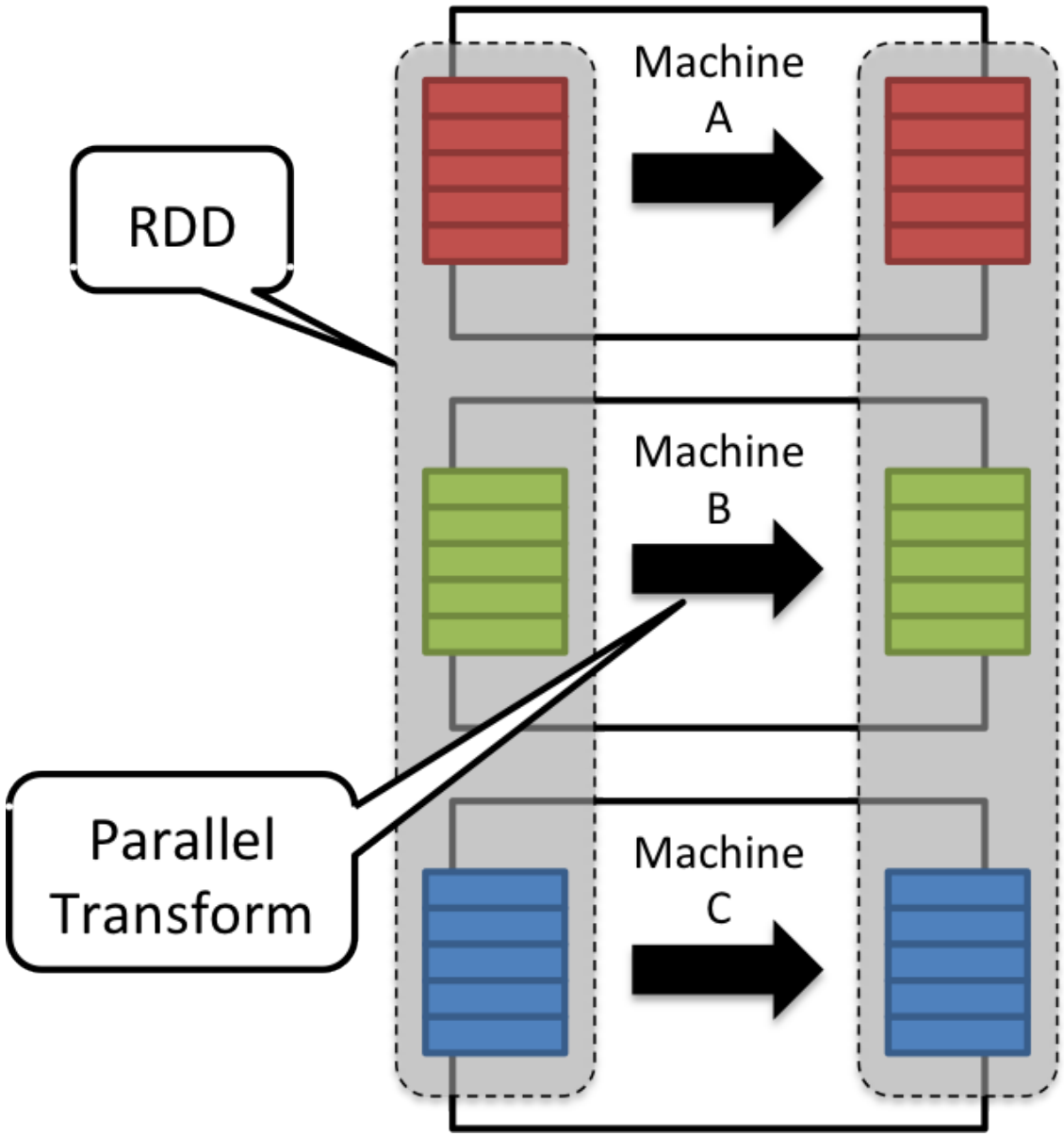
Распределённая обработка данных с Apache Spark

Притыковская Наташа

1 марта 2018 г.

