PySpark I

Poliana N Ferreira

Big Data

O que é Big Data?

Termo usado para se referir ao estudo e aplicação de bases de dados que são muito complexas para softwares de processamento de dados comuns

3 Vs do Big Data:

- Volume – Tamanho dos dados
- Variedade – Diferentes fontes e formatos
- Velocidade – Velocidade dos dados

O que é Big Data?

Sistemas de processamento

- Hadoop/MapReduce – escalável, suporta erros, processamento em batches
- Apache Spark – propósito geral e sistema em cluster rápido, processamento em batch e tempo real.

Hadoop

Hadoop

O que é?

- Hadoop é uma plataforma de software que facilita o armazenamento e processamento distribuído de grandes conjuntos de dados.
- Baseado em Java

Hadoop

Componentes

- O **HDFS** é um sistema de arquivos que distribui dados por vários servidores, aumentando a confiabilidade e a disponibilidade. Ele é projetado para lidar com grandes volumes de dados, oferecendo uma solução eficaz para armazenamento de dados distribuído.

- **MapReduce** é um modelo de programação que processa grandes volumes de dados em paralelo, dividindo-os em tarefas menores. Esta abordagem é eficiente para realizar análises e agregações complexas de dados, sendo fundamental em ambientes de big data.

- **YARN** é responsável por gerenciar os recursos dos clusters, otimizando a utilização das capacidades computacionais. Ele permite que várias aplicações utilizem o mesmo cluster de maneira eficiente, coordenando a execução de tarefas.

Hadoop

Componentes

Apache Spark

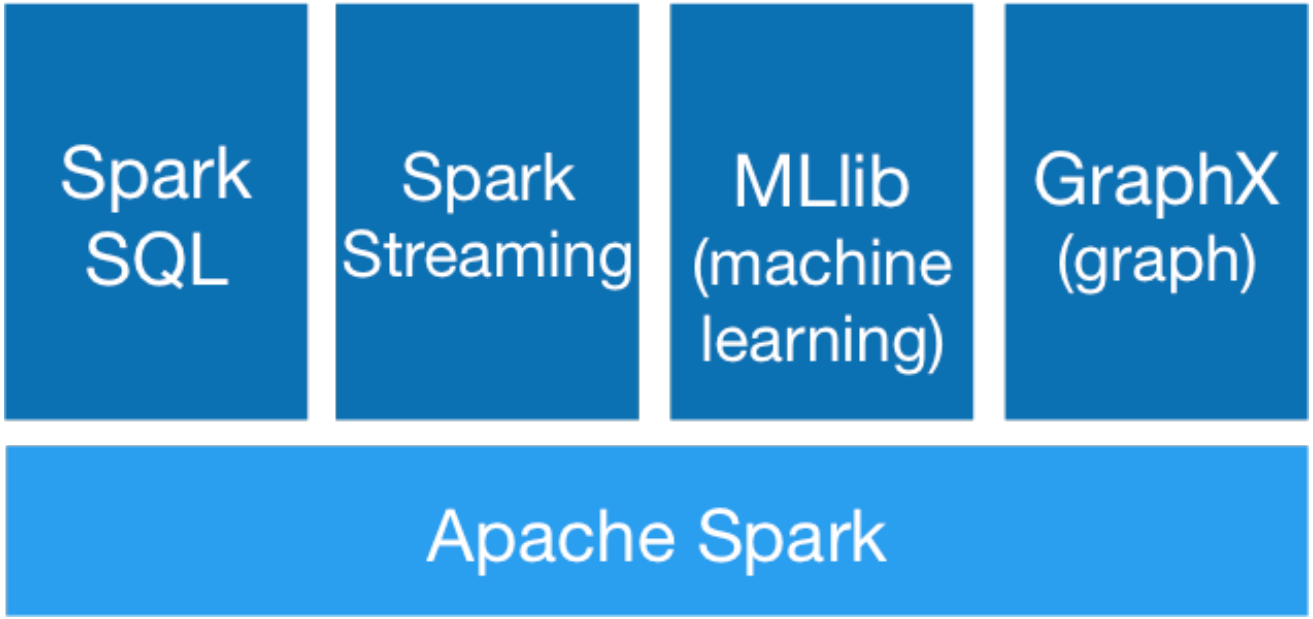
O que é o Apache Spark framework?

