# Конспект по теме “PySpark”

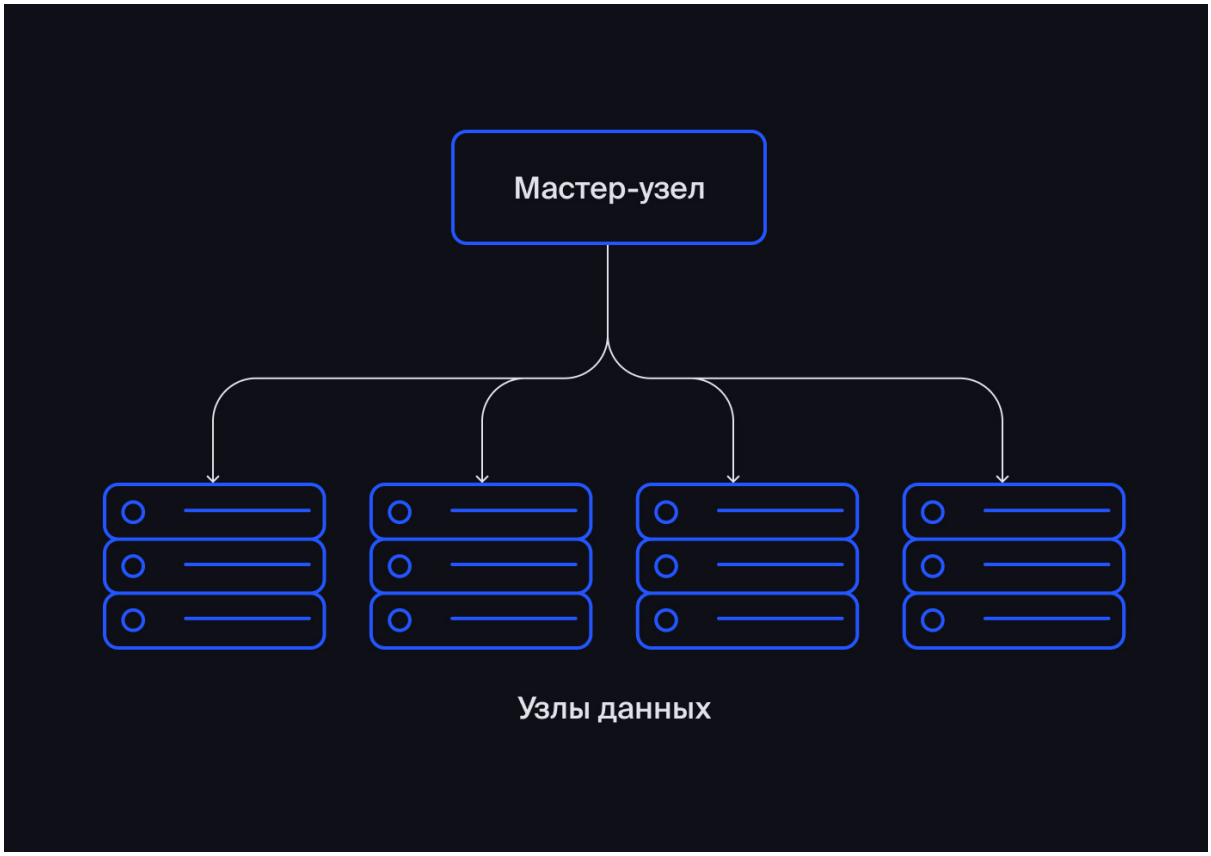
## Распределённые системы

Когда объём данных растёт, и один компьютер с вычислениями уже не справляется, подключают **распределённые системы** (англ. *Distributed File Systems*). Они хранят файлы с данными на нескольких компьютерах и предоставляют доступ к данным. Файл делится на фрагменты, причём каждый фрагмент может быть сохранён несколько раз на разных компьютерах. Так гарантируется целостность данных.

Распределённая система состоит из несколько **узлов** (англ. *nodes*). Это отдельные компьютеры с ресурсами вычисления и хранения данных.

