ТЕХНОСФЕРА

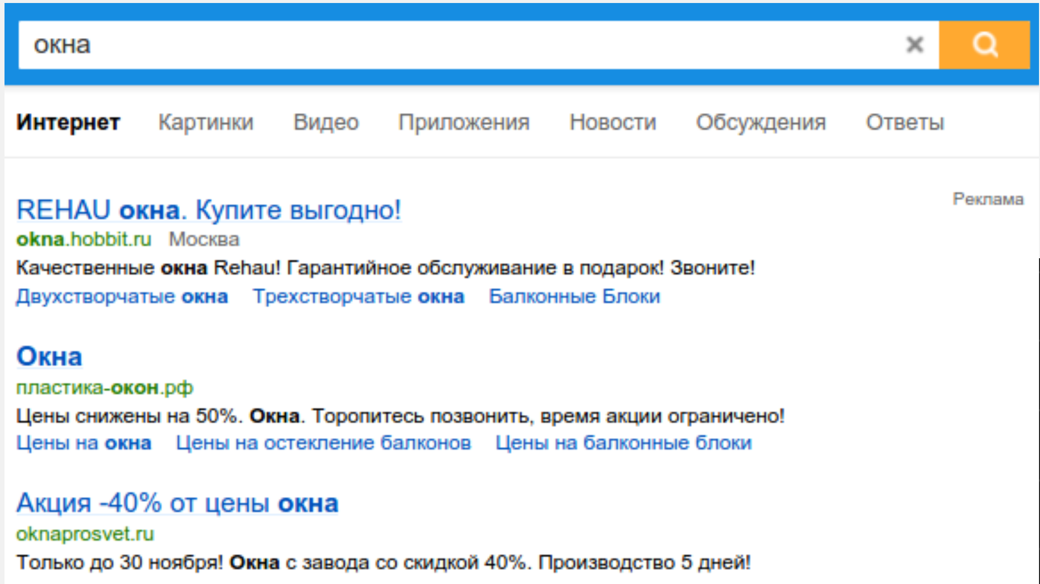
Лекция 10

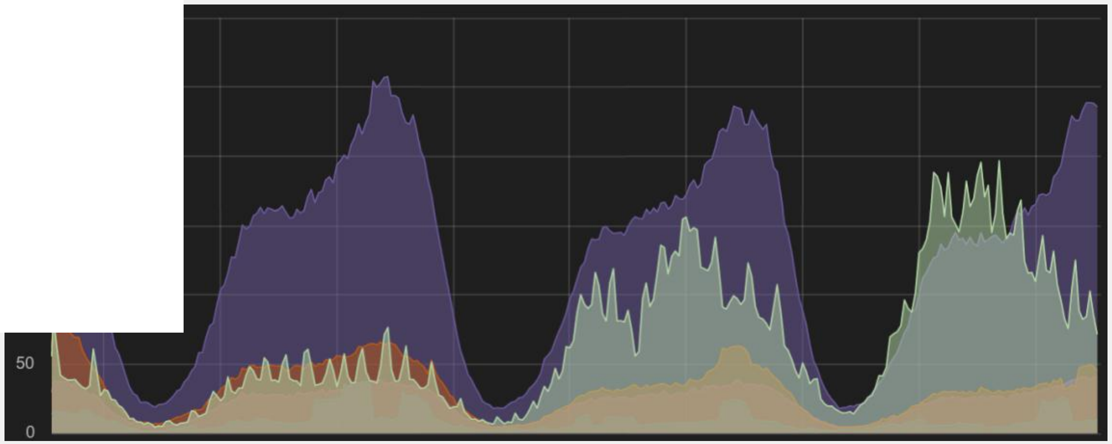
Spark Streaming. MLLib.