Características Principais

- Computação em cluster distribuída
- Computação eficiente em memória para grandes bases de dados
- Muito rápido para processar dados
- Suporte para Java, Scala, Python, R e SQL

O que é o Apache Spark framework?

Componentes



O que é o Apache Spark framework?

Modos de utilização

- Local: máquinas simples como seu computador pessoal
  - Convenientes para testar, debugar e para demonstrações
- Cluster: conjunto de máquinas pré-definidas
  - Usado para produção
- Workflow: Local -> clusters (não é necessário modificações no código para fazer isso)

- Apache Spark é um mecanismo de processamento analítico para aplicativos poderosos de processamento de dados distribuídos e aprendizado de máquina em grande escala.

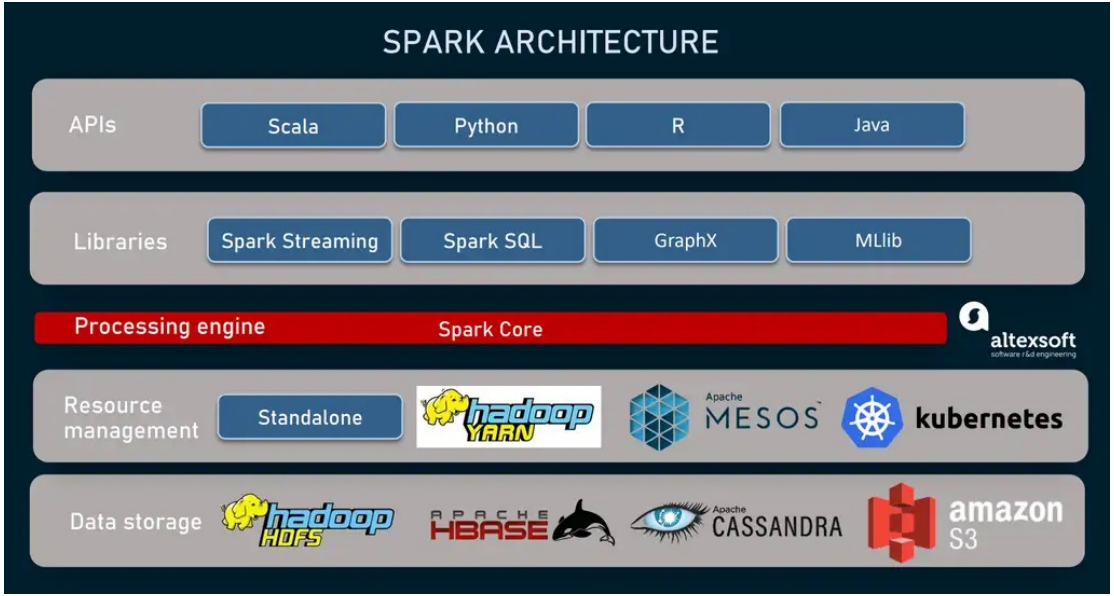
Spark vs. Hadoop

Características Principais

- Hadoop é baseado no modelo MapReduce, que realiza operações de leitura e escrita no disco, tornando-o mais lento comparado ao Spark.
- Apache Spark realiza operações em memória, oferecendo uma performance significativamente mais rápida, especialmente para aplicações que necessitam de várias operações de leitura e escrita.
- Hadoop, principalmente focado em Java, tem um limite maior de entrada devido à complexidade de seu modelo MapReduce.
- O Hadoop, com seu HDFS, tem uma vantagem robusta na gestão de dados distribuídos e tolerância a falhas devido à sua maturidade e ampla adoção.

Spark vs. Hadoop

Uso conjunto



PySpark: Spark com Python

PySpark

O que é?

- PySpark é uma biblioteca Spark para ser usada com Python.
- Em outras palavras, PySpark é uma API Python para Apache Spark

PySpark

Utilização

- Em tempo real, o PySpark tem sido muito usado na comunidade de aprendizado de máquina e cientistas de dados, graças às vastas bibliotecas de aprendizado de máquina Python.