Overview

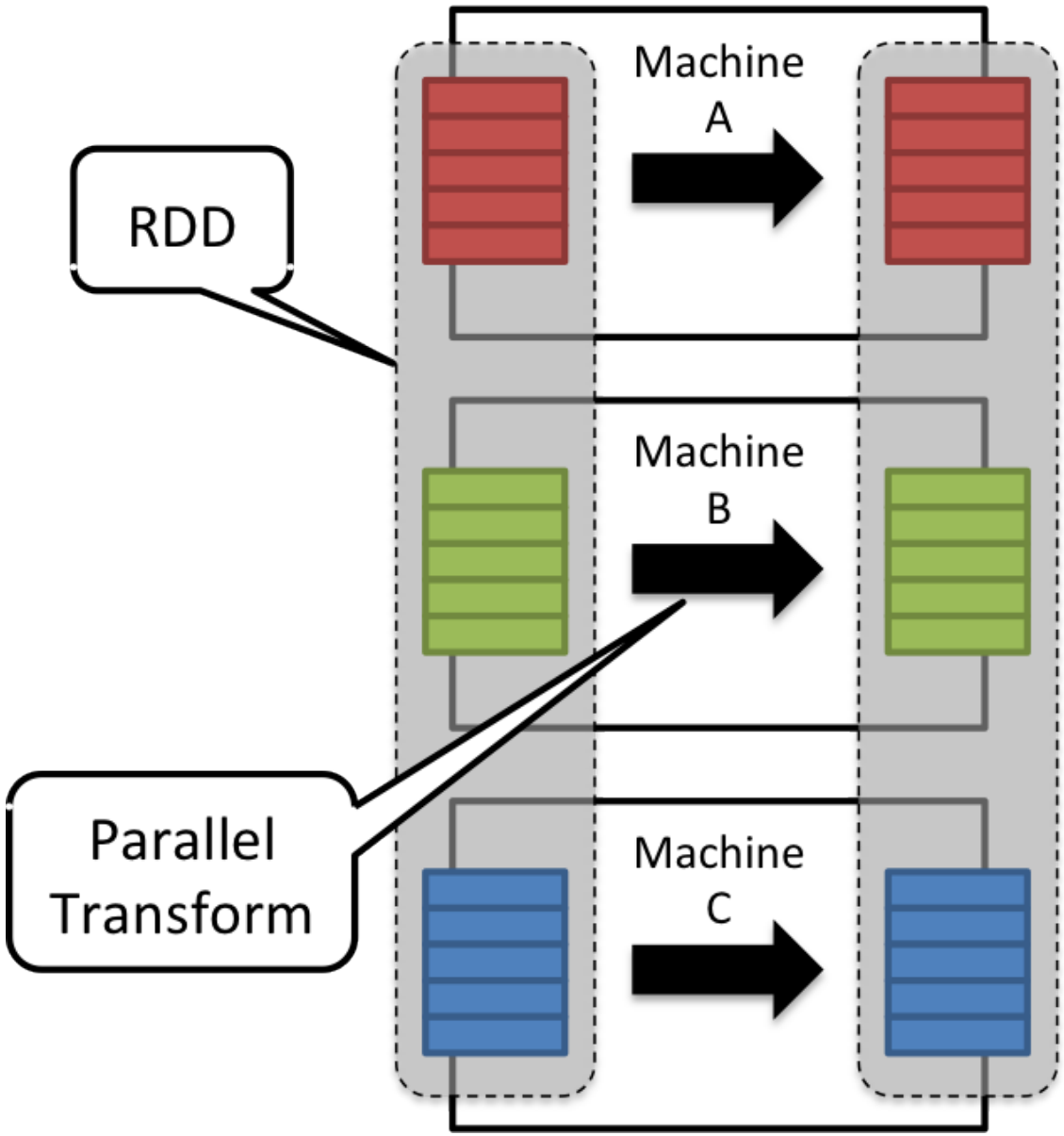
RDD



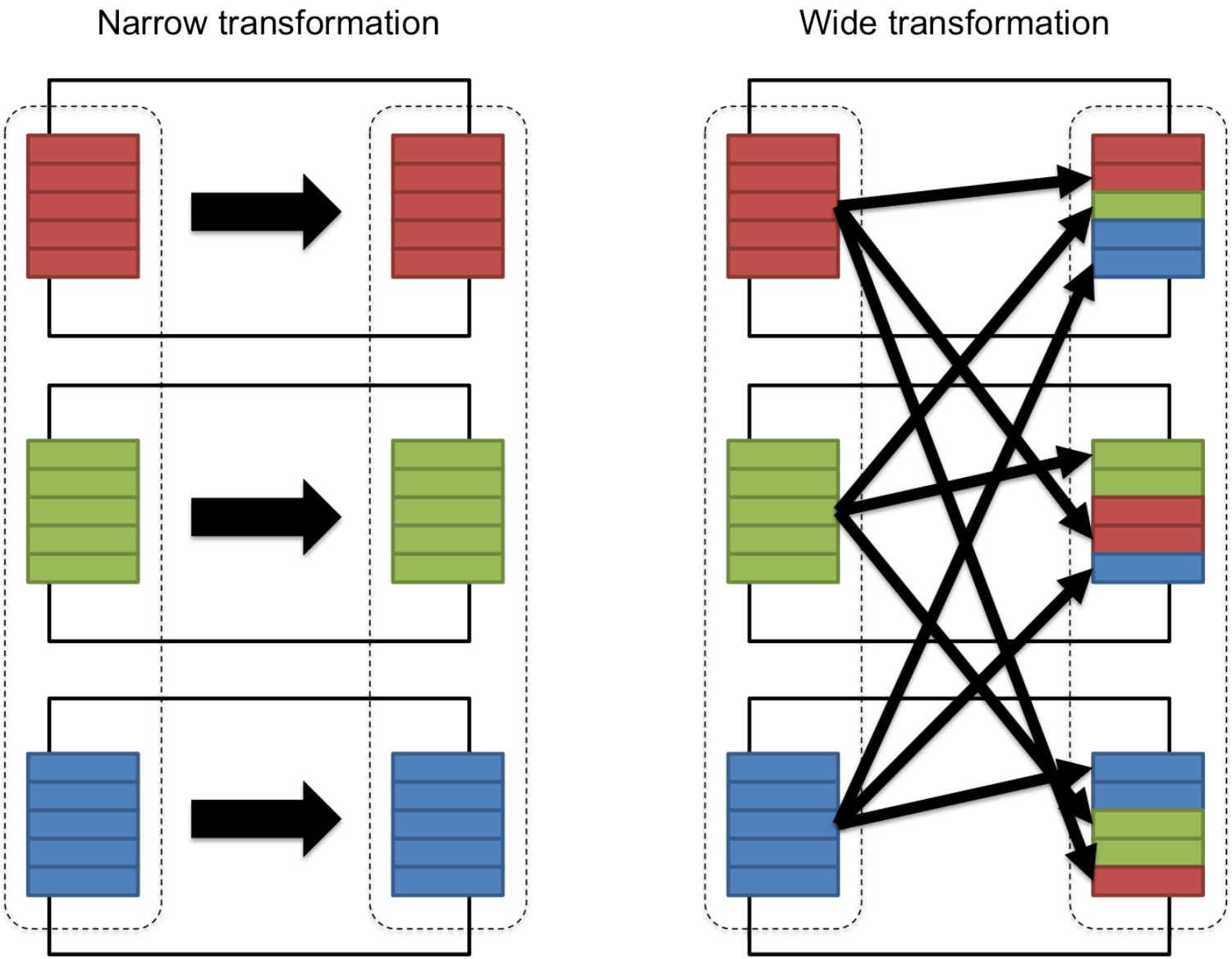
Выполнение Spark-программы

- Каждое действие (action) инициирует новое задание (job)
- Spark анализирует граф RDD и строит план выполнения
- План выполнения может включать несколько этапов (stages)
  - Этап включает набор трансформаций, которые можно выполнить без пересылки всех данных по сети (shuffle)
- Каждый этап состоит из набора задач (tasks)
  - Выполняют одинаковый код на различных фрагментах данных

