Fundamentos de Big Data

UNIDADE 06

Acessando dados no HDFS com PySpark

O problema dos "hot takes" e o mercado de dados

Você já deve ter visto em discussões pela internet frases como:

> *Python é uma linguagem lenta e horrível, todo mundo deveria começar estudando C.*
> *Assembly é melhor do que C.*
> *Odeio Java.*
> *Só faço os meus códigos em R para a empresa de Finanças que trabalho,*

e é uma perda de tempo querer estudar outra coisa.
Todo mundo deveria saber Excel.
Só preciso saber JavaScript.

Diria que frases assim são "*hot takes*". *Hot takes* são opiniões rápidas e muitas vezes provocativas. Podem gerar discussões interessantes, mas são *opiniões*, e muitas vezes sem embasamento algum. A pessoa acha que aquilo é verdade "porque sim". Contudo, isso é um problema para nós que atuamos em TI e, ainda, para quem atua em dados.

Primeiramente, os *hot takes* podem muitas vezes sacrificar a profundidade e a nuance do mundo real. Isso pode levar a uma compreensão superficial de eventos complexos e a uma polarização desnecessária. Você pode achar JavaScript a melhor linguagem do mundo, mas não significa que ela seja *a melhor linguagem para todas as aplicações*, por exemplo. Outra pessoa pode achar Python uma linguagem lenta – não porque tenha testado isso em aplicações reais, mas porque ouviu outra pessoa falar isso.

A cultura do *hot take* pode incentivar a reação em vez da reflexão. Isso pode desvalorizar o pensamento crítico e a análise cuidadosa, que são bem importantes para uma tomada de decisão completa e equilibrada dos eventos. Logo, cuidado com estas opiniões.

VIDEOAULA: Python, Java, Scala e R: o que isso tem a ver com Big Data?

Agora: por que estou falando sobre isso com você? Em dados, não estamos restritos a somente uma linguagem de programação – de fato, temos quatro grandes linguagens nesta área (isso sem contar o SQL): *Python, Java, Scala* e *R.* E podemos usar o Spark em todas elas.

Relembrando o Spark

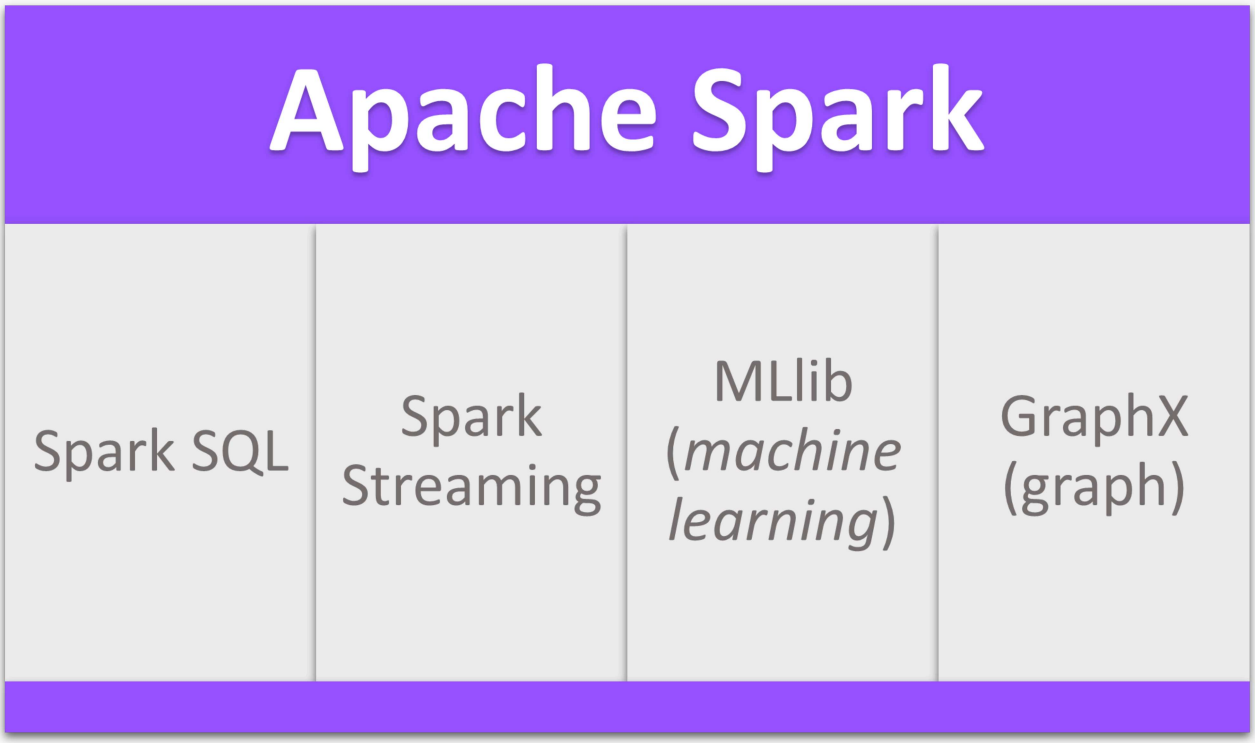
O Apache Spark é um framework de computação distribuída que oferece uma velocidade de processamento de dados muito superior ao Apache Hadoop, especialmente quando os dados são armazenados em memória. Ele pode ser até cem vezes mais rápido em comparação com o Hadoop MapReduce, e até dez vezes mais rápido quando os dados estão em disco. Além disso, o Spark é *muito* versátil, permitindo a leitura e escrita de diversos tipos de dados, como HDFS, Cassandra, Apache HBase e Amazon S3.