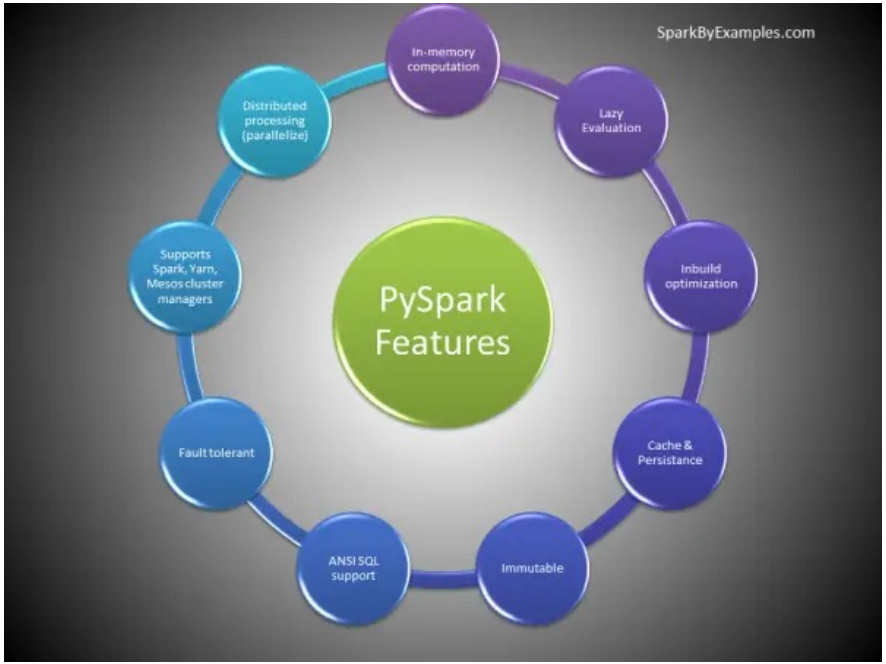
- O Spark executa operações em bilhões e trilhões de dados em clusters distribuídos 100 vezes mais rápido que os aplicativos Python tradicionais ou o Pandas DataFrame.

- PySpark tem sido usado por muitas organizações como Walmart, Trivago, Sanofi, Runtastic e muitas outras.

PySpark

Características do PySpark

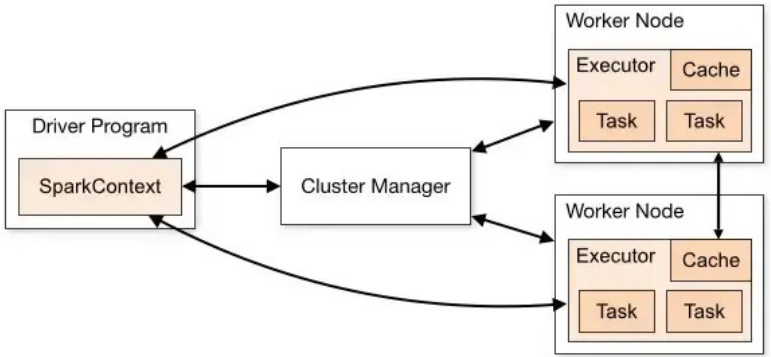
- Computação na memória
- Processamento distribuído usando paralelização
- Pode ser usado com muitos gerenciadores de cluster (Spark, Yarn, Mesos etc.)
- Tolerante a falhas
- Imutável
- Avaliação preguiçosa
- Cache e persistência
- Otimização interna ao usar DataFrames
- Suporta SQL ANSI



Arquitetura

- O Spark funciona em uma arquitetura mestre-escravo onde o mestre é chamado de “Driver” e os escravos são chamados de “Trabalhadores”.

- Quando você executa um aplicativo Spark, o Spark Driver cria um contexto que é um ponto de entrada para seu aplicativo, e todas as operações (transformações e ações) são executadas em nós de trabalho e os recursos são gerenciados pelo Cluster Manager.



Gerenciador de Clusters

- Autônomo – um gerenciador de cluster simples incluído no Spark que facilita a configuração de um cluster.
- Apache Mesos – Mesons é um gerenciador de cluster que também pode executar aplicativos Hadoop MapReduce e PySpark.
- Hadoop YARN – o gerenciador de recursos no Hadoop 2. É usado principalmente como gerenciador de cluster.
- Kubernetes – um sistema de código aberto para automatizar a implantação, escalonamento e gerenciamento de aplicativos em contêineres.
- local – que não é realmente um gerenciador de cluster, usamos “local” para executar o Spark em seu computador.