Клюкин Денис

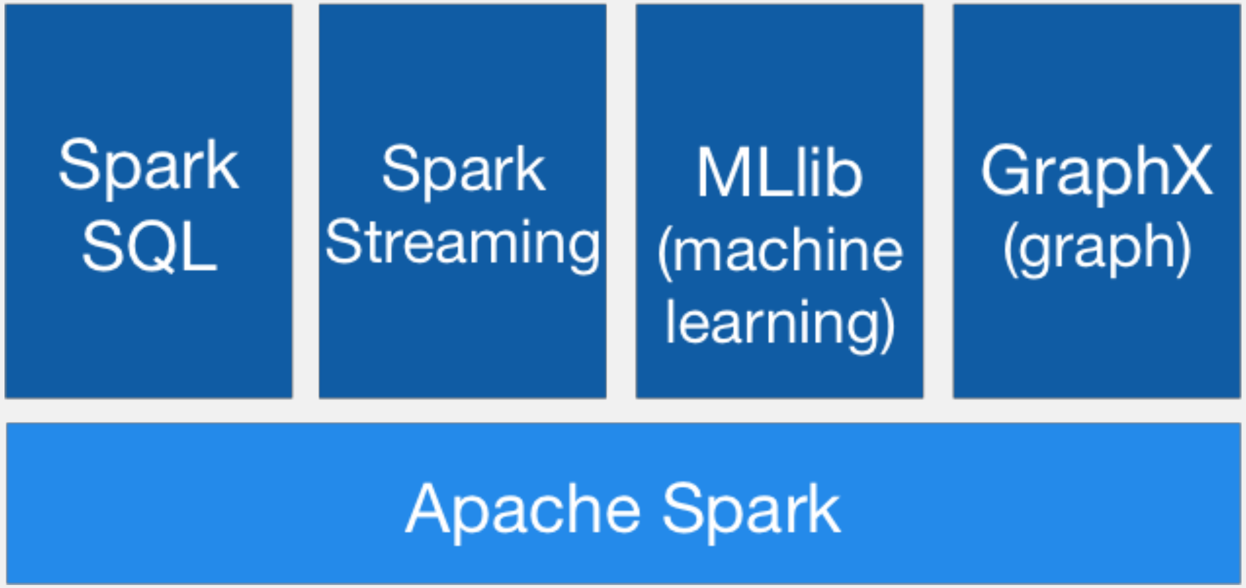
Batch vs Stream processing

- MapReduce и Spark хорошо подходят для пакетной обработки данных.
- Иногда бывает потребность обрабатывать бесконечные потоки данных.



