Tema 3

# Princípios de Desenvolvimento de Spark com Python

É fundamental que o profissional de Tecnologia da Informação compreenda a tecnologia Hadoop para gerenciar uma infraestrutura complexa e desenvolver aplicações para uma das áreas que mais cresce na atualidade: Big Data.

Tempo total de leitura
126 min.

Créditos

F
Professor (a)
FERNANDO DURIER

## Introdução

Você sabia que as aplicações de Big Data estão em praticamente todos os setores da sociedade moderna?

Nesse cenário de oportunidades e desafios, a Apache Foundation desenvolveu o framework Hadoop: a tecnologia que estudaremos aqui. Além de explorar conceitos, também vamos demonstrar alguns exemplos práticos que ajudarão você a entender o funcionamento do Hadoop.

### Preparação

Para reproduzir os exemplos apresentados ao longo deste conteúdo, você deve utilizar o sistema operacional Windows e instalar o Java 8 em seu computador. Além disso, também precisará instalar o Hadoop.

### Objetivos

Ao final desta aula, você será capaz de:

- Analisar a arquitetura da tecnologia de framework de software livre chamada Hadoop.
- Comparar os sistemas HDFS e RDBMS.
- Descrever a preparação do ambiente para uso prático do Hadoop.

- Aplicar um exemplo prático no Hadoop.

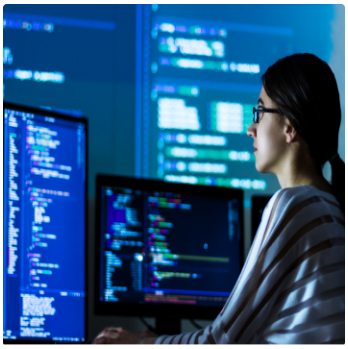
# 01. Framework Apache SPARK

## Fundamentos da tecnologia Spark

### O que é Apache Spark?

É um framework de código aberto de computação distribuída usado para aplicações de Big Data. É muito mais eficiente do que o Hadoop para gerenciar e processar tarefas, devido à utilização de cache de memória e algoritmo de processamento otimizado.

O Apache Spark fornece APIs de desenvolvimento para as linguagens de programação Python, Java, Scala e R. Além disso, fornece recursos para o desenvolvimento de aplicações de aprendizado de máquina, SQL, análise gráfica e bibliotecas de streaming.



O principal objetivo do Spark é trabalhar com grandes volumes de dados gerados em tempo real. É uma tecnologia de computação em cluster baseada no modelo MapReduce do Hadoop. Seu elemento fundamental para obter eficiência superior ao Hadoop é a estratégia de uso de memória, chamada de **clustering**.

Acompanhe a evolução dessa tecnologia:

**2009**

O Spark foi iniciado na Universidade da Califórnia, em Berkeley.

**2010**

O código foi aberto sob uma licença BSD (Berkeley Software Distribution).

### 2013

O projeto foi adquirido pela Fundação de Software Apache.

### 2014

O Spark foi promovido a um projeto Apache de importante relevância.

No início, os desenvolvedores trabalhavam no Hadoop. Diante de suas limitações, perceberam a necessidade de aumentar:

- A velocidade no processamento das consultas aos dados;
- A eficiência do processamento em memória;
- A eficiência de tolerância às falhas.

## Benefícios do Spark

O Spark se estabeleceu como um framework que tem como benefícios:

# Custo

O Spark é um framework de código aberto. Portanto, sua licença é gratuita, assim como é o caso do Hadoop. Mas, para que o Spark possa ser utilizado na prática, é necessário usar equipamentos que, obviamente, têm custos.

# Facilidade de uso

Além de oferecer APIs para programação em Java, Scala, Python, linguagem R e SQL, o Spark também disponibiliza componentes de aplicação, como Hive, Pig, Sqoop, HBase.

# Bibliotecas com diversas finalidades

O Spark produz alto desempenho para processamento de dados em lote e fluxo (streaming).

# Velocidade

O Spark produz alto desempenho para processamento de dados em lote e fluxo (streaming).

## Cenários adequados para uso do Spark

O Spark é um framework voltado para aplicações de Big Data que obteve sucesso reconhecido na prática, principalmente por ser muito mais veloz do que o Hadoop.

Entre os cenários adequados de aplicação do Spark estão:

### Integração de dados

Consiste nos processos de extrair, transformar e carregar os dados – Extract Transform Load (ETL). É uma situação bastante rotineira, em que é necessário realizar tratamento dos dados antes de incorporá-los às bases de dados da organização. Por exemplo, podemos processar grandes volumes de arquivos de texto e planilhas.

## Processamento de fluxo

Fazer o processamento de dados gerados em tempo real é um desafio, como, por exemplo, analisar os arquivos de log de um site acessado por diversos usuários simultaneamente.

## Aprendizado de máquina

Funciona melhor quando os modelos são treinados por grande volume e variedade de dados. Por exemplo, podemos utilizar os logs dos arquivos de navegação dos usuários para alimentar modelos de redes neurais que traçam padrões de comportamento.

## Análise interativa

É a capacidade de gerar respostas rapidamente. Portanto, em vez de executar consultas predefinidas, é possível lidar com os dados de forma interativa. Por exemplo, podemos traçar gráficos que nos ajudam a entender comportamentos em tempo real. Isso é muito útil para sistemas de segurança e monitoramento de audiência.

### Cenários inadequados para uso do Spark

Existem algumas situações em que o uso do Spark é inadequado, e é melhor utilizar outra ferramenta de Big Data. Entre essas situações estão:.

## Gerenciamento de cluster

O ajuste e a manutenção do Spark são tarefas complexas. Essa dificuldade tem um grande impacto em garantir o alto desempenho do Spark para tratar grandes volumes de dados. É uma situação típica de aplicações de Ciência de Dados.

## Ambiente com limitação dos recursos computacionais

A principal causa para o alto desempenho de processamento de Spark é a utilização intensa da memória do cluster. Se o cluster ou máquinas virtuais tiverem pouca capacidade de computação, o Spark não será uma escolha adequada. No lugar dele, podemos utilizar o Hadoop.

## Atividade

### Questão 1

As aplicações de Big Data são dinâmicas e cada vez mais comuns. Então, é necessário utilizar recursos computacionais, como frameworks e plataformas de desenvolvimento que auxiliem na criação de aplicações eficientes para atender a essa demanda.

O Spark é um framework de alto desempenho usado com sucesso exatamente para esse tipo de situação. Nesse sentido, assinale a alternativa correta a respeito do Spark e seus cenários de aplicação:

> Para que o Spark possa ser utilizado com alto desempenho, é

A

necessário fazer e manter configurações de diversos parâmetros.

B

As aplicações de Big Data são caracterizadas por seu dinamismo, que envolve volume, variedade e velocidade sobre a geração e o tratamento dos dados – situações essas verificadas automaticamente pelo Spark.

C

O Spark é uma tecnologia de programação distribuída que utiliza os recursos da rede da melhor forma possível, sem a necessidade de nenhum tratamento específico.

D

Apesar de ser aplicado com sucesso para Big Data, o Spark não é adequado para projetos de aprendizado de máquina, devido a características específicas.

E

O Spark é uma ferramenta ideal para aplicações do tipo ETL, mas não é adequado para operações analíticas em tempo real.

## Parabéns! A alternativa A está correta.

O Spark é um framework para aplicações de Big Data que possui alto desempenho. Por isso, é a principal tecnologia atualmente para esse tipo de aplicação. Ele pode ser utilizado para aplicações de Internet das Coisas (IoT), redes sociais, portais de notícias e muitas outras situações similares. Mas sua configuração é um desafio! É necessário um profundo conhecimento de sua plataforma e da aplicação para configurar os parâmetros adequadamente.