Instalação PySpark

Instalação

- Você pode realizar a instalação de diversos modos.
  Vai necessitar de: Python, JDK (Java), Spark.

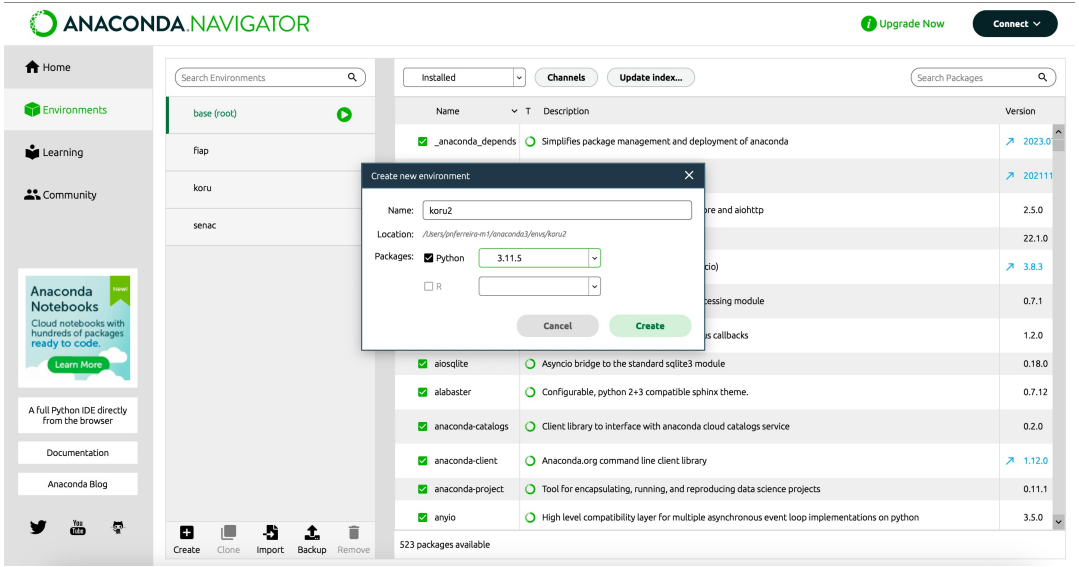
- Localmente, você pode seguir os passos do tutorial a seguir:
  https://sparkbyexamples.com/pyspark/how-to-install-and-run-pyspark-on-windows/

- Ou instalar pelo jupyter notebook usando o Anaconda!

