph.groupBy('user_id').count(). \
    sort('count', ascending=False).show(5)

t_graph.join(numeric2screen, t_graph.user_id == \
    numeric2screen.user_id)
```

Spark SQL

Часть экосистемы Spark

Совместим с Hive

- Может использовать Hive Metastore с информацией о таблицах
- Может выступать в качестве средства запуска запросов Hive вместо MapReduce

Собственный парсер SQL

Spark SQL часть Spark DataFrame API

- SparkSQL может делать все, что умеет Spark DataFrame (и наоборот)

Spark SQL

Создание временной таблицы на основе DataFrame:

- `t_graph.createOrReplaceTempView('twitter')`
- `spark.sql('SELECT * FROM 'twitter')`

DataFrame на основе таблицы

- `twitter = spark.read.table('twitter')`
- `top20 = twitter.groupBy('user_id').count(). \`
  `sort('count', ascending=False).limit(20)`
- `top20.show()`

Сохранение DataFrame в виде таблицы

- `top20.write.saveAsTable('twitter_top20')`

Spark SQL

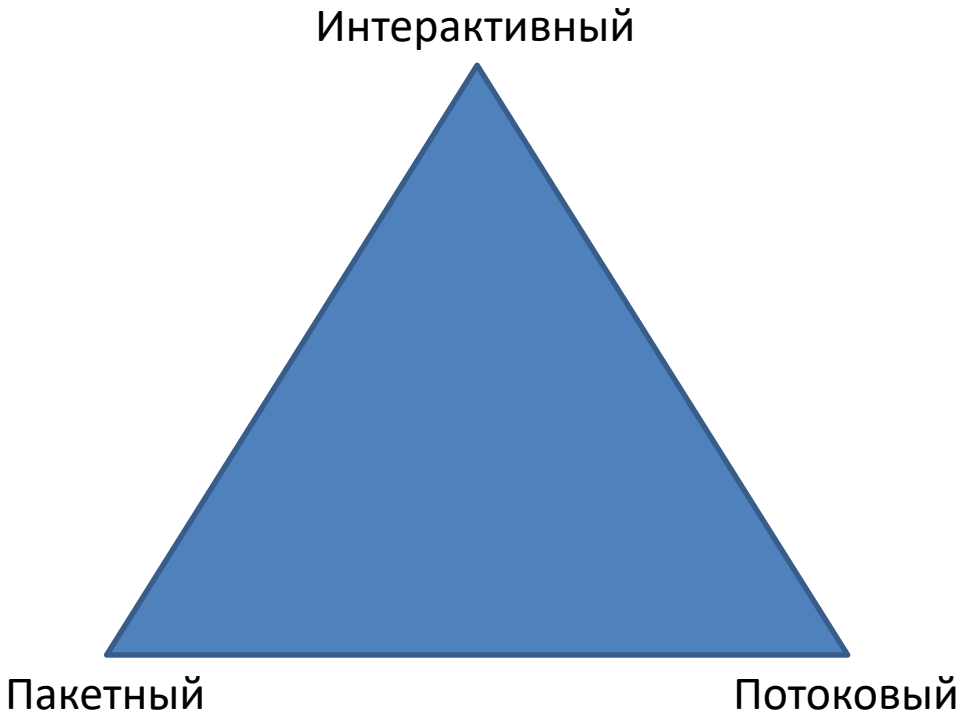
Пример использования:

- `spark.sql('SELECT user_id, COUNT(follower_id) AS fc FROM twitter GROUP BY user_id ORDER BY fc DESC LIMIT 50’)`

Как это будет выполняться:

- `spark.sql('SELECT user_id, COUNT(follower_id) AS fc FROM twitter GROUP BY user_id ORDER BY fc DESC LIMIT 50').explain()`

Режимы работы Spark



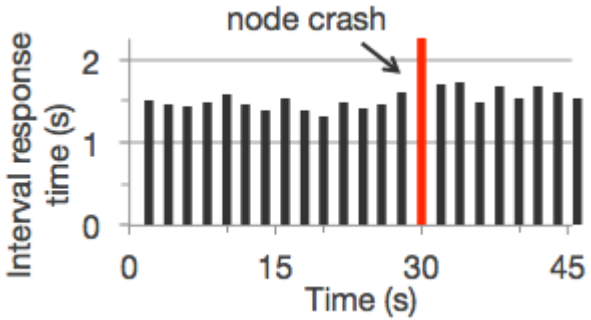
Spark Streaming

Типы Spark Streaming

- Spark Streaming – операции с RDD
- Structured Spark Streaming – операции с DataFrame

Отказоустойчивость

- Exactly-once semantics
- Используется ZooKeeper и HDFS

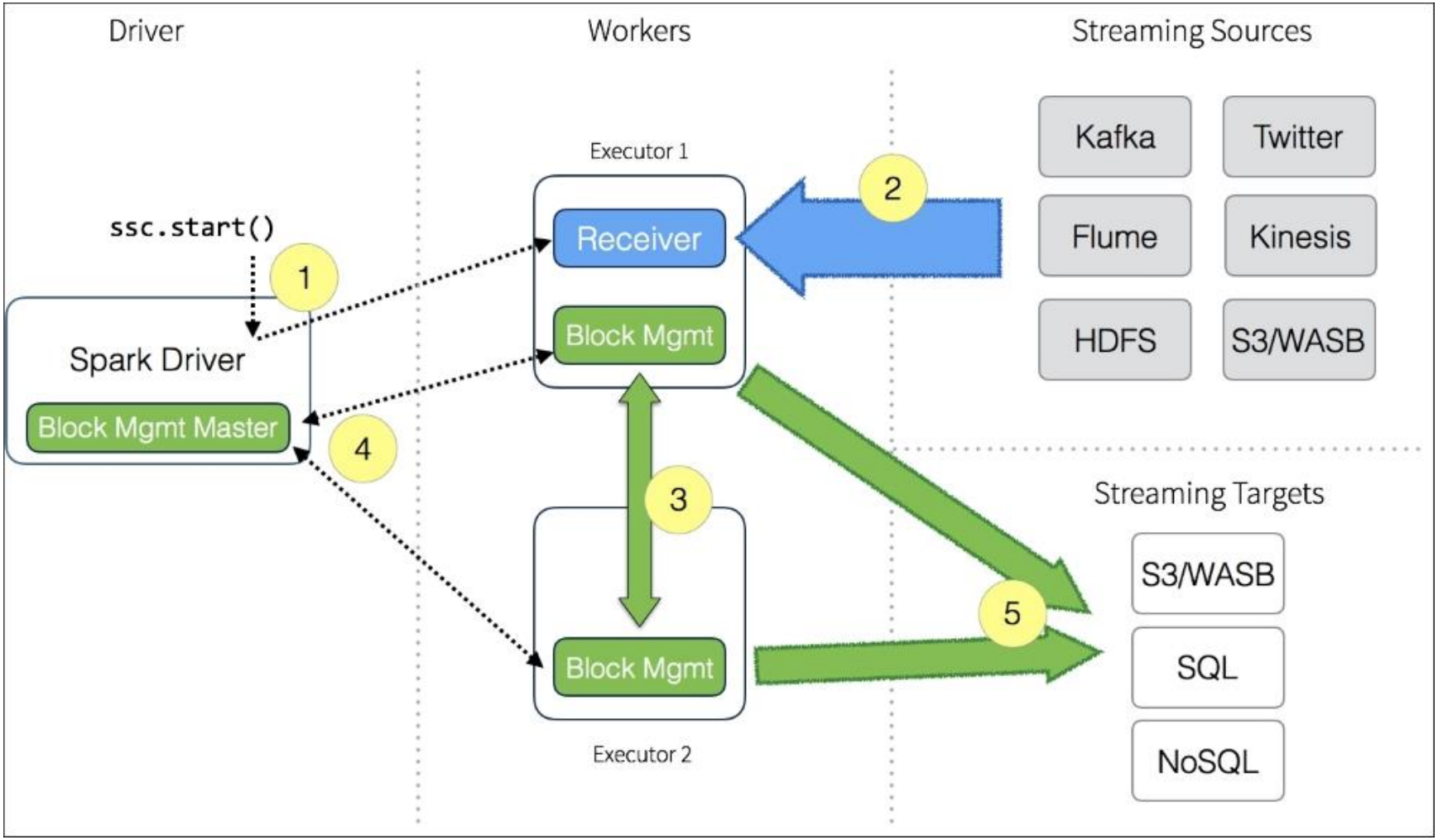


http://spark.apache.org/streaming/

Spark Streaming

Spark DStream

Поток данных Spark Streaming



Пример программы Spark Streaming

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="PythonStreaming")
sc.setLogLevel("ERROR")
ssc = StreamingContext(sc, 1) # Интервал 1 секунда
```