# Arquitetura do Spark

### Modelo de abstrações

A arquitetura do Spark é baseada no modelo mestre-escravo com camadas e componentes bem definidos e fracamente acoplados. Além disso, ela também possui

integração com várias bibliotecas. Essa arquitetura é baseada em duas abstrações. Vamos entendê-las com mais detalhes a seguir.

### Conjunto de dados distribuídos resilientes

Do inglês “Resilient Dataset Distribution (RDD)”. O **conjunto de dados** (datasets) contempla os dados organizados em grupos. Esses dados são:

## Distribuídos

São alocados a nós diferentes.

## Resilientes

São recuperados em caso de falha.

Em outras palavras, são grupos de itens de dados que podem ser armazenados na memória dos nós trabalhadores.

Os RDDs suportam esses dois tipos de operações:

### Transformações

Operações como mapa, filtro, junção e união, que são executadas em um RDD e geram um novo RDD contendo o resultado.

### Ações

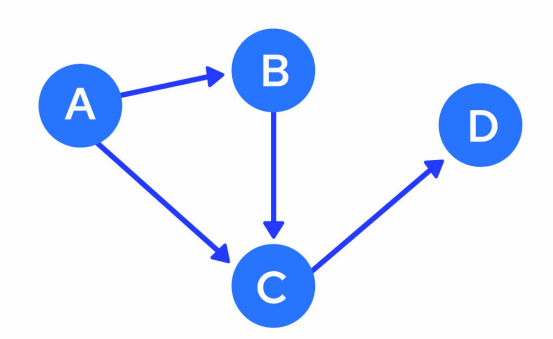
Operações como redução, contagem e primeiro ou último, que retornam um valor após a execução de um cálculo em um RDD.

### Grafo acíclico dirigido

Do inglês “Directed Acyclic Graph” (DAG). O grafo representa um mapa de navegação, enquanto as propriedades **acíclico** e **dirigido** fazem referência ao modo como a

navegação é feita.

Na imagem a seguir, apresentamos um exemplo de um grafo desse tipo:




Exemplo de grafo dirigido acíclico.

Os nós do grafo são os círculos com rótulos **a**, **b**, **c** e **d**. As arestas são os pares ordenados dados por \((a, b),(b, c),(a, c)\) e \((c, d)\).

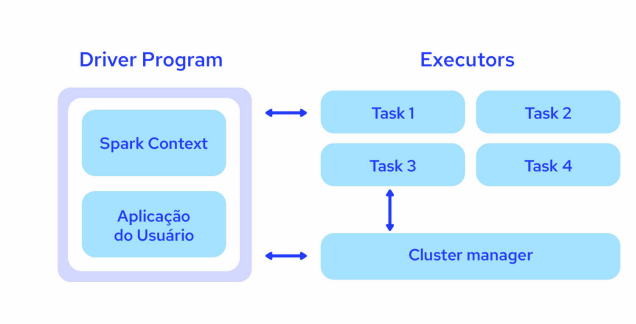
No caso do Spark, o DAG é um grafo que executa uma sequência de operações nos dados. Cada nó do grafo é uma partição RDD, e a aresta representa uma transformação sobre os dados.

## Elementos estruturais

A arquitetura básica do Spark é composta por três componentes principais:

- Driver Program;
- Cluster Manager;
- Executors;

Na imagem a seguir, podemos visualizá-los:




Arquitetura do Spark.

Agora, vamos detalhar cada um desses componentes e seus elementos.

## Driver Program

É responsável por converter uma **aplicação do usuário** em unidades de execução menores, chamadas de **task** (tarefas). Em seguida, essas tarefas são programadas (agendadas) para serem executadas com um gerenciador de cluster (Cluster Manager) nos executores. Também é da responsabilidade do Driver Program executar a aplicação Spark e retornar o status ou resultado para o usuário.

### SparkContext

Conecta-se ao Cluster Manager para executar em um cluster. Em seguida, executa as seguintes tarefas:

- Adquire executores em nós do cluster.
- Envia o código da aplicação para os executores – o código da aplicação pode ser definido por arquivos JAR ou Python passados para o SparkContext.
- Envia tarefas para que sejam processadas pelos executores.

### Cluster Manager

É o gerenciador de cluster – um componente opcional que só será necessário se o Spark for executado de forma distribuída. Ele é responsável por administrar as máquinas que serão utilizadas como workers.

Os gerenciadores de cluster podem ser classificados em três tipos (APACHE SPARK, 2018):

## Independente (standalone)

Gerenciador de cluster simples que facilita a configuração de um cluster.

## Apache Mesos

Gerenciador geral de cluster que também pode executar Hadoop MapReduce.

# Hadoop YARN

Gerenciador de recursos no Hadoop.

Além desses três tipos, há um suporte experimental para **Kubernetes**, que é uma plataforma de código aberto para fornecer infraestrutura centrada em contêineres.

### Worker Node

São os nós de trabalho, ou seja, as máquinas que, de fato, executarão as tarefas enviadas pelo Driver Program. Se o Spark for executado no modo local, a máquina desempenhará tanto o papel de Driver Program quanto de worker.

Os workers são nós escravos, e sua função é executar o código do aplicativo no cluster. Além disso, eles são compostos por:

- **executor** (executores) – processos de nós de trabalho encarregados de executar tarefas individuais em determinado trabalho do Spark;
- **task** (tarefas) – unidades de trabalho que são enviadas a um executor.

## Atividade

### Questão 1

A arquitetura do Spark é projetada em modelo hierárquico com entidades com responsabilidades bem definidas. Essas entidades se relacionam entre si para gerenciar etapas do ciclo de vida dos processos. A forma como o processo é executado é essencial para o alto desempenho do Spark. Nesse sentido, assinale a alternativa correta a respeito da arquitetura do Spark.

A | Uma de suas abstrações é o conjunto de dados distribuídos resilientes (RDD), que viabiliza a execução paralela dos processos com tolerância a falhas.

B | Essa arquitetura é baseada no modelo mestre-escravo, em que o componente Executor realiza o papel do mestre, e o Worker Node desempenha o papel do escravo.

C

O SparkContext é o componente responsável por controlar a execução das tarefas.

D

O Cluster Manager é o componente que transforma as aplicações em tarefas.

E

Essa arquitetura é formada por componentes que separam os processos em blocos tratados como tarefas pelo SparkContext.

## Parabéns! A alternativa A está correta.

Uma de suas abstrações é o conjunto de dados distribuídos resilientes (RDD), que viabiliza a execução paralela dos processos com tolerância a falhas.

# Fluxo de execução e ecossistema do Spark

## Passo a passo de execução no Spark

Após conhecer os papéis dos elementos estruturais da arquitetura do Spark, agora, vamos analisar o ciclo de vida de uma aplicação no Spark.

O fluxo de execução de uma aplicação segue estes passos:

1. Quando uma aplicação inicia, o Driver Program converte-a em um grafo acíclico dirigido (DAG) e cria uma instância de um SparkContext;
2. O Driver Program solicita recursos para o gerenciador do cluster (cluster manager) para iniciar os executores;
3. O gerenciador de cluster ativa os executores;
4. O processo do Driver Program é executado por meio da aplicação do usuário. Dependendo das ações e transformações sobre os RDDs, as tarefas são enviadas aos executores;
5. Os executores executam as tarefas e salvam os resultados;
6. Se algum Worker Node falhar, suas tarefas serão enviadas a outros executores para serem processadas novamente.