PySpark

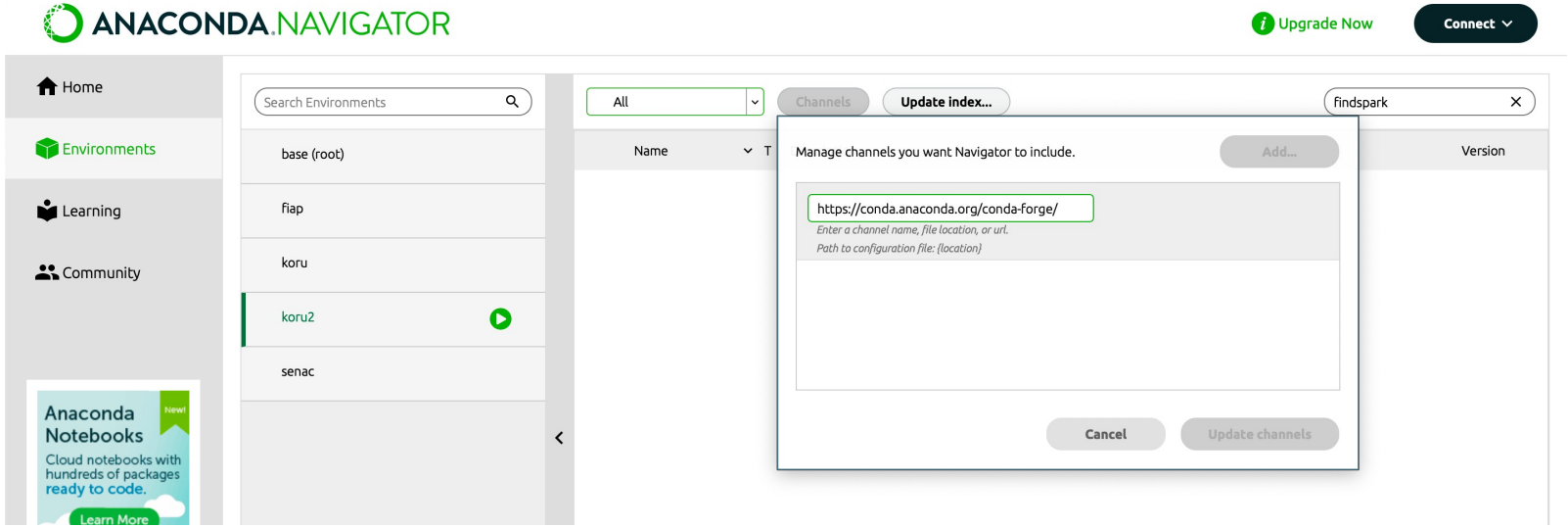
Instalação

- Entrar no anaconda navigator



Canal para o conda-forge

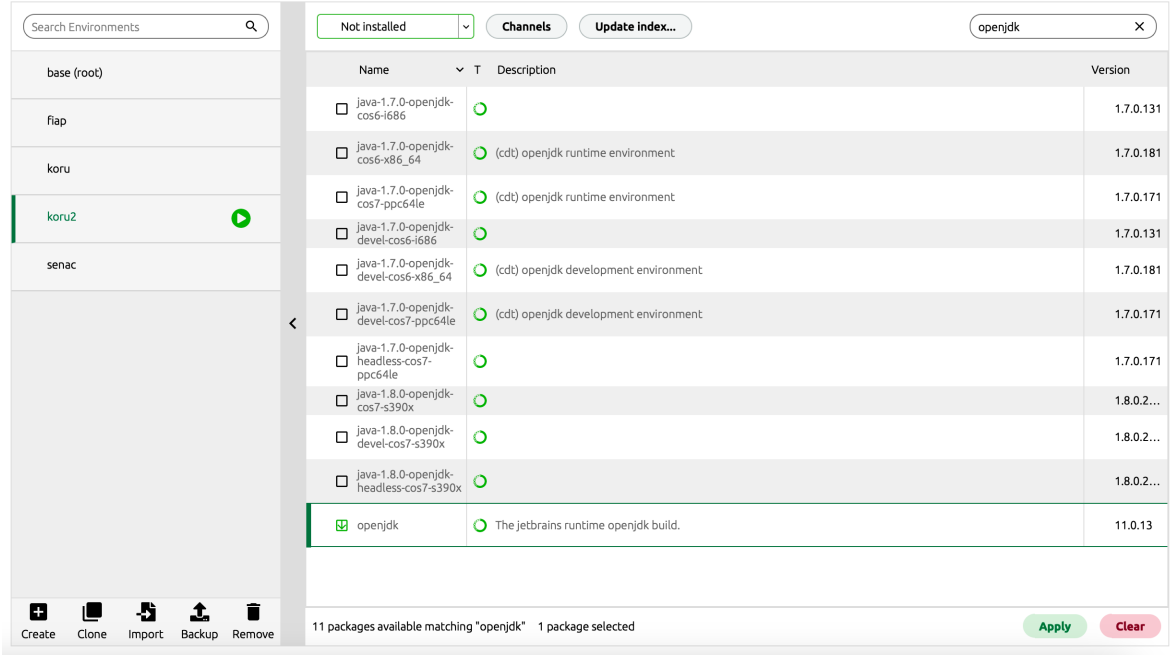
- Adicionar um canal para o conda-forge



PySpark

Instalação

- Instalar as bibliotecas **openjdk**, **pyspark** e **findspark**



PySpark

Instalação

- Abrir o jupyter-notebook
- Conferir instalação

```
import findspark
findspark.init()
findspark.find()
```

- Importando o PySpark

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
      .master("local[1]") \
      .appName("koru") \
      .getOrCreate()
```

SparkSession

SparkSession

O que é?

- Desde o Spark 2.0, o SparkSession se tornou o ponto de entrada para o PySpark trabalhar com RDD e DataFrame.
- Seu objeto spark está disponível por padrão no pyspark-shell e pode ser criado programaticamente usando SparkSession.
- SparkSession também inclui todas as APIs disponíveis em diferentes contextos.

SparkSession

Modos de utilização