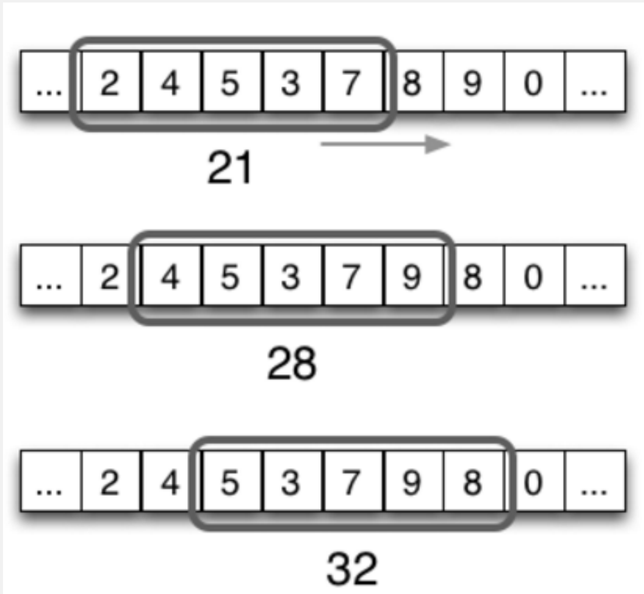


Apache Spark

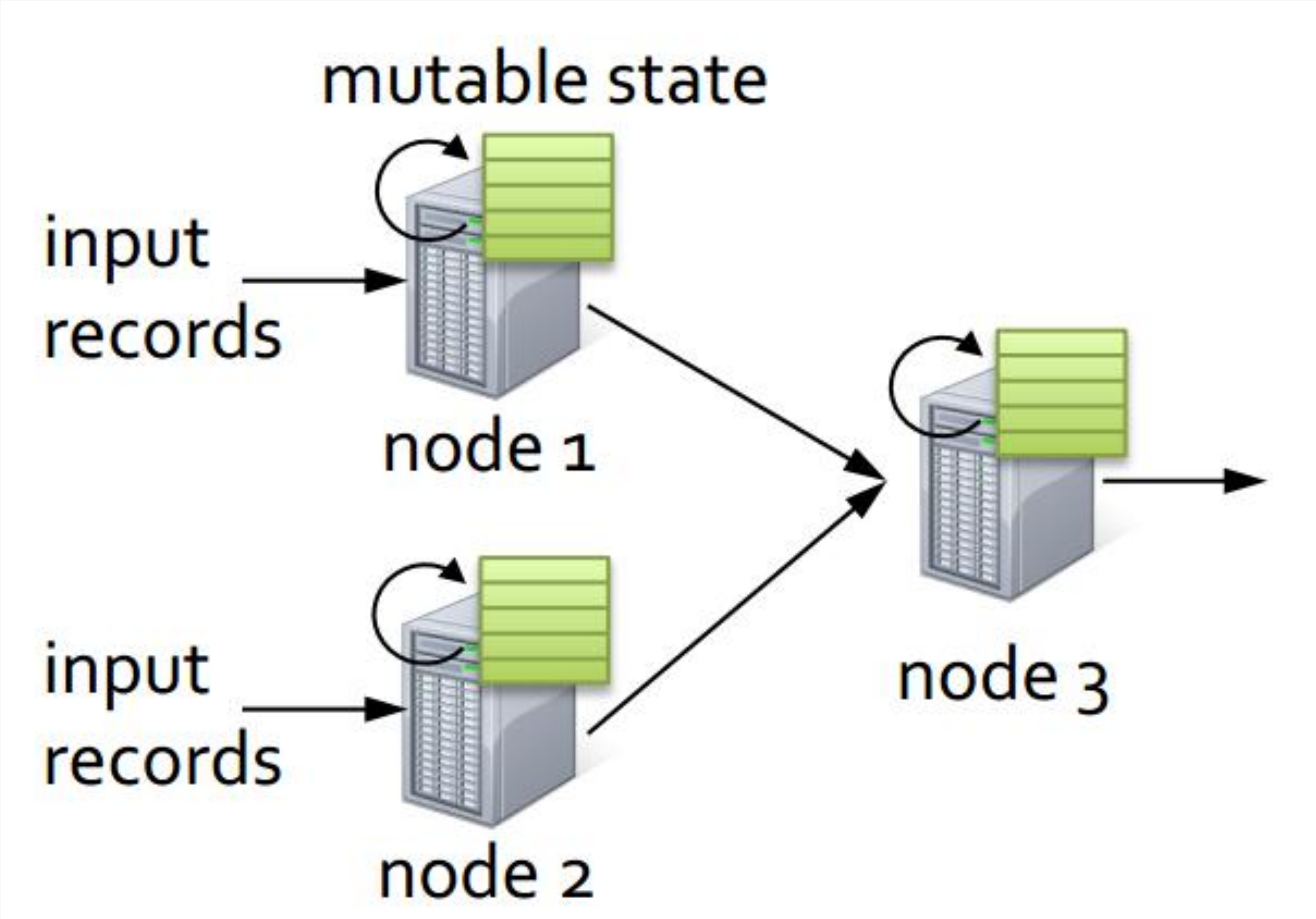


Stream processing используя batch

- Можно использовать метод скользящего окна
- Hadoop MapReduce плохо подходит для real-time