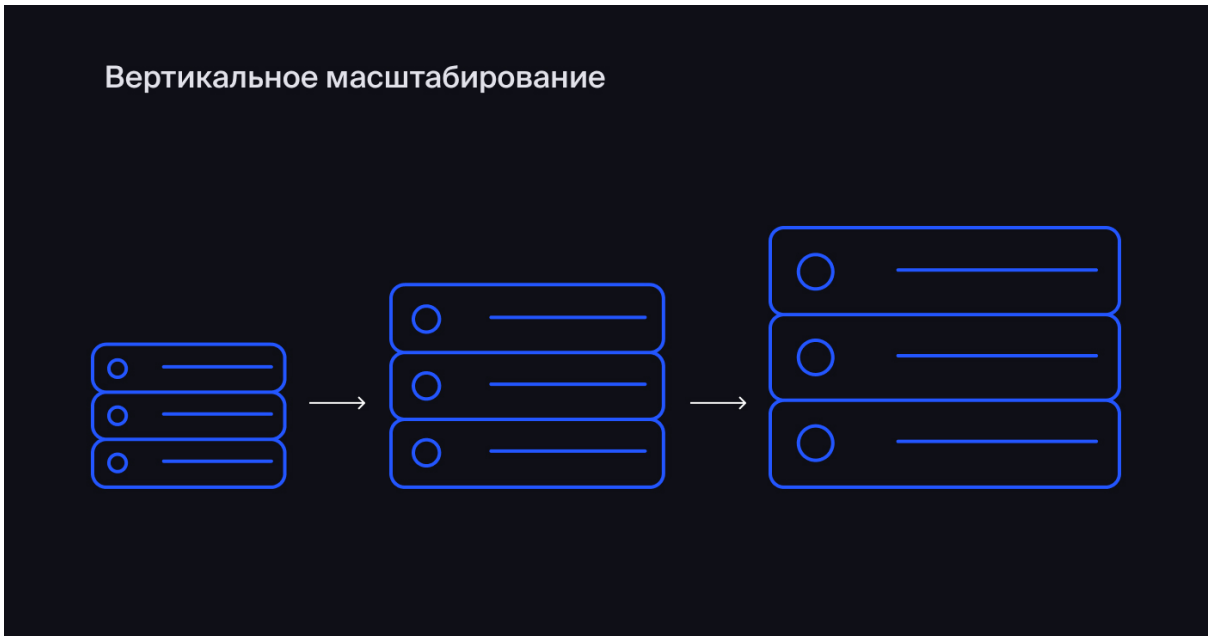
Узлы бывают двух типов:

- **Мастер-узел**, или **ведущий узел** (англ. *Name Node*). Он распределяет файлы между компьютерами в **кластере** (англ. *cluster*) — наборе связанных узлов.
- **Узлы данных** (англ. *Data Node*). В них данные содержатся и обрабатываются. Чтобы избежать потери информации, каждый файл дублируется в нескольких узлах данных.