```
# Import SparkSession
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
      .master("local[1]") \
      .appName("koru") \
      .getOrCreate()
```

```
# Get Existing SparkSession
spark3 = SparkSession.builder.getOrCreate
print(spark3)
```

RDDs- Resilient Distributed Dataset

PySpark RDD – Resilient Distributed Dataset

O que é?

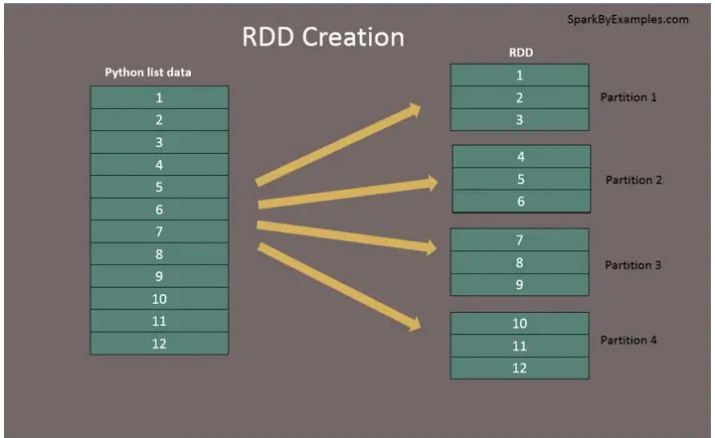
- Estrutura de dados fundamental do PySpark – coleção de objetos distribuídos, imutáveis e tolerantes a falhas, o que significa que depois de criar um RDD, você não poderá alterá-lo.
- Cada conjunto de dados no RDD é dividido em partições lógicas, que podem ser computadas em diferentes nós do cluster.
- A sessão do Spark cria internamente uma variável sparkContext do SparkContext.

PySpark RDD – Resilient Distributed Dataset

using parallelize()

- SparkContext possui diversas funções para usar com RDDs. Por exemplo, seu método parallelize() é usado para criar um RDD a partir de uma lista.

```
# Create RDD from parallelize
dataList = [("Java", 20000), ("Python", 100000), ("Scala", 3000)]
rdd=spark.sparkContext.parallelize(dataList)
```



PySpark RDD – Resilient Distributed Dataset

using textFile()

- RDD também pode ser criado a partir de um arquivo de texto usando a função textFile() do SparkContext.
- Depois de ter um RDD, você poderá realizar operações de transformação e ação. Qualquer operação executada no RDD é executada em paralelo.

```
# Create RDD from external Data source
rdd2 = spark.sparkContext.textFile("/path/test.txt")
```

Operações no RDD

Operações

- No PySpark RDD, você pode realizar dois tipos de operações.
- Transformações RDD – As transformações são operações preguiçosas. Quando você executa uma transformação (por exemplo, atualização), em vez de atualizar um RDD atual, essas operações retornam outro RDD.
- Ações RDD – operações que acionam a computação e retornam valores RDD ao driver.

Operações no RDD

Transformações

- As transformações no Spark RDD retornam outro RDD e as transformações são preguiçosas, o que significa que não são executadas até que você chame uma ação no RDD. Algumas transformações em RDDs são flatMap(), map(), reduzByKey(), filter(), sortByKey() e retornam um novo RDD em vez de atualizar o atual.

Operações no RDD

Ações

- A operação de ação RDD retorna os valores de um RDD para um nó de driver.
- Algumas ações em RDDs são count(), collect(), first(), max(), reduce() e mais.


TALK IS CHEAP
SHOW ME THE CODE

Dataframes

PySpark DataFrame

O que é?

- Coleção distribuída de dados organizados em colunas nomeadas. É conceitualmente equivalente a uma tabela em um banco de dados relacional ou a um quadro de dados em R/Python, mas com otimizações mais ricas nos bastidores.

- O PySpark DataFrame é muito semelhante ao Pandas DataFrame, com a exceção de que os PySpark DataFrames são distribuídos no cluster (o que significa que os dados nos quadros de dados são armazenados em máquinas diferentes em um cluster) e quaisquer operações no PySpark são executadas em paralelo em todas as máquinas, enquanto o Panda Dataframe armazena e opera em uma única máquina.

Pandas e PySpark

- Devido à execução paralela em todos os núcleos em várias máquinas, o PySpark executa operações mais rapidamente que o Pandas em caso de ter muitos e muitos dados.

- Em outras palavras, o Pandas DataFrames executa operações em um único nó, enquanto o PySpark é executado em várias máquinas.

PySpark DataFrame

using createDataFrame()

- Usando a função createDataFrame() do SparkSession você pode criar um DataFrame.