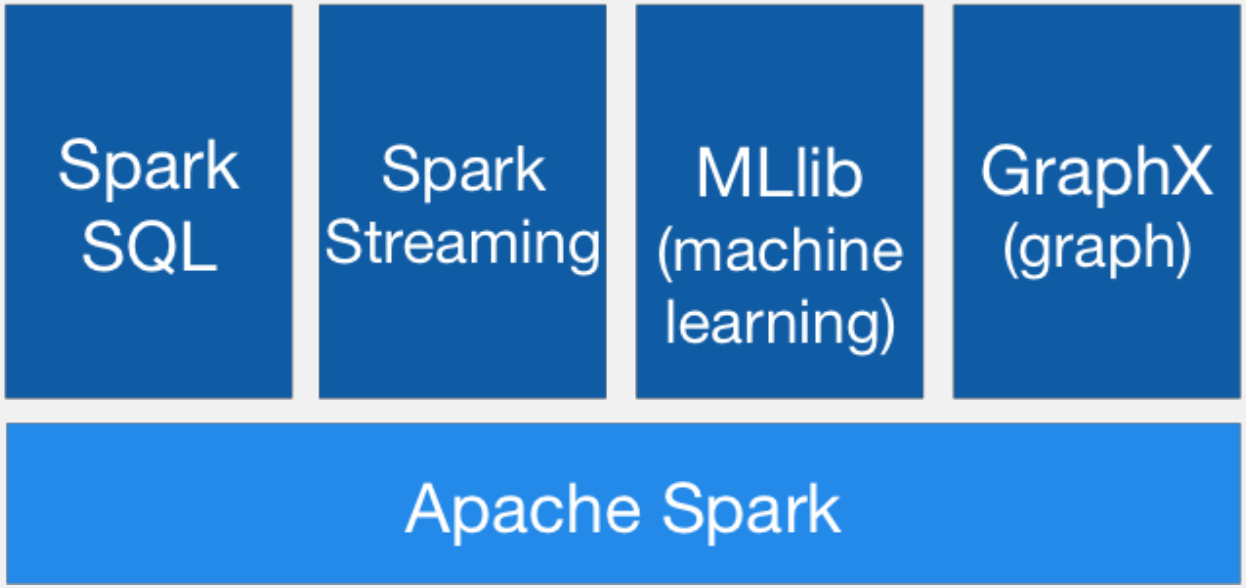


Stateful processing



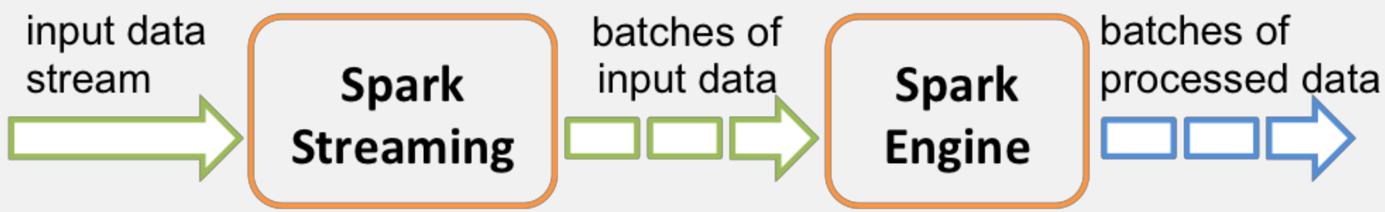
Spark Streaming

- Масштабируемый
- Подходит для real-time (порядка 1 сек)
- Встроенная fault tolerance
- Можно комбинировать batch & streaming



Mini-batch модель

- Делим входной поток на фрагменты (1 сек)
- Рассматриваем каждый фрагмент как RDD
- Выход – поток обработанных RDD



- Используем общий код для обработки данных
- Легко организовать fault tolerance
- Можно комбинировать с batch обработкой

Discretized Streams (DStreams)

2 способа создать DStream:

1. Из входного источкника
2. Применяя трансформации к другим DStream

Receiver

- *Базовые источники:* файловая система, сокеты

```
streamingContext.textFileStream(dataDirectory)
```

- *Расширенные источники*: Kafka, Flume, Twitter...

```
KafkaUtils.createStream(ssc, zk, “name", {topic: 1})
```

- Кастомные источники

Важно! Один приёмник расходует 1 слот!

