

Анализ больших данных с Apache Spark

Лекция 5. Spark Streaming

Мурашкин Вячеслав
2017

<https://github.com/a4tunado/lectures-hse-spark/tree/master/005>

Лекция 5. Spark Streaming

- Задачи
- StreamingContext
- Источники данных
- DStream
- Оконные операции
- Сохранение данных
- Пример: подсчет частоты слов в потоке
- Пример: анализ потока твитов

Задачи

- **Примеры источников данных:**
 - логи веб-сервисов
 - банковские транзакции
 - IoT - данные с различных сенсоров
- **Задачи:**
 - предобработка сырых данных перед сохранением в БД (ETL)
 - аналитика потока в реальном времени
 - поиск аномалий в реальном времени

StreamingContext

- Задаёт входные источники данных
- Определяет преобразование над потоковыми данными
- Запускает процесс чтения данных из источника
- Останавливает чтение данных из источника

StreamingContext

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

sc = SparkContext(master, appName)
ssc = StreamingContext(sc, 1)
```

ИСТОЧНИКИ

- **File Streams**
 - `streamingContext.textFileStream(dataDirectory)`
- **TCP Socket Streams**
 - `streamingContext.socketTextStream(...)`
- **Queue of RDDs as a Stream**
 - `streamingContext.queueStream(queueOfRDDs)`
- **Advanced Sources**
 - Kafka
 - Flume
 - Kinesis

<https://spark.apache.org/docs/latest/streaming-programming-guide.html#basic-sources>

Discretized Streams (DStreams)

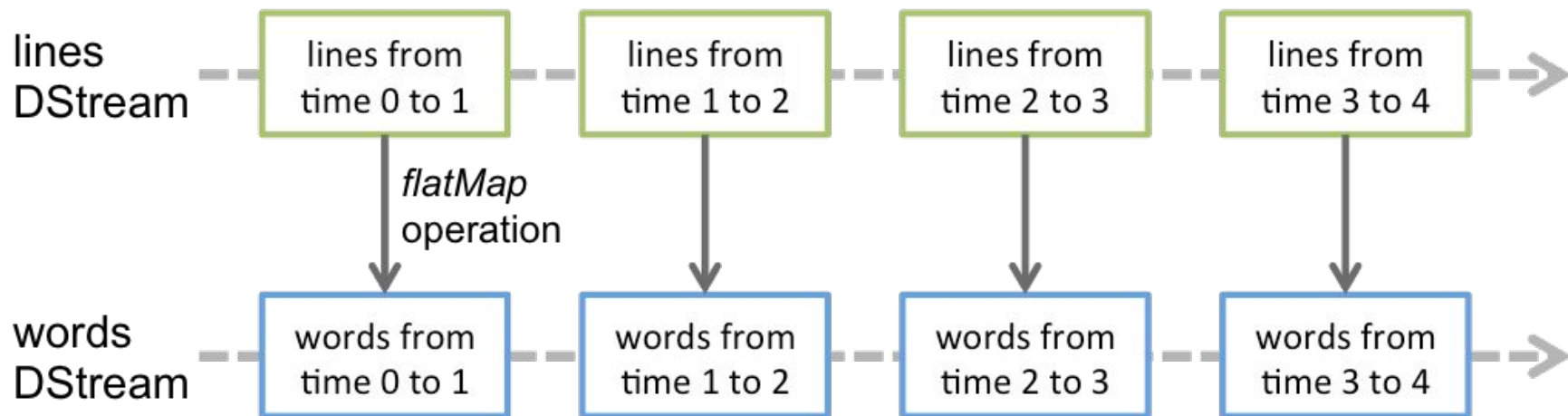


Преобразования (transformations)

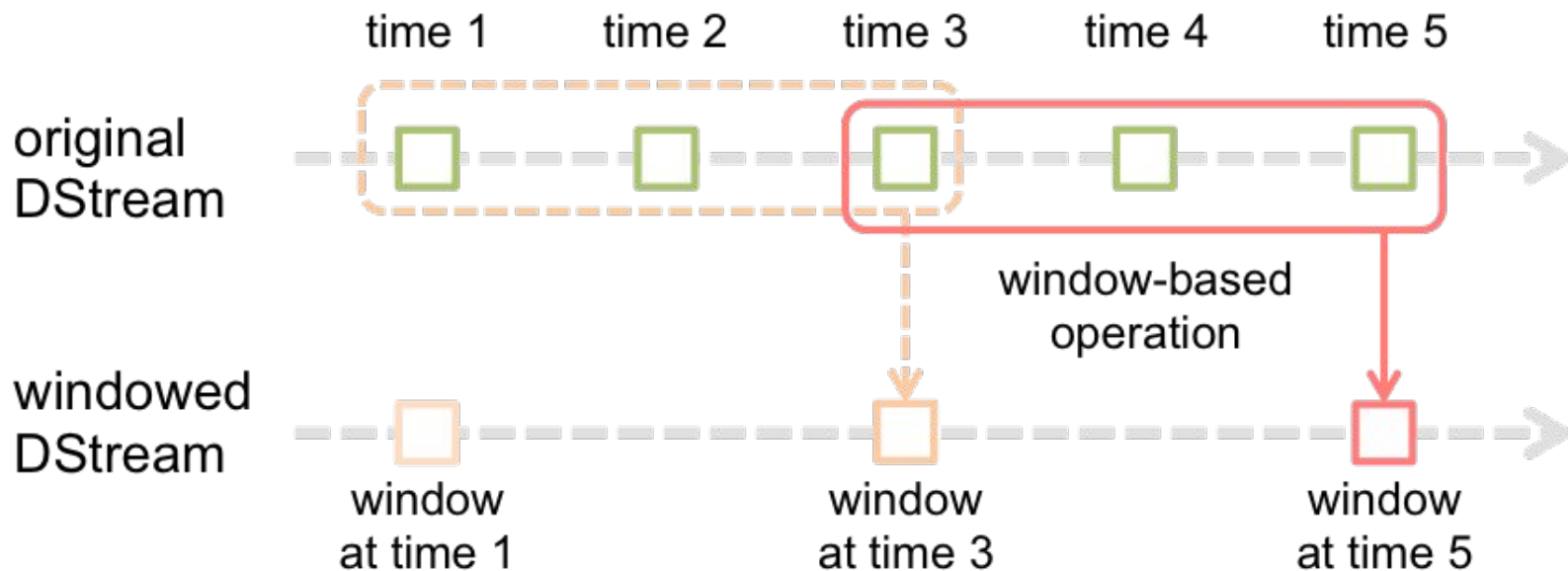