## Linguagens de programação do Spark

De forma resumida, o Spark é uma ferramenta computacional voltada para aplicações de Big Data, em que podemos fazer o agendamento, a distribuição e o monitoramento de várias aplicações distribuídas.

O ecossistema do Spark é formado pelos recursos de linguagem de programação, os quais permitem que os projetos possam ser programados em diferentes linguagens e por distintos tipos de componentes totalmente integrados à plataforma.

As linguagens de programação que podemos utilizar para desenvolver projetos no Spark são:

### Scala

É a linguagem em que o framework do Spark foi desenvolvido. A programação em Scala para Spark possui algumas vantagens, como ter acesso aos recursos mais recentes antes que estejam disponíveis em outras linguagens de programação.


### Python

Possui diversas bibliotecas adequadas para aplicações de ciência de dados e aprendizado de máquina.

### Linguagem R

É utilizada também em contextos de ciência de dados e aprendizado de máquina.


### Java

É uma linguagem especialmente interessante para desenvolver projetos relacionados ao Hadoop.


## Componentes do Spark

Têm como objetivo facilitar o desenvolvimento de projetos com finalidades específicas, como os de aprendizado de máquina, por exemplo.

Os principais componentes do ecossistema do Spark são (CHELLAPPAN; GANESAN, 2018):

Spark Core

Contém a funcionalidade básica do Spark. Todas as demais bibliotecas do Spark são construídas sobre ele. Esse componente suporta RDD como sua representação de dados. Entre suas atribuições estão: agendamento de tarefas, recuperação de falhas, gerenciamento de memória e distribuição de trabalhos entre nós de trabalho (Worker Nodes). Um exemplo são as bibliotecas SQL.

Spark SQL

Fornece suporte para dados estruturados e permite consultar os dados por meio da linguagem SQL. Além disso, também oferece suporte para representação dos dados a partir de datasets e dataframes.

Spark Streaming

Oferece suporte ao processamento escalonável e com tolerância a falhas de dados de streaming. Esse componente é usado para processar dados que são transmitidos em tempo real. Um exemplo é quando pesquisamos por um produto na Internet e imediatamente começamos a receber anúncios sobre ele nas plataformas de mídia social. Outras aplicações do Spark Streaming incluem análise de sensores de IoT, aplicações no processamento de log de redes, detecção de intrusão e detecção de fraude, anúncios e campanhas on-line, finanças e gerenciamento da cadeia de suprimentos.

MLlib

É uma biblioteca que contém vários algoritmos de aprendizado de máquina, como correlações e testes de hipóteses, classificação e regressão, agrupamento e análise de componentes principais. Por exemplo, o Spark MLlib inclui algoritmos para clusterização.

GraphX

É uma biblioteca usada para manipular e realizar cálculos paralelos aos grafos. Com ela, podemos visualizar os dados em forma de grafo, além de fornecer

vários operadores para manipulação de grafos e combiná-los com RDDs. Essa biblioteca é bastante útil para trabalharmos com bancos de dados orientados a grafos.

## Atividade

### Questão 1

A tecnologia do Spark oferece diversos recursos e nos permite desenvolver aplicações específicas com mais praticidade. Nesse sentido, assinale a alternativa correta a respeito do ecossistema do Spark:

A Permite trabalhar com diversas linguagens de programação.

B É focado na manipulação de dados.

C Trata apenas da aceleração das operações de cálculo.

D Trabalha somente com aplicações de Internet das Coisas (IoT).

E É voltado só para aplicações de streaming.

## Parabéns! A alternativa A está correta.

Os principais componentes do Spark nos oferecem diversos recursos para trabalharmos com aplicações que vão desde as operações com banco de dados e o tratamento de fluxo de dados em tempo real até algoritmos de aprendizado de máquina. Além disso, podemos utilizar esses recursos em diversas linguagens de programação, como Python, linguagem R, Scala e Java.

# 02. Instalação, configuração e uso do Spark

## Preparação do ambiente

**Preparando o ambiente para aplicação prática no Spark**

1. Instale em seu computador a linguagem Java 8, que é o pré-requisito para usar o Apache Spark. Como utilizaremos o Google Colab, então, crie uma célula de código e execute o seguinte comando:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

2. Instale o Apache Spark com Hadoop, executando o seguinte comando:

```
!wget -q https://downloads.apache.org/spark/spark-3.2.2/spark-3.2.2-bin-hadoop3.2.tgz
```

   Isso significa que você fez o download do arquivo a seguir, da versão 3.2.2 do Spark e 3.2 do Hadoop: **spark-3.2.2-bin-hadoop3.2.tgz**

   Para escolher outras versões, acesse o site https://downloads.apache.org/spark/.

3. Instale o pacote PySpark, que é uma interface do Python para o Apache Spark, essencial para o desenvolvimento de nossas aplicações. Como ele não está no caminho do sistema por padrão (sys.path), você precisa resolver isso adicionando o PySpark ao sys.path em tempo de execução por meio do FindSpark, que torna o PySpark uma biblioteca regular.

   3.1. Para instalar o FindSpark, execute o seguinte comando:

```
!pip install -q findspark
```

   3.2. Para instalar o PySpark, execute a linha de comando a seguir em uma célula do Google Colab:

```
!pip install -q pyspark
```

4. Configure as variáveis de ambiente JAVA_HOME e SPARK_HOME, pois, para funcionar, o Spark precisa acessar os recursos dessas variáveis. Para isso, escreva os comandos a seguir em uma célula de código e execute-o:

4.1. Fique atento aos nomes dos arquivos. Eles precisam fazer referência às versões que você baixou. Do contrário, a configuração estará errada.

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.2.2-bin-hadoop3.2"
```

4.2. Na primeira linha do código, observe que importamos o pacote **os**, que é uma interface para o sistema operacional. No caso de nosso programa, associamos as variáveis de ambiente aos caminhos em que o Java e o Spark estão instalados no Google Colab.

4.3. Fique atento aos nomes dos arquivos. Eles precisam fazer referência às versões que você baixou. Do contrário, a configuração estará errada.

# Atividade

## Questão 1

Agora é sua vez!

Você já fez a instalação e configuração do Spark. Agora, é sua vez de praticar para adquirir mais confiança sobre o funcionamento do Spark!

Depois de realizar os passos anteriores, teste se o módulo FindSpark foi instalado corretamente. Dica: utilize a função help(nome_do_módulo).

Abrir solução

Para testar se o módulo FindSpark foi instalado corretamente, basta criar uma nova célula de código e executar o comando:

```
help(findspark)
```

A saída vai informar detalhes do módulo, a versão e o local de instalação.

## Começando a trabalhar com o Spark

**Aprendendo a desenvolver uma primeira aplicação no Spark**

1. Inicie o FindSpark para utilizar o PySpark depois. Para isso, escreva e execute o programa a seguir em uma célula de código do Google Colab:

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

Na primeira linha, observe que importamos o FindSpark. Já na segunda, nós o iniciamos para habilitar a utilização do PySpark.

2. Crie uma sessão no Spark, que permite trabalhar com RDD, Dataframe e Dataset. Para isso, escreva o código a seguir em uma célula de código do Google Colab e, em seguida, execute-o:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Observe que, na segunda linha, tivemos de usar o método getOrCreate para criar uma sessão no Spark ou, simplesmente, obter uma sessão, caso já exista.

2.1. Ainda na sessão do Spark, analise o master. Existem dois casos possíveis:

2.1.1. Execução no cluster – use um nome como yarn ou mesos como um argumento para master, dependendo da configuração do cluster.

2.1.2. Execução no modo autônomo – use local[parâmetro]. O parâmetro deve ser um valor inteiro maior que 0, pois representa quantas partições devem ser

criadas ao usar RDD, DataFrame e Dataset. O valor ideal de parâmetro deve ser o número de núcleos de CPU que temos à disposição.

3. Depois de criar a sessão do Spark, utilize-a para acessar dados. Considere os dados que já estão disponíveis no Google Colab – no caso, um dataset sobre residências na Califórnia. Para ver esses dados, acesse um símbolo de diretório no lado esquerdo do Google Colab.

    3.1. Crie outra célula de código, implementando e executando o seguinte comando:

```
dataset =
spark.read.csv('/content/sample_data/california_housing_test.csv',inferSchema=True,
header =True)
```

    3.2. Você terá os dados do arquivo “california_housing_test.csv” disponíveis na variável “dataset”. Para visualizar a estrutura desses dados, execute a linha a seguir em uma célula de código do Google Colab: dataset.printSchema()

    A saída é dada por:

    root

```
|-- longitude: double (nullable = true)
|-- latitude: double (nullable = true)
|-- housing_median_age: double (nullable = true)
|-- total_rooms: double (nullable = true)
|-- total_bedrooms: double (nullable = true)
|-- population: double (nullable = true)
|-- households: double (nullable = true)
|-- median_income: double (nullable = true)
|-- median_house_value: double (nullable = true)
```

    Com isso, você executou nosso exemplo completo do Spark no Google Colab.

4. Feche a sessão no Spark, escrevendo o código a seguir em uma célula de código do Google Colab e, em seguida, execute-o: spark.stop()

# Atividade

## Questão 1

Você acabou de desenvolver uma aplicação prática com o Spark. Agora, refaça todos os passos anteriores, com exceção do comando spark.stop(). Seu exercício é apresentar o conteúdo básico do Dataset.

Abrir solução

Aqui, você precisa criar uma célula de código e executar o seguinte comando: dataset.head()

A saída da execução deve ser semelhante a:

```
Row(longitude=-122.05, latitude=37.37, housing_median_age=27.0, total_rooms=3885.0, total_bedrooms=661.0, population=1537.0, households=606.0, median_income=6.6085, median_house_value=344700.0)
```

Excelente! Você já consegue explorar algumas das funcionalidades do Spark. Continue seus estudos para aprofundar seus conhecimentos. Bom trabalho!

# 03. Interface PySpark

## Spark no Python: PySpark

### O que é o PySpark?

É uma biblioteca Spark para executar programas Python usando recursos do Apache Spark, ou seja, trata-se de uma API Python para Apache Spark. Como já vimos, o Apache Spark é um framework aplicado para o processamento de dados distribuídos de alto desempenho.

Vejamos as principais vantagens do PySpark:

- Tem um mecanismo de processamento distribuído em memória de uso geral, que permite processar dados com eficiência de maneira distribuída.
- Executa as aplicações de forma muito mais eficiente do que os sistemas tradicionais.
- É muito eficiente para pipelines de ingestão de dados.
- Pode processar dados HDFS do Hadoop e muitos outros sistemas de arquivos.
- Pode ser usado para processar dados em tempo real.
- Possui bibliotecas de grafos e de aprendizado de máquina.

## Utilização do PySpark

Para utilizarmos o PySpark, precisamos instalar o Spark e suas dependências, além de configurarmos as variáveis de ambiente. Já aprendemos a fazer isso.

Agora, precisamos realizar os seguintes passos:

# Conectar-se a um cluster Spark do PySpark.

# Realizar operações com Spark DataFrames.

Vamos entender melhor cada um deles.

## Conectando-se a um cluster Spark do PySpark

O primeiro passo que precisamos fazer para utilizar o Spark é conectar-se a um cluster. Para realizar a conexão, devemos criar uma instância da classe SparkContext. O construtor dessa classe possui alguns argumentos que permitem especificar os atributos do cluster ao qual estamos nos conectando.

- Durante esse processo, vamos utilizar um computador, chamado de mestre, que faz o gerenciamento da divisão dos dados e dos cálculos.
- O mestre é conectado aos demais computadores do cluster, que são chamados de trabalhadores.
- O mestre envia os dados e cálculos para execução dos trabalhadores, e estes enviam seus resultados de volta ao mestre.

Em síntese, para criar uma instância da classe SparkContext, devemos executar os seguintes passos:

1

## Importar a biblioteca PySpark.

2

## Instanciar um objeto SparkContext.

O código que faz exatamente essa sequência é este:

Python

```python
from pyspark import SparkContext
spark_contexto = SparkContext()
print(spark_contexto)
print(spark_contexto.version)
```

Observe as tarefas realizadas em cada linha:

- Na primeira linha, fazemos a importação do PySpark;
- Na segunda linha, instanciamos o objeto SparkContext;
- Na terceira linha, imprimimos informações sobre o objeto spark_contexto;
- Na quarta linha, imprimimos a versão do objeto spark_contexto.

### Realizando operações com Spark DataFrames

O Spark DataFrame tem um comportamento similar a uma tabela SQL com variáveis nas colunas e registros nas linhas. Além disso, os DataFrames são mais otimizados para operações complexas do que os RDDs.

> Quando trabalhamos diretamente com os RDDs, fica sob nossa responsabilidade realizar operações de modo eficiente, enquanto o DataFrame já faz esse processo otimizado automaticamente.

Já realizamos o primeiro passo para começarmos a trabalhar com o Spark DataFrames, que foi a criação do SparkContext para estabelecermos uma conexão com o cluster..

Agora, precisamos criar um SparkSession, que é a interface com essa conexão. Para isso, vamos executar o código a seguir para criar uma sessão no Spark (SparkSession):

Python

```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate() # Create my_spark
print(spark) # Print my_spark
```

Observe as tarefas realizadas em cada linha:

- Na primeira linha, importamos o SparkSession a partir do pyspark.sql;
- Na segunda linha, criamos uma sessão do Spark;
- Na terceira linha, imprimimos o objeto Spark.

Agora, vamos ler os dados de um arquivo disponível no diretório sample_data do Google Colab. Em nosso caso, utilizamos o arquivo california_housing_test.csv. Vejamos o seguinte código para fazer a leitura dos dados do arquivo:

Python

```
1  dataset = spark.read.csv('/content/sample_data/california_housing_test.
```

Para visualizarmos os dados de Dataset, executamos o código a seguir, que vai exibir o cabeçalho das colunas e o conteúdo da primeira linha: dataset.head()

Esse código produz a saída dada por:

Python

```
1  Row(longitude=-122.05, latitude=37.37, housing_median_age=27.0, total_r
```

Para obtermos a quantidade de linhas no Dataset, basta executarmos o seguinte código: dataset.count()

Agora, vamos criar uma tabela SQL temporária com os dados do Dataset. Para isso, devemos executar o código:

Python

```python
dataset.createOrReplaceTempView("tabela_temporaria")
print(spark.catalog.listTables())

```

A segunda linha imprime as tabelas no catálogo.

O próximo passo é fazer uma consulta SQL. Vamos selecionar apenas três registros com os dados referentes às colunas longitude e latitude. O código que devemos executar é:

Python

