

Промышленное машинное обучение на Spark



Содержание курса

01

Оборачиваем модель в сервис. Первая часть: Docker. Flask

02

Оборачиваем модель в сервис. Вторая часть: Requests. REST API

03

Распределенные вычисления. HDFS. MapReduce. **Spark DataFrame**

04

Погружение в среду Spark. RDD, SQL, Pandas API.

05

Генерация признаков. Spark feature engineering

06

Распределенное обучение моделей. Spark ML

07

Обработка и хранение текстовых данных и картинок.
Spark image processing. Spark NLP.

08

Обработка потоковых данных. Spark Streaming.

3. Распределенные вычисления.

Spark DataFrame

План:

I. Spark

- Spark — основные абстракции и объекты.
- Действия и преобразования.
- Data locality. Сериализация

Spark - основные абстракции и объекты.

Основные концепции Spark

- RDD
- SparkContext (sc)
- Directed acyclic graphs (DAG)
- Actions and transformations
- Spark DataFrames

Resilient distributed datasets (RDD).

RDD (Resilient Distributed Dataset) — это фундаментальная структура данных Apache Spark, которая представляет собой неизменяемую коллекцию объектов, которые вычисляются на разных узлах кластера. Каждый набор данных в Spark RDD логически разделен на множество серверов, чтобы их можно было вычислить на разных узлах кластера.

SparkContext (sc)

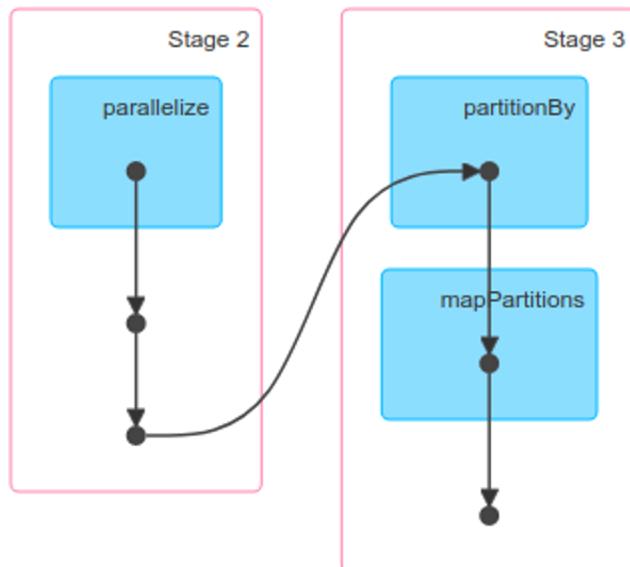
SparkContext — это точка входа для всех операций Spark и средство, с помощью которого приложение подключается к ресурсам кластера Spark. Он инициализирует экземпляр Spark и впоследствии может использоваться для выполнения действий, преобразований над RDD, а также извлечения данных и других функций Spark.

SparkContext также инициализирует различные свойства процесса, такие как имя приложения, количество ядер, параметры использования памяти и другие характеристики. В совокупности эти свойства содержатся в объекте SparkConf, который передается в SparkContext в качестве параметра.

Directed Acyclic Graph (DAG).

Пример с прошлой лекции подсчет слов

```
wordCountsCollected = (sparkdata
    .map(mapword)
    .reduceByKey(lambda a,b: a+b)
    .collect())
```



DAG (Направленный ациклический граф)

— это набор вершин и ребер, где вершины представляют RDD, а ребра представляют операцию, которая будет применяться к RDD. В Spark DAG каждое ребро направляет от более раннего к более позднему в последовательности. При вызове действия созданный DAG отправляется планировщику DAG, который далее разбивает граф на этапы и задачи.

Spark DataFrame.

Spark DataFrame — это набор данных, организованный в именованные столбцы. Концептуально он эквивалентен таблице в реляционной базе данных или фрейму данных в R / Python, но с более обширной внутренней оптимизацией. DataFrames могут быть созданы из широкого спектра источников, таких как файлы структурированных данных, таблицы в Hive, внешние базы данных или существующие RDD.

Transformations.