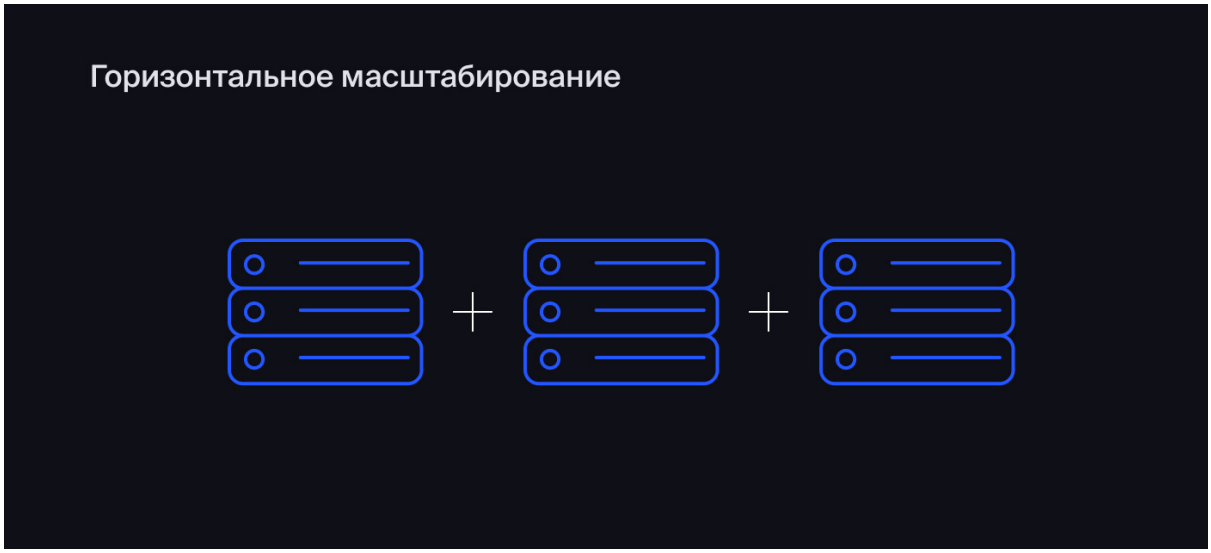


Чтобы компьютеры хранили и обрабатывали больше данных, их совершенствуют двумя способами:

- Повышение производительности каждого узла или замена более мощными узлами. Такой подход называется **вертикальным масштабированием** (англ. *scale up*).




- Увеличение количества узлов в кластере — это **горизонтальное масштабирование** (англ. *scale out*). Так узлы будут параллельно обрабатывать данные (англ. *parallel processing*, «параллельная обработка»). Этот подход помогает быстрее масштабироваться.



## Spark и RDD

*Apache Spark* — фреймворк для распределённых вычислений с открытым исходным кодом, который позволяет распределять и обрабатывать данные на нескольких компьютерах одновременно. Для работы в Python выпустили библиотеку **PySpark** (англ. «искра для Python»).

Отказоустойчивый распределённый набор данных (англ. *Resilient Distributed Dataset, RDD*) — тип структуры данных, который можно распределить между несколькими узлами в кластере. RDD — основной инструмент для преобразования данных и часть датафреймов.

Чтобы работать с RDD, импортируем из библиотеки *PySpark* объект **SparkContext**, который отвечает за операции с кластером в *Spark*, инициализируем объект *SparkContext* и передаём ему настройки. Это могут быть URL-адрес мастер-узла и название приложения:

```
from pyspark import SparkContext

# sc — от англ. spark context
# appName — от англ. application name, название приложения
sc = SparkContext(appName="IntroToSpark")
```

Вызовом функции **sc.parallelize()** (англ. «параллелизовать») можно перевести список в RDD:

```
pyspark_entry = sc.parallelize(['2009-01-01', 0, 0, 24])
```

Если распечатать RDD на экране, то получим **параллельную коллекцию RDD** (англ. *Parallel Collection*), то есть объекты, которые будут обрабатываться параллельно:

```
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:195
```

Извлечь содержимое параллельной коллекции RDD можно вызовом функции *take()*, принимающей в качестве аргумента количество извлекаемых элементов:

```
print(pyspark_entry.take(n_elements))
```

## Датафреймы в PySpark

Датафрейм в *PySpark* — это таблица, строки которой хранятся в RDD. Он похож на датафрейм в *Pandas*, но есть отличия:

- В *PySpark* датафреймы неизменяемые. То есть добавление или переименование столбца приводит к созданию копии существующего датафрейма.
- Работа с датафреймами ведётся по принципу «**ленивых вычислений**» (англ. *lazy evaluations*). Это вычисления, которые откладываются до тех пор, пока пользователь не запросит их результат. Посмотреть на датафрейм можно после вызова функций **collect()** (англ. «собирать») или **show()** (англ. «показывать»).

Для работы с распределённой файловой системой нужен специальный интерфейс передачи данных — **DataFrame API**. Он находится в модуле **Spark SQL** в библиотеке *PySpark*. SQL-запрос можно написать в командной строке этого модуля, то есть без импорта из библиотеки.

Повторим: доступ к библиотеке *PySpark*, в которой создавали RDD, нам предоставил объект *SparkContext*. Есть и другая точка доступа — это объект **SparkSession**.

Создадим объект *SparkSession* вызовом функции **getOrCreate()** (англ. «получить или создать»):

```
from pyspark.sql import SparkSession

APP_NAME = 'sampleApp'

# builder – англ. конструктор сессии
spark = SparkSession.builder.appName(APP_NAME).getOrCreate()
```

Если в программе ещё раз вызвать *getOrCreate()*, функция вернёт тот же объект *SparkSession* и не создаст новый.