```python
query = "FROM tabela_temporaria SELECT longitude, latitude LIMIT 3"
saida = spark.sql(query)
saida.show()

```

# Atividade

## Questão 1

Uma das vantagens de trabalhar com o Spark é que essa tecnologia oferece suporte para diversas linguagens de programação. Uma dessas linguagens é o Python, que pode ser acessado pela API PySpark. Essa API possui diversas características que precisam ser bem compreendidas para desenvolver aplicações de qualidade do Python no Spark. Nesse sentido, assinale a alternativa correta sobre as características do PySpark:

A — O PySpark interage com os RDDs exclusivamente por meio de consultas SQL.

B — Uma das vantagens de utilizar o PySpark é que não é necessário fazer a instalação de dependências.

C — Uma das limitações do PySpark é que a única maneira disponível para interagir com os RDDs do Spark é por meio de listas.

D — Para utilizar o PySpark, é necessário fazer apenas a instalação das dependências e importar o pacote PySpark.

E — Os nós de execução de um processo com o PySpark são abstratos, ou seja, não são acessíveis diretamente.

## Parabéns! A alternativa E está correta.

O PySpark é uma API do Spark para utilizarmos Python e desenvolvermos soluções. Nesse contexto, as regras de gerenciamento dos processos são do Spark, em que não é possível endereçar um nó de processamento individualmente.

# PySpark com Pandas

## Funcionalidades do pacote Pandas

O Pandas facilita trabalharmos com dados relacionais a partir de várias estruturas de dados e operações que nos ajudam a manipular dados numéricos e séries temporais. Ele foi construído sobre NumPy, que é outra biblioteca muito importante (talvez a mais importante) do Python para aplicações de ciência de dados.

As estruturas e funcionalidades do Pandas foram construídas para otimizar o desempenho e a produtividade para os desenvolvedores.

> Aqui, temos especial interesse no Pandas DataFrame: uma estrutura de dados tabular bidimensional que pode mudar o tamanho durante a execução – desde que queiramos que isso ocorra. Essa estrutura também pode ser heterogênea, ou seja, os dados podem ser de tipos diferentes.

As linhas e colunas do DataFrame são rotuladas. Portanto, o Pandas DataFrame possui três componentes principais:

Dados

Linhas

Colunas

Com o Pandas, também podemos fazer a limpeza de dados por meio da exclusão de linhas, seja porque os dados não são relevantes, seja porque contêm valores errados, como valores vazios ou nulos.

A seguir, vamos estudar como o Pandas se relaciona com as aplicações do PySpark. Em especial, vamos usar o Pandas para trabalhar com SQL e DataFrame do Spark.

## Convertendo Spark SQL para Pandas

Vamos voltar ao exemplo que já havíamos iniciado: a ideia é obter a localização de residências com a maior quantidade de quartos. Portanto, o resultado da consulta deve

apresentar a latitude e a longitude do bloco residencial com a maior quantidade de quartos.

A primeira parte de nossa solução vai ser dividida nos seguintes passos:

1. Implementar a consulta SQL na tabela chamada de tabela_temporaria, que já carregamos no Spark para retornar a quantidade máxima de quartos;
2. Executar a consulta SQL no Spark e, assim, obter um DataFrame do Spark;
3. Converter o resultado da etapa anterior para um DataFrame do Pandas;
4. Imprimir o resultado da consulta;
5. Converter o valor do DataFrame para um valor inteiro.

A seguir, podemos ver o código de nossa solução, que reproduz exatamente os passos que descrevemos.

Python

```
query1 = "SELECT MAX(total_rooms) as maximo_quartos FROM tabela_tempora
q_maximo_quartos = spark.sql(query1)
pd_maximo_quartos = q_maximo_quartos.toPandas()
print('A quantidade máxima de quartos é: {}'.format(pd_maximo_quartos['
qtd_maximo_quartos = int(pd_maximo_quartos.loc[0,'maximo_quartos'])

```

Na última linha do código, tivemos de fazer algumas manipulações para acessar o valor que queremos converter para inteiro. Isso foi necessário, pois o retorno é um DataFrame. Depois de executar o programa, obtemos a seguinte saída:.

Python

```
A quantidade máxima de quartos é: 0    30450.0
Name: maximo_quartos, dtype: float64
```

```
3
4
```

Observe que, na primeira linha da saída, aparecem os valores 0 e 30450.0, que são, respectivamente, a localização do elemento e o valor dentro do DataFrame. Na segunda linha, aparece o campo Name com o rótulo maximo_quartos, que foi o nome associado ao retorno do SQL.

Agora, vamos continuar a solução, que consiste em obter a localização do bloco residencial com a maior quantidade de quartos. Para isso, vamos realizar os seguintes passos:

1. Implementar a consulta SQL para retornar a latitude e a longitude da residência com a quantidade máxima de quartos que obtivemos na execução do programa anterior;
2. Executar o SQL no Spark e obter o resultado no DataFrame do Spark;
3. Converter o DataFrame do Spark para o DataFrame do Pandas;
4. Exibir o resultado.

A seguir, apresentamos nosso código:

Python

```
query2 = "SELECT longitude, latitude FROM tabela_temporaria WHERE total
localizacao_maximo_quartos = spark.sql(query2)
pd_localizacao_maximo_quartos = localizacao_maximo_quartos.toPandas()
print(pd_localizacao_maximo_quartos.head())
```

Logo no início do código, na consulta SQL, tivemos de converter a variável qtd_maximo_quartos para string e, em seguida, fazer a concatenação dela.

Outro ponto que precisamos observar é a conversão do DataFrame do Spark para o DataFrame do Pandas por meio da função toPandas.

Por fim, na última linha, exibimos o resultado por meio da função head do Pandas DataFrame. A saída do programa é:

Python

```
}longitude  latitude
·0     -117.2     33.58
```

No caso, havia apenas um bloco residencial com 30450 quartos. O zero (0) que aparece na segunda linha da saída se refere ao número do registro. No Python, a indexação de listas e vetores começa na posição zero.

## Atividade

### Questão 1

A linguagem de programação Python é utilizada em diversos contextos de aplicação. É uma referência para aplicações de ciência de dados e de aprendizado de máquina, que, por sua vez, têm no domínio do Big Data o cenário ideal para aplicação. É possível

desenvolver aplicações no Spark com Python por meio da API PySpark. Nesse sentido, assinale a alternativa correta a respeito da programação Python no Spark:

A — O único modo de manipular dados com os DataFrames é por meio do uso da indexação, similar ao que ocorre com uma matriz.

B — Uma das principais características do Spark é o alto desempenho da manipulação dos dados, obtido pelo uso direto dos RDDs com o PySpark.

C — Os DataFrames facilitam o processo de manipulação dos dados a partir de estruturas similares a planilhas, mas que precisam ser convertidas em RDDs no final da aplicação.

D — Os DataFrames são recursos do Python que permitem trabalhar com dados heterogêneos, facilitando a manipulação de RDDs.

E — A estrutura de dados DataFrame é exclusiva do pacote Pandas do Python, assim como o RDD é um componente exclusivo do Spark.

## Parabéns! A alternativa D está correta.

Os DataFrames são estruturas de dados do Python que pertencem à biblioteca Pandas, muito utilizadas para manipular dados heterogêneos – uma situação recorrente para aplicações de Big Data. Com a utilização do PySpark, é possível fazer a interação entre DataFrames e RDDs e, assim, utilizar diversas funcionalidades que facilitam o processo de desenvolvimento de uma solução.

# Convertendo Pandas DataFrame para Spark DataFrame

### Processo de conversão

Agora, vamos estudar um exemplo que converte um DataFrame do Pandas para um DataFrame no Spark. Nosso exemplo seguirá estes passos:

1. Gerar dados aleatórios que seguem a distribuição normal com média e desvio-padrão que fornecemos – vamos usar a biblioteca Numpy para gerar os dados e a biblioteca Pandas para organizá-los em um DataFrame;
2. Converter o DataFrame do Pandas para um DataFrame do Spark;
3. Imprimir a lista de tabelas no catálogo do Spark;
4. Adicionar a tabela temporária no catálogo do Spark;
5. Examinar as tabelas no catálogo do Spark novamente.

A seguir, apresentamos o código:

Python