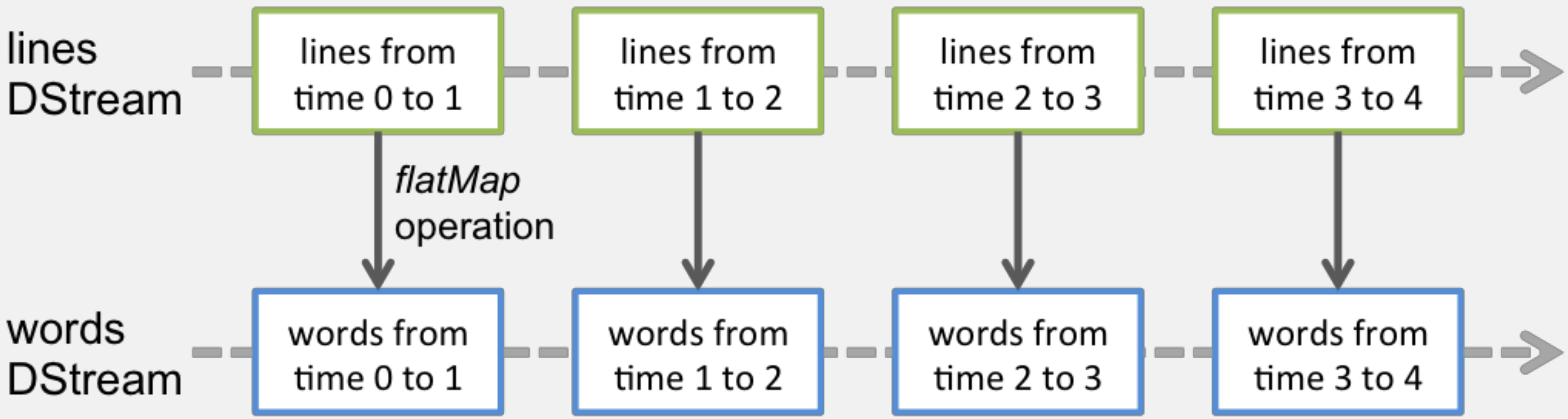
Кастомный Receiver

```
class MyReceiver(storageLevel: StorageLevel) extends NetworkReceiver[String](storageLevel) {
def onStart() {
    // Setup stuff (start threads, open sockets, etc.) to start receiving data.
    // Must start new thread to receive data, as onStart() must be non-blocking.

    // Call store(...) in those threads to store received data into Spark's memory.

    // Call stop(...), restart(...) or reportError(...) on any thread based on how
    // different errors needs to be handled.

    // See corresponding method documentation for more details
}

def onStop() {
    // Cleanup stuff (stop threads, close sockets, etc.) to stop receiving data.
}
}
```

- Компактное API
- Бывают 2х видов: надежные и ненадежные

Transformations

1. Классические spark преобразования (map, reduceByKey, и т.д.) Применяются к каждому RDD индивидуально.



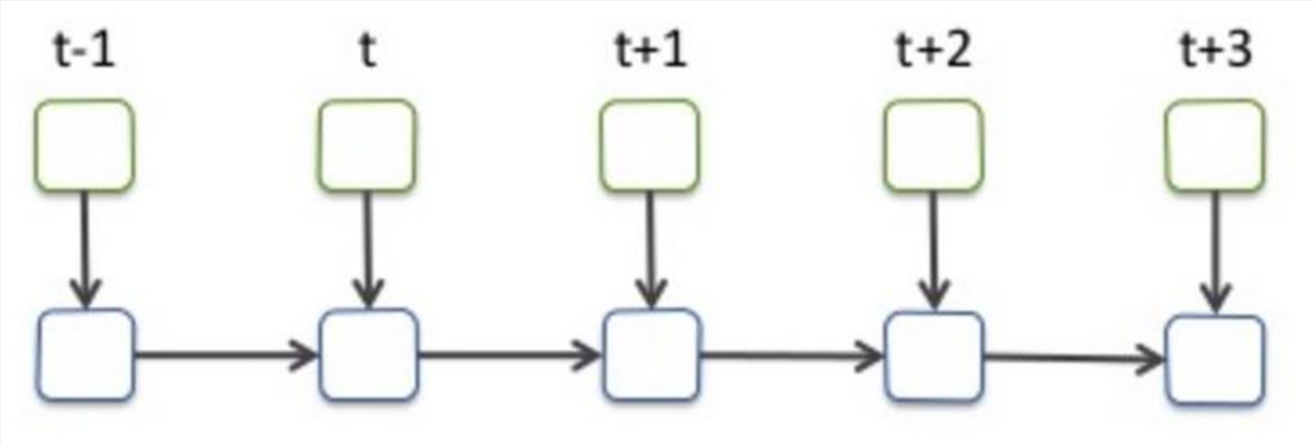
2. Специальные операции
   - Скользящее окно
   - Обновление состояния