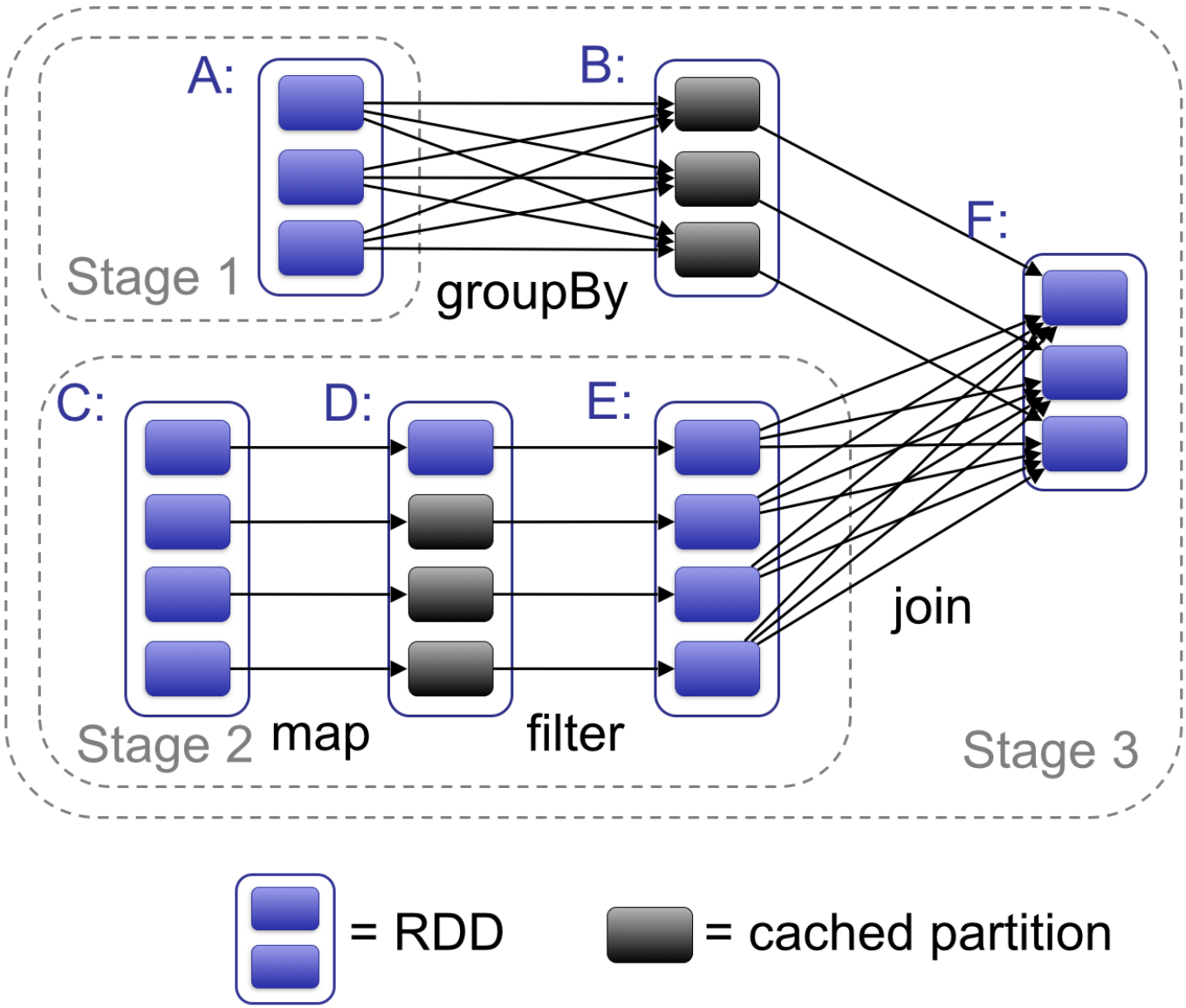
RDD



Типы преобразований



Разбиение на этапы



Настройка параллелизма

- Число задач в этапе равно числу фрагментов (partitions) у последнего RDD в этапе
- Число фрагментов в RDD
  - rdd.partitions.size()
  - обычно равно числу фрагментов у родительского RDD
  - у RDD, считываемых из HDFS, обычно равно числу HDFS-блоков
- Управление числом фрагментов
  - rdd.repartition(numPartitions)
  - rdd.reduceByKey(...., numPartitions=N) - у всех трансформаций с shuffle
- Увеличение числа задач путем увеличения числа фрагментов может положительно сказаться на производительности
  - Меньше объем обрабатываемых данных, меньше нагрузка на память и диск

Обработка отказов

- Промежуточные результаты не сохраняются в HDFS
- RDD по-умолчанию не реплицируются
- Для восстановления потерянных данных используется информация о происхождении
  - Трансформации над RDD должны быть детерминированными
  - После создания RDD нельзя изменять (immutable)
  - Spark восстанавливает потерянные фрагменты RDD, заново вычисляя их путем применения трансформаций

Кэширование данных?

- RDD.persist(StorageLevel)
- уровни хранения
  - *MEMORY_ONLY*, аналог вызова RDD.cache()
  - *MEMORY_AND_DISK*
  - ...
- Spark автоматически кэширует промежуточные данные для операций с shuffle
  - Например, reduceByKey()
- Удаление данных из кэша
  - Автоматическое (LRU cache)
  - RDD.unpersist()