Функции преобразования аналогичны функциям преобразования RDD, возвращают новые объекты DStream

- **map(*func*)**
преобразует каждый элемент в потоке с помощью функции *func*
- **flatMap(*func*)**
аналогична **map**, но каждый элемент может быть преобразован в один или несколько объектов
- **transform(*func*)**
преобразует отдельно каждый RDD из потока
- **updateStateByKey(*func*)**
возвращает объект DStream с обновленным состоянием с учетом предыдущих значений

Преобразования (transformations)



Оконные операции



Оконные операции

Функции возвращают новый объект DStream

- **window**(*windowLength*, *slideInterval*)
 - *window length* - размер окна, в секундах
 - *sliding interval* - частота применения операций, в секундах
- **countByWindow**(*windowLength*, *slideInterval*)
подсчет объектов в потоке в заданном окне
- **reduceByWindow**(*func*, *windowLength*, *slideInterval*)
агрегация элементов из потока с помощью функции *func*

Сохранение данных

- **pprint(*num*)**
выводит в stdout первые *num* элементов чанка
- **saveAsTextFiles(*prefix*, [*suffix*])**
сохраняет каждый RDD в директорию *prefix* в текстовом формате, добавляет *suffix* к именам файлов
- **foreachRDD(*func*)**
позволяет кастомизировать процесс сохранения данных RDD

DataFrames

```
words = ... # DStream of strings

def process(time, rdd):
    print("===== %s =====" % str(time))
    try:
        # Get the singleton instance of SparkSession
        spark = getSparkSessionInstance(rdd.context.getConf())