Пример программы Spark Streaming

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(appName="PythonStreaming")
sc.setLogLevel("ERROR")
ssc = StreamingContext(sc, 1) # Интервал 1 секунда
# Создаем Dstream
lines = ssc.socketTextStream("localhost", 9999))
```

Пример программы Spark Streaming

```
# Создаем Dstream
lines = ssc.socketTextStream("localhost", 9999))
# Считаем количество слов
counts = lines.flatMap(lambda line: line.split(" "))\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
counts.pprint()
```

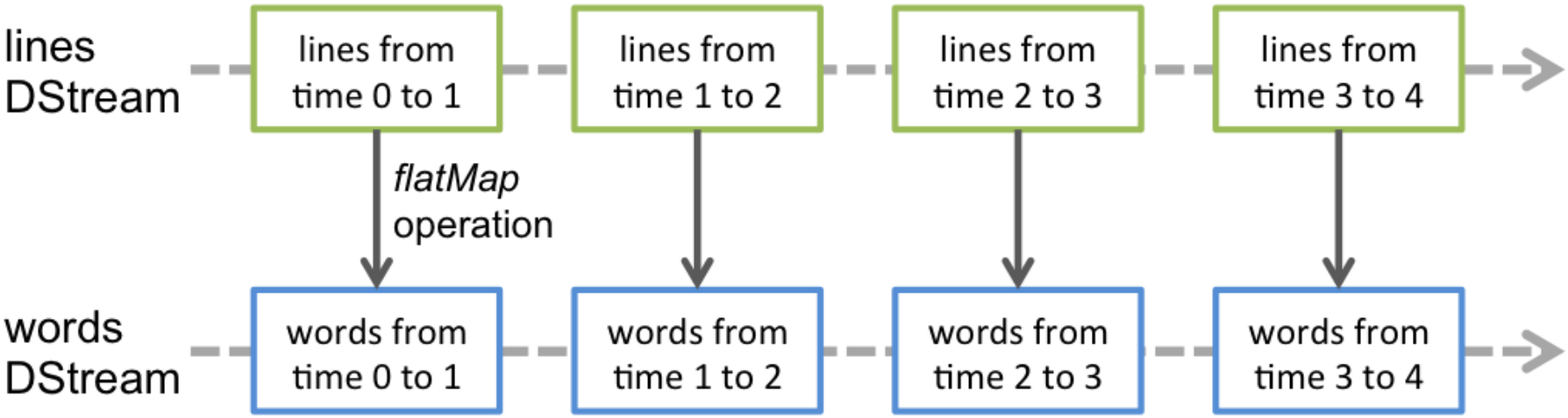
Пример программы Spark Streaming

```
# Создаем Dstream
lines = ssc.socketTextStream("localhost", 9999))
# Считаем количество слов
counts = lines.flatMap(lambda line: line.split(" "))\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
counts.pprint()
# Запускаем Spark Streaming Context
ssc.start()
ssc.awaitTermination()
```

Создаем поток данных

```
$ nc -lk 9999
Hello world!
Word count application.
spark spark spark
```

DStream


lines DStream
lines from time 0 to 1
lines from time 1 to 2
lines from time 2 to 3
lines from time 3 to 4
flatMap operation
words DStream
words from time 0 to 1
words from time 1 to 2
words from time 2 to 3
words from time 3 to 4

Итоги

- Apache Spark
  - Обработка данных в памяти RDD
  - Модель обработки данных DAG (трансформации и действия)
- Интерфейсы программирования
  - RDD
  - DataFrame
- Экосистема Apache Spark
  - Spark Streaming, Spark SQL, Spark MLlib, Spark GraphX

Вопросы?