UpdateStateByKey

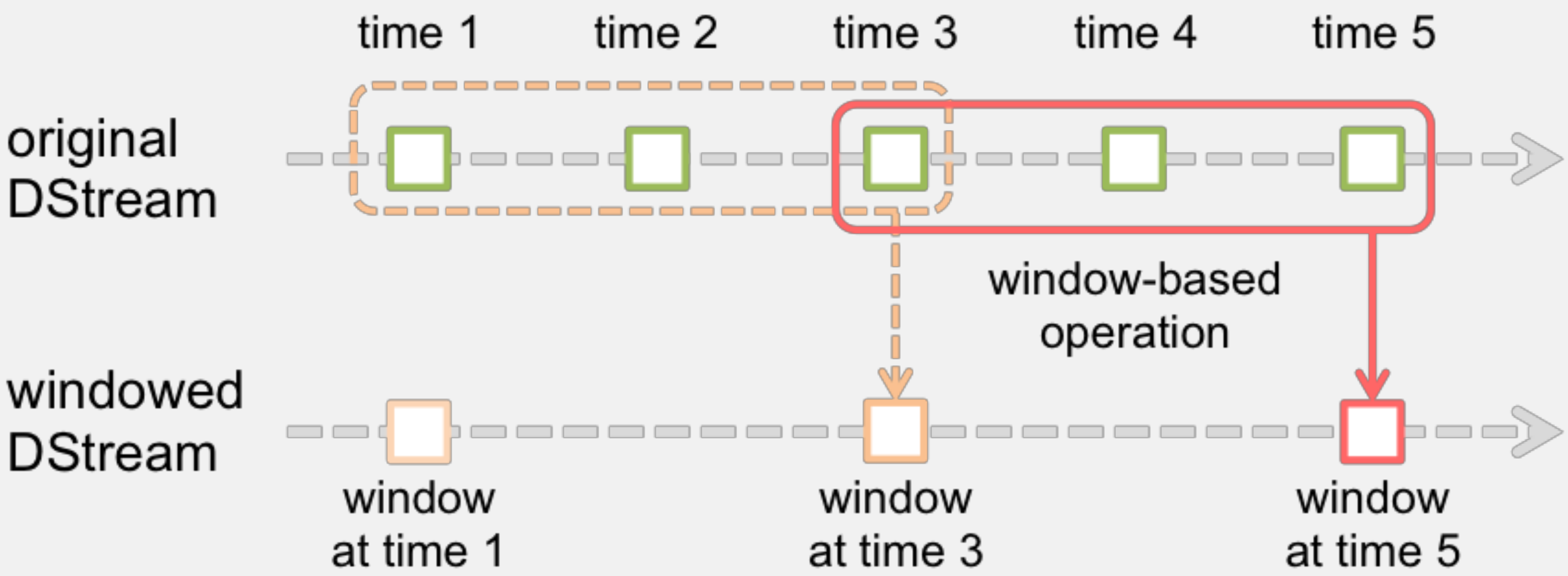
- Возможность работать с состояниями

```
def updateFunction(newValues, runningCount):
    if runningCount is None:
        runningCount = 0
    return sum(newValues, runningCount)
```

```
runningCounts = pairs.updateStateByKey(updateFunction)
```



Скользящее окно



Скользящее окно

- `window(windowLength, slideInterval)`
- `countByWindow(windowLength, slideInterval)`
- `reduceByWindow(func, windowLength, slideInterval)`
- `reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])`
- `reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])`
- `countByValueAndWindow(windowLength, slideInterval, [numTasks])`