```python
import pandas as pd
import numpy as np
media = 0
desvio_padrao=0.1
pd_temporario = pd.DataFrame(np.random.normal(media,desvio_padrao,100)
spark_temporario = spark.createDataFrame(pd_temporario)
print(spark.catalog.listTables())
spark_temporario.createOrReplaceTempView("nova_tabela_temporaria")
print(spark.catalog.listTables())

```

Demos o nome para a tabela do Spark de nova_tabela_temporaria. O primeiro print produz a seguinte saída:

Python

```python
[Table(name='tabela_temporaria', database=None, description=None, table

```

Aqui, listamos apenas as tabelas que já estavam no catálogo. Depois de executar a função createOrReplaceTempView, obtemos a seguinte saída:

Python

```
[Table(name='nova_tabela_temporaria', database=None, description=None,
```

Essa saída inclui informações sobre a última tabela que criamos. Para concluir, precisamos fechar a sessão, executando o código: spark.stop().

## Dicas e boas práticas

O fato de termos usado o Google Colab para desenvolver nosso projeto facilita bastante o processo, pois é mais fácil realizarmos testes e ajustes sem a necessidade de fazer muitas configurações.

Uma boa prática de programação nesse tipo de ambiente é usar uma célula de código para executar as sequências que apresentamos, pois, se houver algum problema, será mais fácil detectá-lo e corrigi-lo.

### Atenção!

Uma fonte de erros comum é não observar a indentação, o que, para a maioria das linguagens, não seria problema. Mas, no caso do Python, a indentação faz parte da sintaxe para delimitação de blocos lógicos.

Outra questão importante: quando rodamos os exemplos, não podemos esquecer de instalar os pacotes e as dependências, exatamente como foi apresentado.

## Atividade

### Questão 1

Utilizar a linguagem de programação Python para desenvolver aplicação com Spark pode trazer bastante benefícios. Um dos recursos do Python é a estrutura DataFrame, que facilita bastante a manipulação dos dados. Nesse sentido, assinale a alternativa correta a respeito da conversão do DataFrame do Python para o Spark:

A — É mais fácil trabalhar no DataFrame do Spark do que no DataFrame do Python.

B — Os programadores tendem a se adaptar rapidamente ao uso de novas tecnologias.

C — Só é possível trabalhar com manipulação de dados com o DataFrame do PySpark.

D — O DataFrame do Spark usa de forma mais eficiente os dados do que o Python.

E — O DataFrame do Python é uma estrutura que não tem flexibilidade.

### Parabéns! A alternativa D está correta.

O DataFrame do PySpark está preparado para trabalhar com o RDD, que é um dos componentes básicos da arquitetura do Spark. Portanto, ele utiliza melhor os recursos computacionais do que o DataFrame tradicional do Python.

## MapReduce

### Aspectos essenciais do Mapreduce

Atualmente, usamos diversas aplicações distribuídas em nosso dia a dia. Alguns exemplos são serviços na Internet, computação móvel e IoT. Com a popularização da Internet, as aplicações distribuídas começaram a ganhar protagonismo, pois começaram a surgir diversas demandas que não precisavam nem podiam passar por uma abordagem sequencial. Dessa forma, houve a necessidade de aplicar técnicas de processamento distribuído.

A ideia básica do processamento distribuído é aplicar uma técnica muito conhecida para resolver problemas de computação: **dividir para conquistar.**. Confira, a seguir, as duas etapas:

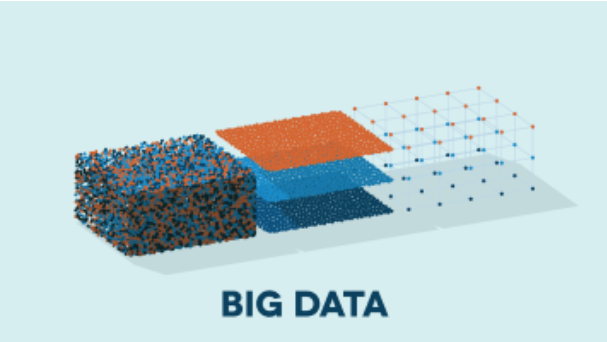
# Etapa de divisão

Chamada de ação mapeamento, os blocos de dados são separados em pequenas partes gerenciáveis.

# Etapa de conquista ou recombinação

Chamada de ação redução, são realizados cálculos sobre esses dados, até que sejam totalmente processados, atingindo, assim, o objetivo da aplicação.

Esse cenário é uma descrição do paradigma **MapReduce**, que foi introduzido pela empresa Google por volta de 2004 (HAMSTRA *et al.*, 2015).




O MapReduce é uma estratégia de computação com capacidade para processar grandes conjuntos de dados de forma distribuída em várias máquinas. Sua essência é mapear um conjunto de dados de entrada em uma coleção de pares (chave-valor) e, em seguida, reduzir todos os pares com a mesma chave.

Essa forma de fazer o processamento é muito eficiente para aplicações distribuídas, pois quase todos os dados podem ser mapeados em pares (chave-valor). As chaves e os valores podem ser de qualquer tipo: strings, inteiros, tipos abstratos e até os próprios pares (chave-valor).

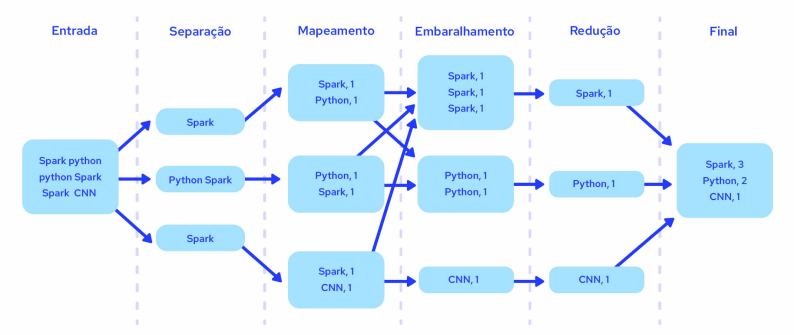
## Exemplo

O exemplo clássico para utilização de MapReduce é o que trata da contagem das frequências de palavras em um texto grande. Mas essa estratégia também pode ser utilizada para fazer classificações e pesquisas distribuídas, em aprendizado de máquina e em muitas outras situações práticas que apenas confirmam a importância dessa técnica utilizada pelo Spark.

## Paradigma MapReduce

Como vimos, o exemplo clássico de MapReduce é o de contar a quantidade de palavras em um arquivo. A ideia é bem simples: a entrada de dados do sistema é um arquivo texto que pode conter palavras repetidas. No final, o algoritmo produz uma lista com as palavras e a quantidade de vezes que elas apareceram no arquivo.

Na imagem a seguir, podemos entender como o MapReduce trabalha no exemplo de contagem de palavras:




Exemplo de MapReduce.

---

Vamos conhecer cada uma das etapas apresentadas:

Entrada

Consiste no conjunto de dados que vamos fornecer para o programa. No caso do exemplo, é um arquivo texto com palavras que podem estar repetidas. De modo geral, a entrada de dados de programação distribuída é composta por elementos em uma lista que podem ser tratados de forma independente ao longo das fases de processamento até que o processo seja concluído.

Separação

Refere-se à separação de dados em listas que serão processadas de modo independente.

Mapeamento

Aplica uma função a cada elemento de uma lista e coleta o resultado. No caso do exemplo de contagem de palavras, nosso objetivo é mapear cada palavra no arquivo de entrada em um par chave-valor: a chave é a palavra, e o valor é o número de ocorrências.

Embaralhamento

Agrupa todos os valores intermediários que possuem a mesma chave de saída. Em nosso exemplo de contagem de palavras, cada retângulo é composto pelas mesmas palavras.

Redução

Conta o número de valores com a mesma chave. Essa etapa é muito eficiente devido ao processamento das etapas anteriores, pois é um simples processo de contagem.

Final

Apresenta as palavras com suas respectivas frequências de repetição.

## Atividade

### Questão 1

A computação distribuída é naturalmente aplicada para Big Data. Isso ocorre devido às características próprias dos projetos com grandes volumes de dados e todas as complexidades envolvidas. Algumas dessas complexidades se referem à velocidade com que os dados são gerados e processados. Nesse sentido, assinale a alternativa correta a respeito da computação distribuída no contexto do Spark.

A

O Spark utiliza a técnica de dividir para conquistar, a fim de processar grandes volumes de dados e, assim, obter alto desempenho.

B

Para processar grandes volumes de dados, o Spark utiliza o MapReduce, que faz a separação dos dados em diversas etapas, de modo a serem processados e recombinados até concluir a solução do problema.