## Создание датафреймов

Датафреймы в *PySpark* состоят из строк. В отличие от *Pandas*, строка в *PySpark* — это тип данных, который содержит имена столбцов и их значения в каждой строке таблицы.

Вызовом функции **createDataFrame()** переведём датафрейм из *Pandas* в *PySpark:*

```
import numpy as np
import pandas as pd
from pyspark.sql import SparkSession

APP_NAME = "DataFrames"
SPARK_URL = "local[*]"

spark = SparkSession.builder.appName(APP_NAME).getOrCreate()

df = pd.read_csv('data.csv')
spark_df = spark.createDataFrame(df)

print(spark_df)
```

Для извлечения данных применим функцию *take():*

```
collected_spark_df = spark_df.take(5)
print(collected_spark_df)
```

Датафрейм в *PySpark* стал списком строк:

```
[Stage 0:>
[Row(date='1/1/2009', hour=0, minute=0, pickups=24.0),
Row(date='1/1/2009', hour=0, minute=30, pickups=35.0),
Row(date='1/1/2009', hour=1, minute=0, pickups=25.0),
Row(date='1/1/2009', hour=1, minute=30, pickups=25.0),
Row(date='1/1/2009', hour=2, minute=0, pickups=16.0)]
```

Чтобы избавиться от отображения прогресса, при создании объекта *SparkSession* добавим настройку `showConsoleProgress`, равную `false`.

```
# переносить строки в Python можно символом '\'
spark = SparkSession.builder.appName(APP_NAME) \
        .config('spark.ui.showConsoleProgress', 'false') \
        .getOrCreate()
```

Чтобы загрузить датафрейм из *csv*-файла, у объекта *SparkSession* возьмём атрибут **read**. Вызовем у атрибута функцию *load()*. Эта функция принимает путь к файлу и параметры загрузки, а возвращает датафрейм *PySpark*.

Прочитаем *csv*-файл. Обратите внимание: значения *true* и *false* записываются строчными буквами.

```
# format='csv' – укажите формат файла
# header='true' – укажите, что в файле есть заголовок (имена столбцов)
# inferSchema='true' – англ. выводить схему,
#    укажите, что типы данных должны быть выведены
taxi = spark.read.load('data.csv', format='csv', header='true', inferSchema='true')
```

## Обработка пропущенных значений

Как и в *Pandas*, в *PySpark* есть несколько способов обработки пропущенных значений.

Найдём пропущенные значения в нашем примере. Чтобы получить информацию о столбцах датафрейма, применим функцию *describe()*. А чтобы распечатать результат, вызовем функцию *show():*

```
import numpy as np
import pandas as pd
from pyspark.sql import SparkSession

APP_NAME = "DataFrames"
SPARK_URL = "local[*]"

spark = SparkSession.builder.appName(APP_NAME) \
        .config('spark.ui.showConsoleProgress', 'false') \
        .getOrCreate()

spark_df = spark.read.load('dataset.csv', format='csv', header='true', inferSchema='true')

print(spark_df.describe().show())
```

Чтобы удалить пропущенные значения, нужно применить метод *dropna()*:

```
spark_df = spark_df.dropna()
```

Заполнить пропущенные значения в датафрейме другими заданными (например, нулём) можно методом *fillna()*:

```
spark_df = spark_df.fillna(value)
```

## SQL-запросы в датафреймах

Выведём на экран подробную информацию о данных методом **summary()**:

```
print(spark_df.summary().show())
```

Напишем SQL-запрос в *PySpark*. Но сначала зарегистрируем **временную таблицу**, то есть добавим датафрейм в базу данных:

```
spark_df.registerTempTable("spark_df")
```

Для выполнения SQL-запроса обратимся к объекту *SparkSession:*

```
# query - sql-запрос
print(spark.sql(query).show())
```

## GroupBy в PySpark

Рассмотрим, как в *PySpark* работает функция *groupby():*

```
print(spark_df.groupBy("column_name").mean().select("column_1", "column_2").show())
```

Для сортировки таблицы применим функцию *sort():*

```
print(spark_df.groupBy("column_name").mean().select("column_1", "column_2")) \
      .sort("column_2", ascending=False).show())
```

Точно так же можно применить конструкцию `GROUP BY` в SQL-запросе.