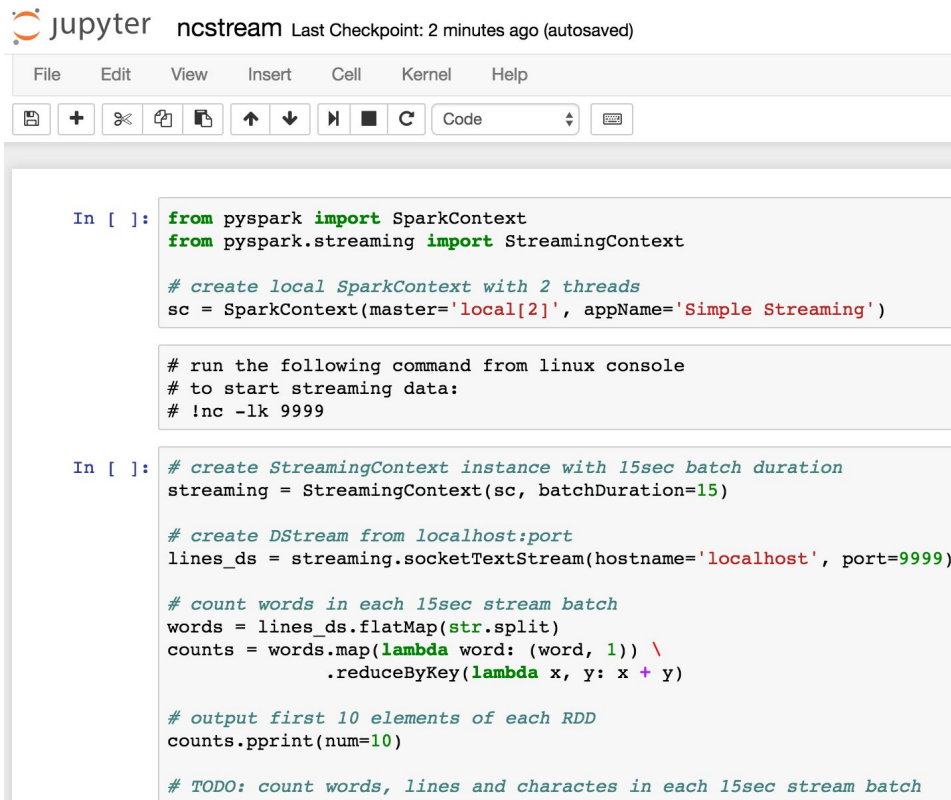
        # Convert RDD[String] to RDD[Row] to DataFrame
        rowRdd = rdd.map(lambda w: Row(word=w))
        wordsDataFrame = spark.createDataFrame(rowRdd)

        # Creates a temporary view using the DataFrame
        wordsDataFrame.createOrReplaceTempView("words")

        # Do word count on table using SQL and print it
        wordCountsDataFrame = spark.sql("select word, count(*) as total from words group by word")
        wordCountsDataFrame.show()
    except:
        pass

words.foreachRDD(process)
```

Пример: подсчет частоты слов в потоке



The image shows a Jupyter Notebook interface with the title 'ncstream' and a status bar indicating 'Last Checkpoint: 2 minutes ago (autosaved)'. The notebook has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. Below the menu is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, and navigation. The main area contains two code cells. The first cell is a code input with the following text:

```
In [ ]: from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# create local SparkContext with 2 threads
sc = SparkContext(master='local[2]', appName='Simple Streaming')

# run the following command from linux console
# to start streaming data:
# !nc -lk 9999
```

 The second cell is also a code input with the following text:

```
In [ ]: # create StreamingContext instance with 15sec batch duration
streaming = StreamingContext(sc, batchDuration=15)