Com o Spark**,** você pode facilmente ler, transformar e agregar grandes volumes de dados, além de treinar e implantar modelos complexos de *machine learning* . Suas APIs são acessíveis em várias linguagens, como Java, Scala, Python, R e SQL, tornando-o adequado para integrá-lo nas diferentes arquiteturas que empresas de setores e tamanhos diferentes possuem.

Também é possível criarmos aplicativos completos ou empacotar suas análises como bibliotecas para serem executadas em um *cluster*. Além disso, o Spark oferece suporte para análises interativas rápidas por meio de notebooks como Jupyter (lembra?), notebooks do Databricks e o Apache Zeppelin.

O Spark é composto de alguns componentes, como visto na figura abaixo. Ele possui o Spark SQL para consultas SQL, Spark Streaming para processamento de dados em tempo real, MLlib para aprendizado de máquina (*machine learning*) e o GraphX para processamento de grafos.

Figura 1. Ferramentas de alto nível suportadas pelo Spark



Fonte: Pereira *et al*. (2019, p. 81).

O Apache Spark também usa *clusters*, como ilustrado na figura a seguir. São três componentes principais:

Driver Program

Esse é o coração do Spark. Ele contém o seu programa **SparkContext**, que é o ponto de entrada para fazermos qualquer coisa com o Spark. O Driver Program também traduz as tarefas do seu aplicativo em unidades de trabalho que podem ser distribuídas e executadas nos *worker nodes*.