Сохранение данных

- print()
- saveAsTextFiles(prefix, [suffix])
- saveAsObjectFiles(prefix, [suffix])
- saveAsHadoopFiles(prefix, [suffix])
- foreachRDD(func)
  - Не путать с transform(func)

Без сохранения данных spark streaming не запустит обработку данных!

foreachRDD

```python
def sendRecord(rdd):
    connection = createNewConnection()  # executed at the driver
    rdd.foreach(lambda record: connection.send(record))
    connection.close()

dstream.foreachRDD(sendRecord)
```

foreachRDD

```python
def sendRecord(record):
    connection = createNewConnection()
    connection.send(record)
    connection.close()

dstream.foreachRDD(lambda rdd: rdd.foreach(sendRecord))
```

foreachRDD

```python
def sendPartition(iter):
    connection = createNewConnection()
    for record in iter:
        connection.send(record)
    connection.close()

dstream.foreachRDD(lambda rdd: rdd.foreachPartition(sendPartition))
```

foreachRDD

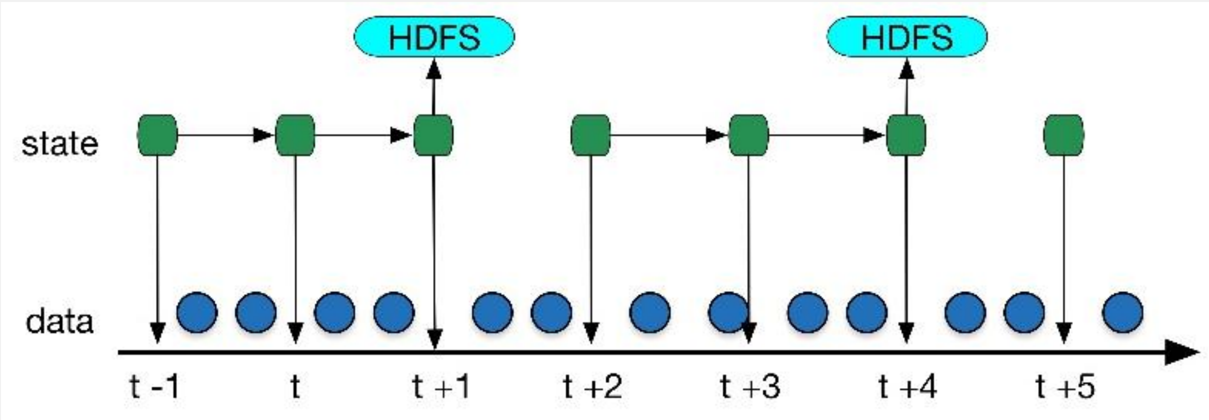
```python
def sendPartition(iter):
    # ConnectionPool is a static, lazily initialized pool of connections
    connection = ConnectionPool.getConnection()
    for record in iter:
        connection.send(record)
    # return to the pool for future reuse
    ConnectionPool.returnConnection(connection)

dstream.foreachRDD(lambda rdd: rdd.foreachPartition(sendPartition))
```

Контрольные точки

- Метаданные
- Некоторые RDD

Когда применять

- updateStateByKey или reduceByKeyAndWindow (with inverse function)
- Работа 24/7



Контрольные точки

```python
# Function to create and setup a new StreamingContext
def functionToCreateContext():
    sc = SparkContext(...)   # new context
    ssc = new StreamingContext(...)
    lines = ssc.socketTextStream(...) # create DStreams
    ...
    ssc.checkpoint(checkpointDirectory)   # set checkpoint directory
    return ssc

# Get StreamingContext from checkpoint data or create a new one
context = StreamingContext.getOrCreate(checkpointDirectory, functionToCreateContext)

# Do additional setup on context that needs to be done,
# irrespective of whether it is being started or restarted
context. ...

# Start the context
context.start()
context.awaitTermination()
```