```
data = [('James','','Smith','1991-04-01','M',3000),
  ('Michael','Rose','','2000-05-19','M',4000),
  ('Robert','','Williams','1978-09-05','M',4000),
  ('Maria','Anne','Jones','1967-12-01','F',4000),
  ('Jen','Mary','Brown','1980-02-17','F',-1)
]

columns = ["firstname","middlename","lastname","dob","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
```

PySpark DataFrame

Exibir df

- Como os DataFrame são formatos de estrutura que contêm nomes e colunas, podemos obter o esquema do DataFrame usando df.printSchema()
- df.show() mostra os 20 elementos do DataFrame.

```
df.printSchema()
```

```
root
 |-- firstname: string (nullable = true)
 |-- middlename: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- dob: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: long (nullable = true)
```

```
df.show()
```

```
+---------+----------+--------+----------+------+------+
|firstname|middlename|lastname|       dob|gender|salary|
+---------+----------+--------+----------+------+------+
|    James|          |   Smith|1991-04-01|     M|  3000|
|  Michael|      Rose|        |2000-05-19|     M|  4000|
|   Robert|          |Williams|1978-09-05|     M|  4000|
|    Maria|      Anne|   Jones|1967-12-01|     F|  4000|
|      Jen|      Mary|   Brown|1980-02-17|     F|    -1|
+---------+----------+--------+----------+------+------+
```

PySpark DataFrame

DataFrame de locais externos

- Em aplicações em tempo real, os DataFrames são criados a partir de fontes externas, como arquivos do sistema local, Big Query, HDFS, S3 Azure, HBase, tabela MySQL etc. Abaixo está um exemplo de como ler um arquivo CSV de um sistema local.

```
df = spark.read.csv("/tmp/resources/zipcodes.csv")
df.printSchema()
```

PySpark DataFrame

Operações

- O DataFrame também tem operações básicas, como exibir, selecionar colunas específicas, filtragem, agrupamento, etc.

```
df = spark.read.csv("/tmp/resources/zipcodes.csv")
df.printSchema()
```

PySpark SQL

PySpark SQL

Operações

- PySpark SQL é um dos módulos PySpark mais usados, usado para processar formato de dados colunares estruturados. Depois de criar um DataFrame, você pode interagir com os dados usando a sintaxe SQL.

- Para usar SQL, primeiro crie uma tabela temporária no DataFrame usando a função createOrReplaceTempView(). Uma vez criada, esta tabela pode ser acessada durante todo o SparkSession usando sql() e será descartada junto com o encerramento do SparkContext.

- Use o método sql() do objeto SparkSession para executar a consulta e este método retorna um novo DataFrame.

```
df.createOrReplaceTempView("PERSON_DATA")
df2 = spark.sql("SELECT * from PERSON_DATA")
df2.printSchema()
df2.show()
```

```
root
 |-- firstname: string (nullable = true)
 |-- middlename: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- dob: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: long (nullable = true)

+---------+----------+--------+----------+------+------+
|firstname|middlename|lastname|       dob|gender|salary|
+---------+----------+--------+----------+------+------+
|    James|          |   Smith|1991-04-01|     M|  3000|
|  Michael|      Rose|        |2000-05-19|     M|  4000|
|   Robert|          |Williams|1978-09-05|     M|  4000|
|    Maria|      Anne|   Jones|1967-12-01|     F|  4000|
|      Jen|      Mary|   Brown|1980-02-17|     F|    -1|
+---------+----------+--------+----------+------+------+
```

PySpark SQL

Exemplo

- Vamos ver outro exemplo do PySpark usando groupby.

```
groupDF = spark.sql("SELECT gender, count(*) from PERSON_DATA group by gender")
groupDF.show()
```

```
+------+--------+
|gender|count(1)|
+------+--------+
|     F|       2|
|     M|       3|
+------+--------+
```

Agora vamos avaliar a aula?

Obrigada!

escola