Общие переменные

- по умолчанию при передачи функции на узел кластера происходит копирование значений используемых переменных
  - изменения переменных обратно не передаются
- Spark реализует 2 типа общих переменных
  - Broadcast-переменные
  - Аккумуляторы

Broadcast-переменные

- позволяют один раз передать и закэшировать значение переменной на всех машинах
- Семантика read-only

```
>>> broadcastVar = sc.broadcast([1, 2, 3])
<pyspark.broadcast.Broadcast object at 0x102789f10>

>>> broadcastVar.value
[1, 2, 3]
```

Аккумуляторы

- Переменные, значение которых может быть изменено только с помощью коммутативной и ассоциативной операции *add*
- Считывать значение может только основная программа драйвер

```
>>> accum = sc.accumulator(0)
Accumulator<id=0, value=0>

>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

scala> accum.value
10
```

Dataframe

- This API is inspired by data frames in R and Python (Pandas), but designed from the ground-up to support modern big data and data science applications.
- Dataframe - распределенная колоночная таблица

устройство Dataframe

collection of "generic"Row instances (as RDD[Row]) and a schema

- Row - упорядоченная коллекция полей
- Schema - описание структуры таблицы. A schema is described using StructType which is a collection of StructField objects (that in turn are tuples of names, types, and nullability classifier).