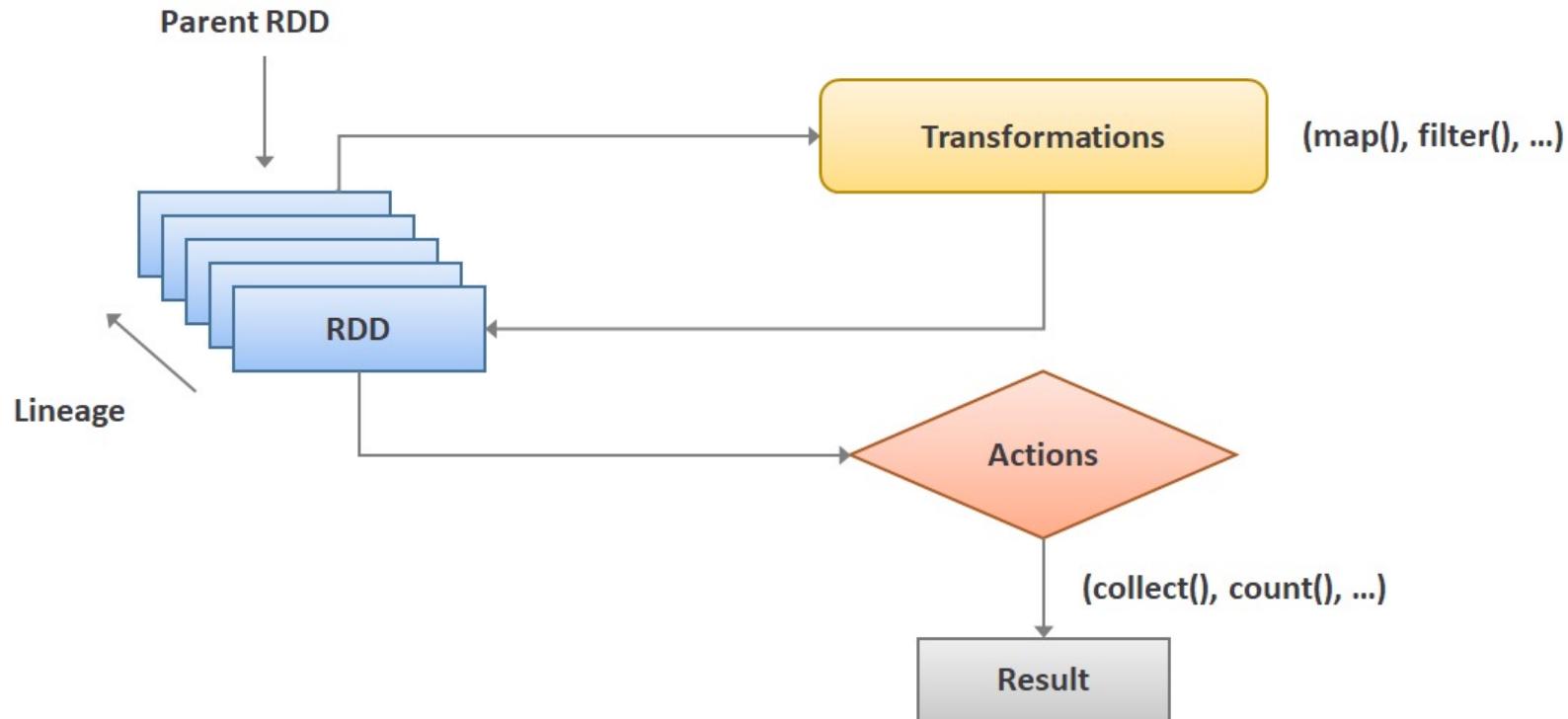
map(func)	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .	groupByKey([numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.
filter(func)	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.		 Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numPartitions argument to set a different number of tasks.
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.		
sortByKey([ascending], [numPartitions])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean <i>ascending</i> argument.	reduceByKey(func, [numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
repartition(numPartitions)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.		

Actions.

reduce(<i>func</i>)	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(<i>n</i>)	Return an array with the first <i>n</i> elements of the dataset.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.

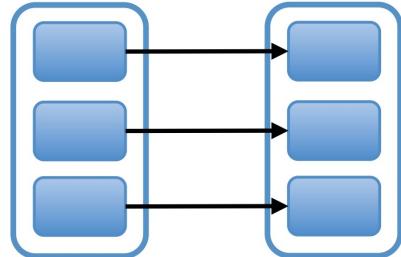


Spark RDD (Unstructured) Operations

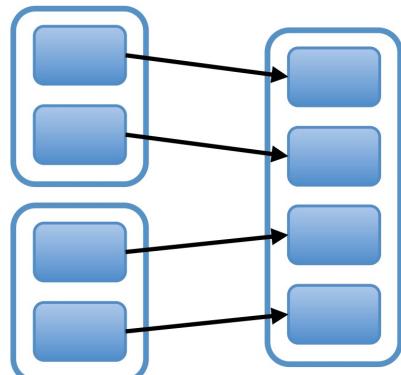


Типы операций. Narrow vs Wide Dependency

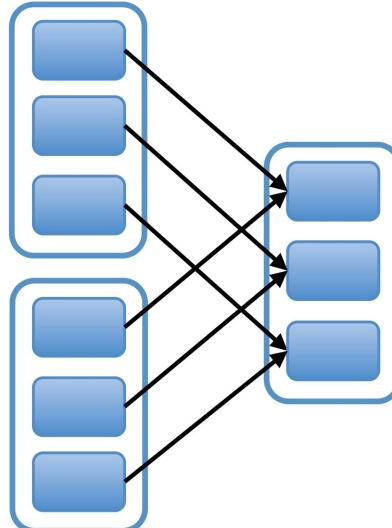
Narrow Dependencies:



map, filter

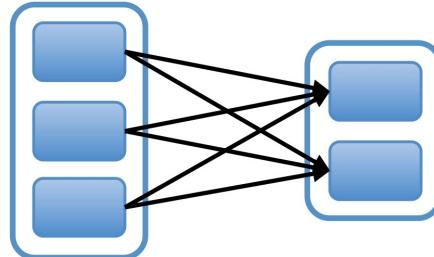


union

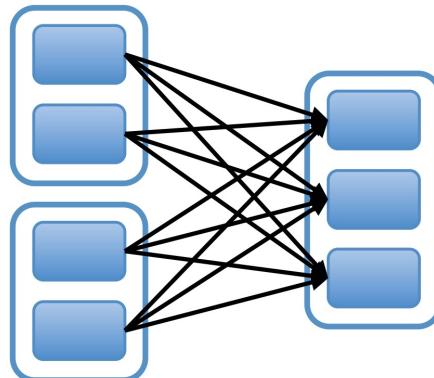


join with inputs
co-partitioned

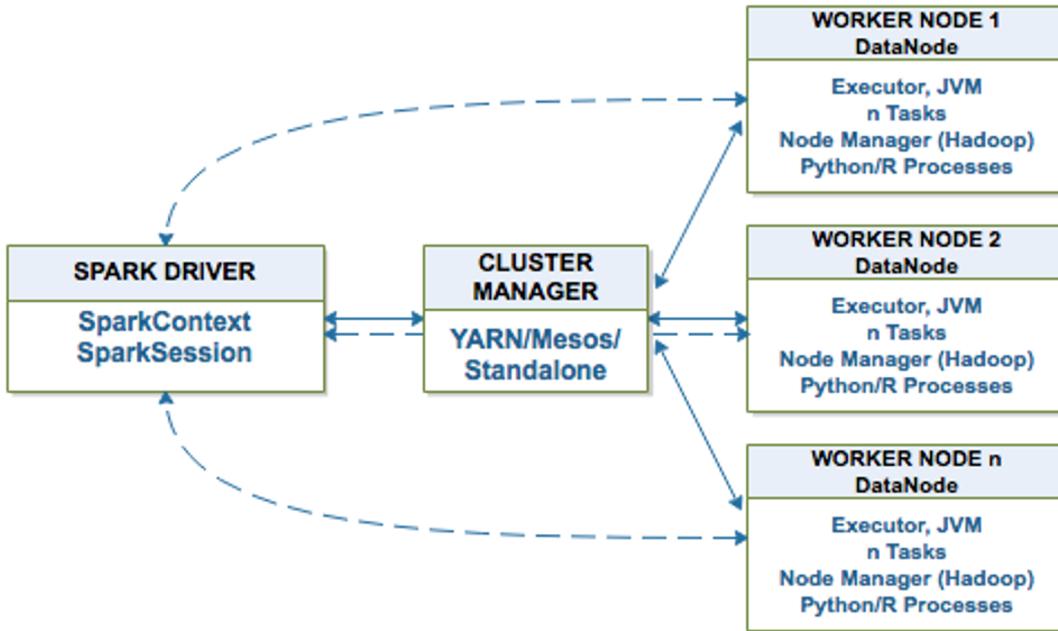
Wide Dependencies:



groupByKey



join with inputs not
co-partitioned



WorkerNode — это серверы, на которых размещены приложения Spark. Каждое приложение получает свой собственный уникальный процесс-исполнитель, а именно процессы, которые выполняют фактическое действие и задачи преобразования.

SparkDriver отправляет инструкции рабочим узлам для планирования задач.

Менеджеры кластеров координируют обмен данными между рабочими узлами, управляемыми узлами (такими как запуск, остановка)

Data locality. Как ее реализовать. Сериализация.
Пример.

Локальность данных — это процесс перемещения вычисления ближе к месту, где находятся фактические данные на узле, вместо перемещения больших данных в вычисления. Это минимизирует перегрузку сети и увеличивает общую пропускную способность системы.

- 1) Данные лежат на том же узле, что и mapper — good
- 2) Данные лежат на соседнем узле в стойке — not so good
- 3) Данные лежат в ноде на другой стойке — bad

Локальность данных обеспечивается сериализацией кода

Сериализация — процесс перевода какой-либо структуры данных в последовательность байтов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности.

Но не все функции сериализуются - например

```
def func(x):
    f = open('file.txt')
    return x in f.readlines()
```