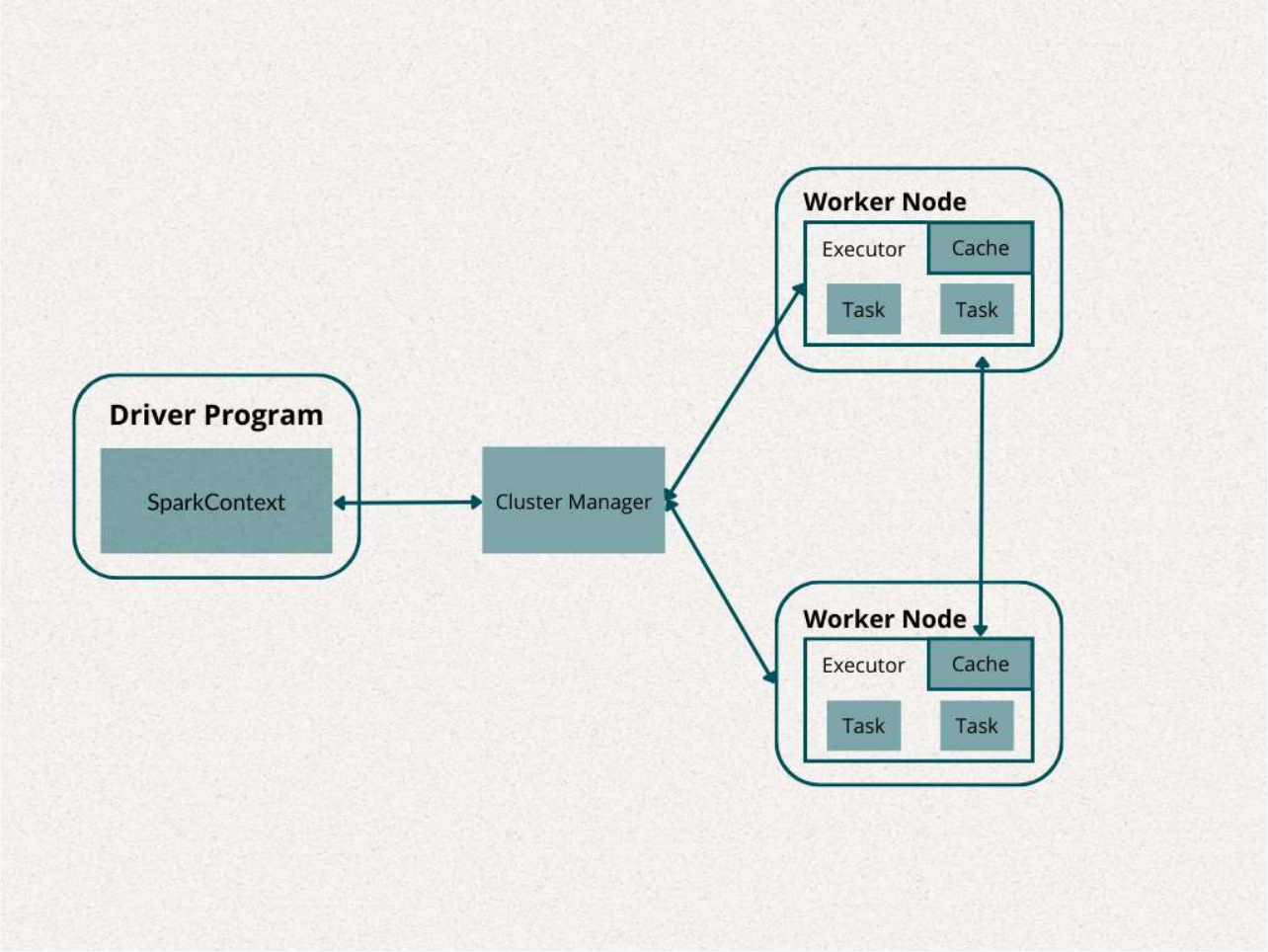
Cluster Manager

Esse é o mediador entre o *driver program* e os *worker nodes*. Ele é responsável por gerenciar e alocar recursos necessários para executar suas tarefas.

Worker Nodes

Esses são os componentes que realmente executam as tarefas (*tasks*). Cada nó (*node*) contém um ***Executor***, que é responsável por executar uma tarefa, e um ***Cache*** para armazenar dados intermediários para acesso rápido.

Figura 2. Arquitetura do *cluster* Spark



Fonte: Pereira *et al*. (2019, p. 84).

Quando o Spark é executado localmente, a máquina desempenha tanto o papel de *driver program* quanto de *worker*. O *driver program* deve apresentar somente uma instância, enquanto podemos ter várias instâncias de *workers* em execução, dentro de uma arquitetura do tipo primário-secundário (antigamente chamada de *master-slave*).

DataFrames

Legal, mas e como usamos o Spark na prática? Aqui, é importante trazermos o conceito de *DataFrames*. O **DataFrame** é uma estrutura de dados bidimensional (duas dimensões: linhas e colunas), semelhante a uma tabela de banco de dados ou uma planilha do Excel. Ele é uma das estruturas de dados fundamentais em ciência de dados e são muito usados em linguagens de programação como Python, R e Scala. Cada coluna em um DataFrame representa uma variável, e cada linha contém um conjunto de valores, um para cada coluna. Os DataFrames são ótimos para manipular, analisar e visualizar dados, pois permitem uma variedade de operações, como seleção, filtragem, agrupamento e agregação de dados.

Os DataFrames são especialmente úteis porque permitem que você trabalhe com dados de diferentes tipos (por exemplo, números, strings e datas) de uma forma que é mais estruturada e intuitiva. Também conseguimos facilmente fazer o pré-processamento de dados com DataFrames, ao processar dados faltantes, *outliers*, ou simplesmente erros de digitação nos dados. De fato, quando treinamos um novo algoritmo de *machine learning* é comum usarmos como base um DataFrame.

Existe uma biblioteca muito popular em Python que permite a manipulação de DataFrames: o ***Pandas***. O Spark também permite a manipulação de DataFrames, e possui uma performance melhor para big data. Logo, será bem interessante realizarmos um teste com o Pandas e com o PySpark.