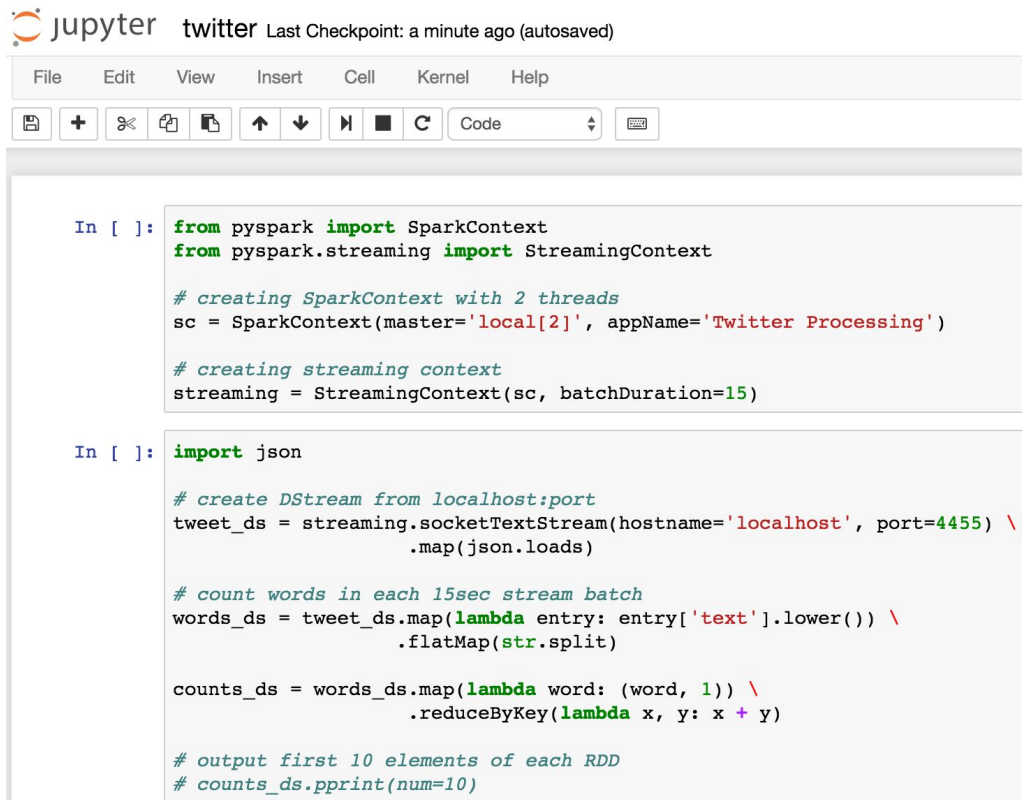
# create DStream from localhost:port
lines_ds = streaming.socketTextStream(hostname='localhost', port=9999)

# count words in each 15sec stream batch
words = lines_ds.flatMap(str.split)
counts = words.map(lambda word: (word, 1)) \
    .reduceByKey(lambda x, y: x + y)

# output first 10 elements of each RDD
counts.pprint(num=10)

# TODO: count words, lines and charactes in each 15sec stream batch
```

Пример: анализ потока ТВИТОВ



The image shows a Jupyter Notebook interface with a light gray header bar. On the left is the Jupyter logo, followed by the text "jupyter" and "twitter" in a larger font, and "Last Checkpoint: a minute ago (autosaved)" in a smaller font. Below the header is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Under the menu bar is a toolbar with icons for saving, adding cells, deleting, copying, pasting, undo, redo, and a dropdown menu currently set to "Code". The main area of the notebook contains two code cells. The first cell starts with "In []:" and contains code to create a SparkContext and a StreamingContext. The second cell starts with "In []:" and contains code to create a DStream from a socket, process the tweets by counting words, and output the results.

```
In [ ]: from pyspark import SparkContext
        from pyspark.streaming import StreamingContext

        # creating SparkContext with 2 threads
        sc = SparkContext(master='local[2]', appName='Twitter Processing')

        # creating streaming context
        streaming = StreamingContext(sc, batchDuration=15)

In [ ]: import json

        # create DStream from localhost:port
        tweet_ds = streaming.socketTextStream(hostname='localhost', port=4455) \
            .map(json.loads)

        # count words in each 15sec stream batch
        words_ds = tweet_ds.map(lambda entry: entry['text'].lower()) \
            .flatMap(str.split)

        counts_ds = words_ds.map(lambda word: (word, 1)) \
            .reduceByKey(lambda x, y: x + y)

        # output first 10 elements of each RDD
        # counts_ds.pprint(num=10)
```

Полезные материалы

- Spark Streaming Programming Guide
<http://spark.apache.org/docs/latest/streaming-programming-guide.html>
- pyspark.streaming module
<http://spark.apache.org/docs/latest/api/python/pyspark.streaming.html>
- Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing
<https://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-259.pdf>