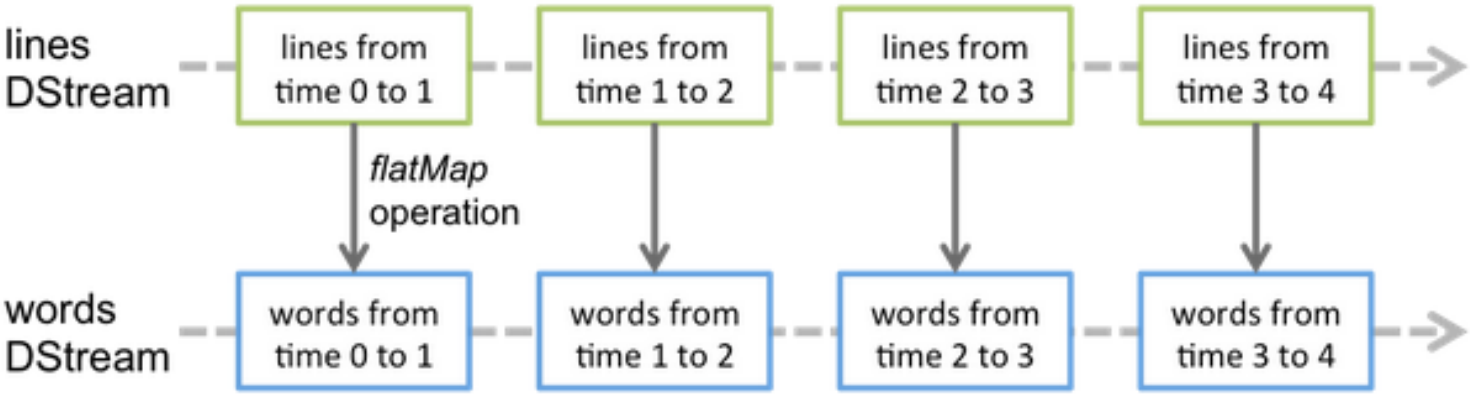
создание Dataframe

- "форматы"
  - json файлы, parquet файлы
  - HIVE таблицы
  - RDD
  - pandas data frame
- источники
  - HDFS
  - локальная файловая система
  - внешние реляционные базы через JDBC

Spark Streaming

Spark Streaming Micro Butch

Discretized Stream (DStream)



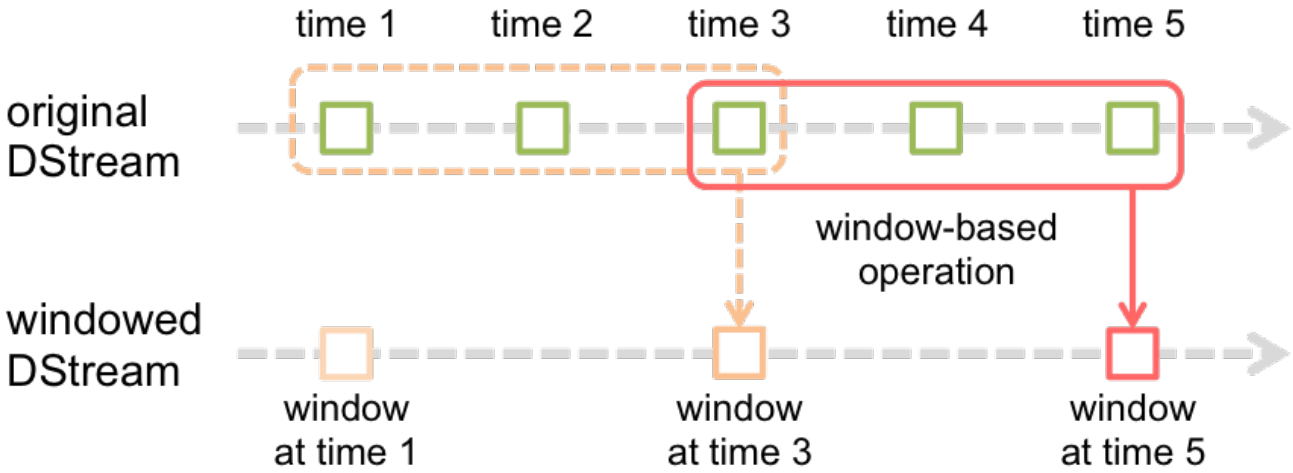
Пример

```
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: network_wordcount.py <hostname> <port>",
              file=sys.stderr)
        exit(-1)
    sc = SparkContext(appName="PythonStreamingNetworkWordCount")
    ssc = StreamingContext(sc, 1)

    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)
    counts.pprint()

    ssc.start()
    ssc.awaitTermination()
```