C

A fase de redução consiste na separação dos dados, os quais, posteriormente, são mapeados com as funções que realizarão operações sobre eles.

D

O Spark faz o processamento dos dados que estão agrupados em RDDs por meio da utilização eficiente do armazenamento híbrido.

E

Para que o Spark possa aplicar um processamento eficiente, é necessário que os dados sejam homogêneos, de forma a serem particionados em pares chave-valor e, em seguida, tratados na fase de redução.

## Parabéns! A alternativa B está correta.

O MapReduce é uma técnica fundamental na computação distribuída, em que os dados são submetidos a duas etapas: mapeamento e redução. Na fase de mapeamento, são gerados pares chave-valor que, em seguida, são agrupados e processados, completando a etapa de redução. Essa técnica é utilizada pelo Spark e por outros frameworks, como o Hadoop, por exemplo.

# Transformações e ações com PySpark

## Transformações do Spark

São funções que recebem um RDD como entrada e produzem um ou mais RDDs como saída. Elas não modificam o RDD de entrada, pois os RDDs são imutáveis. A ideia é transformar um conjunto de dados de entrada no conjunto que queremos obter.

Por definição, as transformações do Spark são operações preguiçosas (lazy evaluations). Isso significa que os novos RDDs são criados apenas depois da execução de uma operação classificada como ação. Portanto, a transformação cria um conjunto de dados a partir de um existente.

Ainda sobre o processo preguiçoso das transformações, quando aplicamos uma transformação em qualquer RDD, ela não será executada de imediato. O Spark cria um grafo acíclico dirigido (DAG) com as seguintes informações:

- Operação aplicada;
- RDD origem;
- Função usada para a transformação.

Apenas quando aplicarmos uma ação, serão realizados os cálculos sobre o RDD. Existem dois tipos de transformações. Vamos conhecer cada uma delas.

## Transformações estreitas

São aquelas resultantes da aplicação de funções de mapeamento e de filtragem. Os dados se originam de uma única partição. Essa característica é chamada de **autossuficiência**.

Portanto, as partições de um RDD de saída possuem registros que se originam de uma única partição no RDD pai. Além disso, o Spark utiliza apenas um subconjunto de partições para obter o resultado.

O Spark agrupa as transformações estreitas como um estágio conhecido como **pipelining**. A ideia é fazer um processamento com a maior quantidade de operações em uma única partição de dados.

### Exemplo

map() – aplica a transformação em cada elemento de uma fonte de dados e retorna um novo conjunto de dados, que podem ser RDD, DataFrame ou Dataset.
flatMap() – faz o nivelamento das colunas do conjunto de dados resultante depois de aplicar a função em cada elemento e retorna um novo conjunto de dados.
filter – faz uma filtragem dos registros de uma fonte de dados.
mapPartitions() – é parecida com a função map(), pois executa a função de transformação em cada partição de dados.
sample() – retorna um subconjunto aleatório dos dados de entrada.
union() – retorna a união de dois conjuntos de dados de entrada.

## Transformações amplas

Em alguns casos, os dados necessários para realizar os cálculos com os registros em uma única partição podem estar em muitas partições. Portanto, é necessário realizar movimentos de dados entre as partições para executar as transformações, que são chamadas de amplas. Elas também são conhecidas como transformações aleatórias, porque podem depender de uma mistura aleatória.

### Exemplo

Intersection() – retorna a interseção de dois RDDs.
distinct() – retorna um novo RDD com os elementos distintos de um RDD de origem.
groupByKey() – é aplicada sobre um conjunto de dados de pares (K, V) – K representa a chave (Key), e V representa o valor (Value) – e retorna um conjunto de dados de pares agrupados pelas chaves.
reduceByKey() – opera também sobre um conjunto de dados de pares (K, V) e retorna um conjunto de dados de pares (K, V), no qual os valores para cada chave são agregados, usando a função redução, que deve ser do tipo (V, V) = > V.
sortByKey() – opera sobre um conjunto de dados de pares (K, V) e retorna um conjunto de dados de pares (K, V) ordenados por K.
join() – combina os campos de duas fontes de dados, usando valores comuns.
coalesce() – utiliza uma partição existente para que menos dados sejam misturados e, assim, seja possível diminuir o número de partições.

## Ações do Spark

Uma ação no Spark retorna o resultado dos cálculos no RDD. Para realizar o processamento, uma ação utiliza o DAG para carregar os dados no RDD original, realizar todas as transformações intermediárias e retornar os resultados para o Driver Program ou gravá-los no sistema de arquivos.

Ou seja, as ações são operações sobre os RDDs que produzem valores, e não RDD.

### Exemplo

first() – retorna o primeiro elemento no RDD.
take() – retorna um vetor com os primeiros n elementos do conjunto de dados, onde n é um parâmetro da função.
reduce() – agrega elementos de um conjunto de dados por meio de funções.
collect() – retorna os elementos do conjunto de dados como um vetor.
count() – retorna a quantidade de elementos no RDD.

## Atividade

### Questão 1

As aplicações de Big Data demandam técnicas específicas de tratamento. Além do grande volume e da grande variedade de dados, existem questões associadas à periodicidade com que esses dados são gerados e a expectativa de tempo em que devem ser processados. O Spark utiliza duas categorias de operações que se complementam para processar os dados: transformações e ações. Nesse sentido, assinale a alternativa correta a respeito dessas categorias do Spark.

A | As transformações não executam o processamento de imediato, apenas criam novos conjuntos de dados a partir dos que já existem.

B | Para otimizar o processamento dos dados, o Spark aplica ações responsáveis por criar grupos de pares chave-valor que serão processados posteriormente pelas transformações.

C | As transformações modificam os dados dos RDDs, de modo a fazer um pré-processamento que reduzirá a quantidade de operações necessárias quando as ações forem aplicadas.

D | As transformações estreitas são responsáveis por modificar pequenas partições dos dados que impactarão na redução de operações quando as ações forem aplicadas.

E

As transformações e ações podem ser executadas simultaneamente ao longo do processamento de RDDs, com o objetivo de otimizar o tempo de execução de um processo no Spark.

## Parabéns! A alternativa A está correta.

As operações do Spark para processamento de dados são classificadas em transformações e ações. Ambas são essenciais no ciclo de vida de processamento e têm responsabilidades distintas, com o objetivo de otimizar o tempo de processamento dos dados. As transformações não executam o processamento imediato dos dados, mas criam dados a partir de RDDs existentes, que serão utilizados posteriormente pelas ações. Esse é o chamado processamento preguiçoso.

# 04. Práticas no PySpark

## Práticas com MapReduce

**Praticando paradigma MapReduce para programação distribuída**

1. Prepare-se para utilizar o MapReduce, instanciando uma sessão. Para isso, escreva o seguinte código:

```
from pyspark import SparkContext
spark_contexto = SparkContext()
```

2. Acompanhe o exemplo 1 – Vetores, que tem como objetivo receber um vetor numérico e, para cada elemento do vetor, aplicar a seguinte função matemática: \(f(x)=x^2+x\).

    2.1. Para criar o vetor, utilize o pacote Numpy e importe esse pacote, usando o seguinte código:

    ```
    import numpy as np
    ```

    2.2. Entre com os dados do vetor, conforme o código:

    ```
    vetor = np.array([10, 20, 30, 40, 50])
    ```

    2.3. Crie um RDD por meio de um SparkContext, usando o seguinte código:

    ```
    paralelo = spark_contexto.parallelize(vetor)
    ```

2.4. Faça uma rápida verificação do conteúdo da variável paralelo, usando o código:

```
print(paralelo)
```

Esse código produz a seguinte saída, a qual indica que você criou o RDD com sucesso:

```
ParallelCollectionRDD[3] at readRDDFromFile at
PythonRDD.scala:274
```

2.5. Como o vetor já está no Spark, você pode aplicar o mapeamento. Para isso, use o seguinte código:

```
mapa = paralelo.map(lambda x : x**2+x)
```

Aqui, lambda x : x**2+x é uma função lambda correspondente à função matemática que queremos aplicar aos elementos da entrada de dados. O código paralelo.map faz o mapeamento da função para cada elemento da entrada.

2.6. Colete os dados, ou seja, verifique o resultado por meio do código:

```
mapa.collect()
```

Esse código produz a seguinte saída que queríamos:

```
[110, 420, 930, 1640, 2550]
```

3. Acompanhe o exemplo 2 – Listas, que tem como objetivo contar a frequência das palavras de uma lista.

3.1. Entre com uma lista de palavras, conforme o seguinte código:

```
paralelo = spark_contexto.parallelize(["distribuida",
"distribuida", "spark", "rdd", "spark", "spark"])
```

Com isso, a variável paralelo faz referência ao RDD.

3.2. Implemente uma função lambda que simplesmente associa o número 1 a uma palavra. O código fica exatamente assim:>

```
funcao_lambda = lambda x:(x,1)
```

Ou seja, a função recebe uma variável x e vai produzir o par (x, 1).

3.3. Aplique o MapReduce, visualizando o código primeiro e analisando-o em seguida. O código é dado por:

```
from operator import add
mapa = paralelo.map(funcao_lambda).reduceByKey(add).collect()
```

Observe que, na primeira linha, importamos o operador add, que será usado para somar as ocorrências das palavras mapeadas.

Já na segunda linha, realizamos muitas tarefas, como:

• O mapeamento dos dados da variável paralelo para a função lambda – ou seja, para cada palavra, criamos um par (palavra, 1).

• A aplicação da redução por meio da função reduceKey, que soma as ocorrências e as agrupa pela chave – no caso, pelas palavras.

• A coleta dos dados na etapa final, dada pela função collect, em uma lista que chamamos de mapa.

3.4. Para visualizar o resultado, use o seguinte código:

```
for (w, c) in mapa:

  print("{}: {}".format(w, c))

Esse código percorre a lista mapa e imprime cada par formado pela palavra e sua respectiva ocorrência. A saída é a seguinte:

distribuida: 2

spark: 3

rdd: 1
```

3.5. Para concluir, feche a sessão, usando o código:

```
spark_contexto.stop()
```

# Atividade

## Questão 1

Você acabou de desenvolver uma aplicação prática utilizando MapReduce com o PySpark. Agora, aproveite os passos anteriores e modifique a função de mapeamento para que calcule \(f(x)=x^4-10 x^2+3\). Ao final do exercício, apresente o resultado da variável somatorio.

Abrir solução

Aqui, você precisa mudar a função de mapa para:

```
mapa = paralelo.map(lambda x : x**4-10*x**2+3)
```

O resultado da variável somatorio é dado por:

```
9735015
```

Pronto! Você já desenvolveu mais um exemplo com PySpark. Continue seus estudos para aprofundar seus conhecimentos.

## Exemplo prático de transformação e ação

**Desenvolvendo um exemplo prático de transformação e ação**

Acompanhe o exemplo que envolve as operações de transformações e de ações. Para isso, instancie um SparkContext, usando o código:

```
from pyspark import SparkContext

spark_contexto = SparkContext()
```

1. Forneça uma lista de entrada e transforme-a em um RDD por meio do código:

   ```
   lista = [1, 2, 3, 4, 5, 3]
   lista_rdd = spark_contexto.parallelize(lista)
   ```

2. Execute a ação de contar os elementos do RDD. Para isso, implemente o código a seguir, que vai produzir na saída o valor 6:

   ```
   lista_rdd.count()
   ```

3. Crie uma função lambda que recebe um número como parâmetro e retorna um par formado pelo número do parâmetro e pelo mesmo número multiplicado por 10. Para isso, use o código:

   ```
   par_ordenado = lambda numero: (numero, numero*10)
   ```

4. Aplique a transformação flatMap com a ação collect da função lambda para a lista_rdd. Para isso, use o código a seguir, que vai produzir a saída [1, 10, 2, 20, 3, 30, 4, 40, 5, 50, 3, 30]:

   ```
   lista_rdd.flatMap(par_ordenado).collect()
   ```

5. Aplique a transformação map com a ação collect da função lambda para a lista_rdd. Para isso, use o código a seguir, que vai produzir a saída [(1, 10), (2, 20), (3, 30), (4, 40), (5, 50), (3, 30)]:

```
lista_rdd.map(par_ordenado).collect()
```

6. Para concluir, feche a sessão com o seguinte código:

```
spark_contexto.stop()
```

## Atividade

### Questão 1

Você já sabe trabalhar com aplicações no Spark, utilizando MapReduce. Agora, aproveite os passos anteriores e modifique a função de mapeamento par ordenado para que calcule \(f(x)=\left(x, x^2\right)\). Ao final do exercício, apresente o resultado do comando

```
lista_rdd.map(par_ordenado).collect().
```

Abrir solução

Você precisa mudar a função de mapa para:

```
par_ordenado = lambda numero: (numero, numero**2)
```

O resultado da execução é dado por:

```
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (3, 9)]
```

Parabéns por ter realizado esses exercícios práticos! Agora, você já sabe como utilizar o Spark com PySpark. Para consolidar seus conhecimentos, reveja os conceitos e exemplos e faça muitos testes. Bons estudos!.

# Conclusão

### O que você aprendeu neste conteúdo?

Neste conteúdo, você aprendeu:

1. Os principais conceitos do framework Apache Spark;
2. A instalação, a configuração e o teste do Spark;
3. Os aspectos básicos do PySpark, como as operações de MapReduce;
4. A implementação de exemplos práticos no PySpark.

### Explore +

- Acesse o site oficial do **Apache Spark** e aprofunde seus conceitos sobre sua arquitetura e os componentes de seu ecossistema.
- Acesse o site oficial do **Pandas com PySpark** e explore diversos exemplos práticos que ajudarão você a aprimorar seus conhecimentos.

## Referências bibliográficas


APACHE SPARK.. **Cluster mode overview.** Maryland: The Apache Software Foundation, 2018.

CHELLAPPAN, S.; GANESAN, D.. **Practical Apache Spark: using the Scala API.** Berkeley: Apress, 2018.

HAMSTRA, M. et al.. **Learning Spark: lightning-fast Big Data analytics.** Sebastopol: O'Reilly Media, 2015.