VIDEOAULA: Carregando DataFrames com Pandas e PySpark

Nesse sentido, o que acha de usarmos o JupyterLab para criarmos um notebook responsável por processar um arquivo no HDFS e processá-lo com o PySpark? Antes disso, precisaremos carregar este arquivo. Logo, vamos também aprender um pouco sobre a manipulação de arquivos com o Pandas e o PySpark. Vamos lá?

Formatos de arquivos tabulares

Dados tabulares são aqueles que lembram tabelas ou, ainda, planilhas do Excel: são compostos de colunas e linhas, e temos valores dentro destas tabelas. Estes valores podem ser numéricos, *strings*, dados binários e datas, por exemplo. Na videoaula acima você viu um exemplo com o arquivo CSV, mas que outros formatos de arquivo trabalhamos em *big data*? Os mais comuns são JSON, CSV, Parquet e AVRO.

- CSV
- JSON
- Parquet
- AVRO

CSV

O arquivo **CSV (*Comma-Separated Values*, ou *valores separados por vírgula*)** é um tipo de arquivo que armazena dados tabulares (números e texto) em texto simples. Cada linha do arquivo é um registro de dados e cada registro consiste em um ou mais campos, separados por vírgulas. No português do Brasil, é comum separarmos estes arquivos com ponto-e-vírgula (";"), já que o separador de decimal em português é a própria vírgula.

Por ser fácil de ler, os arquivos CSV são muito usados para exportar e importar dados. Eles são amplamente suportados por muitos tipos de software, incluindo aplicativos de planilha como Microsoft Excel e Google Sheets, e linguagens de programação como Python e R. No exemplo da imagem acima, observe que a primeira linha é o cabeçalho (isto é, o nome das colunas da tabela), e cada coluna é separada por vírgula. Observe pelo esquema de cores que é possível identificarmos onde cada coluna começa e onde cada coluna termina. Em casos onde a própria vírgula é usada dentro da coluna (como o nome do autor, por exemplo), acabamos cercando os valores em aspas duplas para que o software responsável por carregar estes dados não se confunda.

Acessando dados no HDFS com PySpark

Era importante falarmos sobre alguns tipos de arquivos mais comuns para que você não interpretasse erroneamente que *tudo* é CSV. Na realidade, preferimos pelo CSV nestas nossas demonstrações porque é um tipo de arquivo mais simples e facilmente interpretável: quer dizer, você pode simplesmente abrir aquele arquivo no editor de texto para ver o seu conteúdo. Já em bases de dados maiores é mais comum trabalharmos em **parquet**, beleza?

VIDEOAULA: Trabalhando com um arquivo CSV em HDFS com PySpark

Isto posto, vamos avançar com as nossas implementações. Vamos carregar um arquivo CSV que está armazenado no HDFS via PySpark. A nossa missão é a de carregar um *DataFrame* do PySpark e testar algumas operações simples com ele.

Conclusão

Nesta unidade, trabalhamos com manipulações de arquivos usando PySpark e HDFS. Contudo, para entender melhor o contexto das coisas também explicamos sobre arquivos CSV, JSON, Parquet e Avro. Também vimos uma biblioteca do Python chamada Pandas, e testamos algumas diferenças entre o Pandas e o PySpark.

Referências

ALYX. **Como instalar o PySpark no seu computador**. 30 nov. 2021. Disponível em: https://alyx.com.br/como-instalar-o-*PySpark*-no-seu-computador. Acesso em: 26 jul. 2023.

DABRAS, T.; LEE, D. **Learning PySpark**: build data-intensive applications locally and deploy at scale using the combined powers of Python and Spark 2.0. Birmingham: Packt, 2017.

DATABRICKS. **Arquivo Avro**. Disponível em: https://learn.microsoft.com/pt-br/azure/databricks/external-data/avro. Acesso em: 23 jul. 2023.

MISHRA, R. K. **PySpark recipes**: a problem-solution approach with PySpark2. New York: Apress, 2018.

PEREIRA, M. A. *et al*. **Frameworks de big data**. Porto Alegre: SAGAH, 2019.

PY4J. **PY4J –** a bridge between Python and Java. Disponível em: https://www.py4j.org/. Acesso em: 25 jul. 2023.

SPARK BY EXAMPLES. **PySpark –** Spark context explained. Disponível em: https://sparkbyexamples.com/*PySpark*/*PySpark*-sparkcontext-explained/. Acesso em: 25 jul. 2023.