Data types

Data types

- Vectors

```python
from pyspark.mllib.linalg import Vectors

# Use a NumPy array as a dense vector.
dv1 = np.array([1.0, 0.0, 3.0])
# Use a Python list as a dense vector.
dv2 = [1.0, 0.0, 3.0]
# Create a SparseVector.
sv1 = Vectors.sparse(3, [0, 2], [1.0, 3.0])
```

- Labled point

```python
# Create a labeled point with a positive label and a dense feature vector.
pos = LabeledPoint(1.0, [1.0, 0.0, 3.0])
```

Data types

- Local matrix

```python
from pyspark.mllib.linalg import Matrix, Matrices

# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
dm2 = Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])

# Create a sparse matrix ((9.0, 0.0), (0.0, 8.0), (0.0, 6.0))
sm = Matrices.sparse(3, 2, [0, 1, 3], [0, 2, 1], [9, 6, 8])
```

Data types

- RowMatrix

```
# Create an RDD of vectors.
rows = sc.parallelize([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

# Create a RowMatrix from an RDD of vectors.
mat = RowMatrix(rows)
```

Data types

- CoordinateMatrix
  - Each entry is (i: Long, j: Long, value: Double)

```
entries = sc.parallelize([MatrixEntry(0, 0, 1.2), MatrixEntry(1, 0, 2.1)
#   - or using (long, long, float) tuples:
entries = sc.parallelize([(0, 0, 1.2), (1, 0, 2.1), (2, 1, 3.7)])

# Create an CoordinateMatrix from an RDD of MatrixEntries.
mat = CoordinateMatrix(entries)
```

Data types

- BlockMatrix
  - Each entry is (i: Long, j: Long, value: Double)

```
# Create an RDD of sub-matrix blocks.
blocks = sc.parallelize([((0, 0), Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])),
                         ((1, 0), Matrices.dense(3, 2, [7, 8, 9, 10, 11, 12]))])

# Create a BlockMatrix from an RDD of sub-matrix blocks.
mat = BlockMatrix(blocks, 3, 2)
```

MLLib

- Classification: logistic regression, naive Bayes,...
- Regression: generalized linear regression, isotonic regression,...
- Decision trees, random forests, and gradient-boosted trees
- Recommendation: alternating least squares (ALS)
- Clustering: K-means, Gaussian mixtures (GMMs),...
- Topic modeling: latent Dirichlet allocation (LDA)
- Feature transformations: standardization, normalization, hashing,...
- Model evaluation and hyper-parameter tuning
- Frequent itemset and sequential pattern mining: FP-growth, association rules, PrefixSpan
- Distributed linear algebra: singular value decomposition (SVD), principal component analysis (PCA),...
- Statistics: summary statistics, hypothesis testing,...

Classification

```
from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model
model = SVMWithSGD.train(parsedData, iterations=100)
```

Classification

```python
# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))

# Save and load model
model.save(sc, "target/tmp/pythonSVMWithSGDModel")
sameModel = SVMModel.load(sc, "target/tmp/pythonSVMWithSGDModel")
```

Collaborative filtering

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Load and parse the data
data = sc.textFile("data/mllib/als/test.data")
ratings = data.map(lambda l: l.split(','))\
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

# Build the recommendation model using Alternating Least Squares
rank = 10
numIterations = 10
model = ALS.train(ratings, rank, numIterations)
```

Collaborative filtering

```
# Evaluate the model on training data
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error = " + str(MSE))

# Save and load model
model.save(sc, "target/tmp/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, "target/tmp/myCollaborativeFilter")
```

Семинар

Реализуем алгоритм PageRank

$$PR(p_i; t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)},$$

Данные: LiveJournal social network

Описание https://goo.gl/ghdIrU

Скачать https://goo.gl/PUhTZb

Хадуп /data/voropaev/soc-LiveJournal1.txt

ДЗ

Сравниваем скорость работы алгоритма PageRank на spark и mapreduce.

Отчет:

- скорость работы двух реализаций алгоритма PageRank.
- Top-10 вершин графа
- Исходные коды

d.klyukin@corp.mail.ru

ТЕХНОСФЕРА

Спасибо за внимание!

Клюкин Денис

d.klyukin@corp.mail.ru

Не забудьте отметиться.