Пример

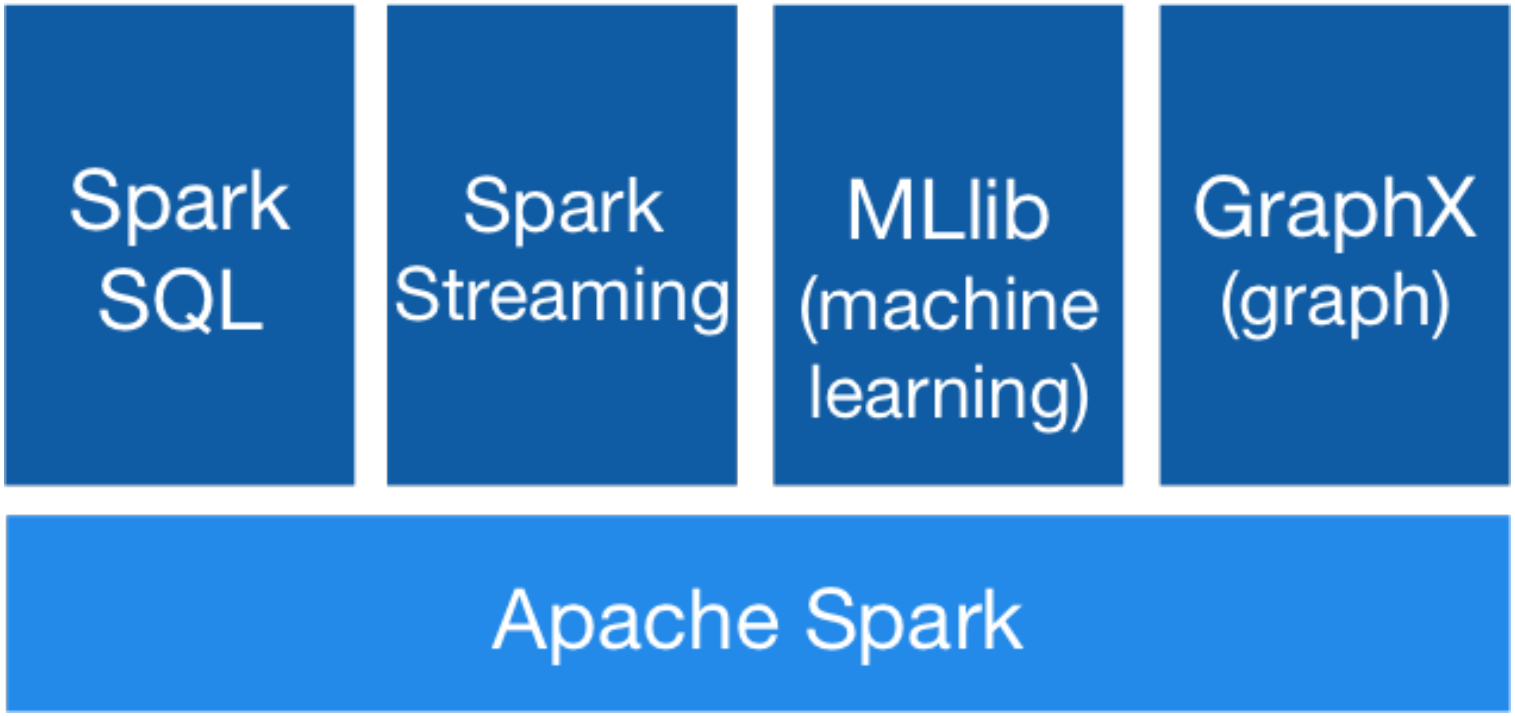
```
sc = SparkContext(appName="PythonStreamingStatefulNetworkWordCount")
ssc = StreamingContext(sc, 1)
ssc.checkpoint("checkpoint")

# RDD with initial state (key, value) pairs
initialStateRDD = sc.parallelize([(u'hello', 1), (u'world', 1)])

def updateFunc(new_values, last_sum):
    return sum(new_values) + (last_sum or 0)

lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
running_counts = lines.flatMap(lambda line: line.split(" "))\
                      .map(lambda word: (word, 1))\
                      .updateStateByKey(updateFunc,
                                        initialRDD=initialStateRDD)

running_counts.pprint()

ssc.start()
ssc.awaitTermination()
```

Скользящее окно




```
# Reduce last 30 seconds of data, every 10 seconds
windowedWordCounts = pairs.reduceByKeyAndWindow(
    lambda x, y: x + y, lambda x, y: x - y, 30, 10
)
```

Spark


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark