Princípios de desenvolvimento de Spark com Python

Conceitos e princípios de desenvolvimento do framework Apache Spark na linguagem Python.

Prof. Sérgio Assunção Monteiro

1. Itens iniciais

Propósito

O conhecimento teórico e prático das tecnologias de Big Data, entre as quais se destaca o framework Apache Spark, é uma necessidade para o profissional de tecnologia da informação que deseja ter domínio do assunto e, certamente, um diferencial de posicionamento em uma das áreas que mais cresce atualmente no mercado mundial.

Preparação

Para rodar os exemplos, você vai precisar criar uma conta no Google para utilizar o Google Colab.

Objetivos

- Reconhecer os conceitos do framework Apache Spark.
- Descrever a interface PySpark em Python para o Apache Spark.
- Aplicar operações de MapReduce com PySpark.
- Aplicar transformações com PySpark.

Introdução

O crescente volume de dados é uma consequência da demanda por serviços que abrangem diversos segmentos da sociedade. Esse cenário apresenta enormes oportunidades e desafios computacionais. Para atender à necessidade de trabalhar com toda essa complexidade de dados, foram desenvolvidas técnicas de computação distribuída, que utilizam arquiteturas elaboradas para otimizar os recursos computacionais e viabilizar a execução das operações sob o ponto de vista econômico.

Uma dessas técnicas de programação distribuída de maior sucesso prático foi o MapReduce. Ela foi implementada em alguns programas, com a obtenção de resultados muito promissores, mas ficou consolidada no framework Hadoop. Com o passar do tempo, foram observadas algumas limitações do Hadoop, que impulsionaram o desenvolvimento de outros projetos de programação distribuída e culminaram na criação do Spark, pela Apache Foundation.

Neste conteúdo, vamos conhecer os conceitos fundamentais do framework Apache Spark e compreender as razões de ter sido tão bem-sucedido para resolver problemas reais. Além disso, vamos desenvolver diversos exemplos práticos que nos ajudarão a consolidar os conceitos e concretizar o potencial de aplicação do Spark.

Introdução e Contextualização

Com a consolidação e evolução das tecnologias para aplicações de Big Data, houve a necessidade de ferramentas computacionais robustas e eficientes de processamento distribuído, capazes de tolerar falhas. Nesse contexto, o primeiro framework que foi um caso de sucesso foi o Hadoop, inicialmente desenvolvido pela Apache Foundation. Com o tempo, o Hadoop apresentou limitações para atender às crescentes demandas, e o próximo framework a se tornar um caso de sucesso foi o Spark.

O Apache Spark é um framework de código aberto de computação distribuída para uso geral, usado para aplicações de Big Data. O Spark 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. Ele fornece APIs de desenvolvimento para as linguagens de programação Python, Java, Scala e R. Além disso, o Spark fornece recursos para o desenvolvimento de aplicações de aprendizado de máquina, SQL, análise gráfica e bibliotecas de streaming.

> O principal objetivo do Apache Spark é trabalhar com grandes volumes de dados gerados em tempo real.

Essa tecnologia de computação em cluster é baseada no modelo MapReduce do Hadoop. Seu elemento fundamental para obter uma eficiência superior à do Hadoop é a sua estratégia de uso de memória chamada de clustering.

O Spark foi iniciado em 2009 na Universidade da Califórnia, Berkeley. Em 2010, o código foi aberto sob uma licença BSD – Berkeley Software Distribution. Em 2013, o projeto foi adquirido pela Fundação de Software Apache e, em 2014, o Spark foi promovido a um projeto Apache de importante relevância. Logo no início, os desenvolvedores trabalhavam no Hadoop e, diante de suas limitações, perceberam a necessidade de melhorar os seguintes aspectos:

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

Desse modo, o Spark se estabeleceu como um framework que tem como benefícios os seguintes itens:

Velocidade

Produz um alto desempenho para processamento de dados em lote e em fluxo (streaming).

Fácil de usar

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

Bibliotecas com diversas finalidades

Essas bibliotecas incluem SQL, funcionalidades para aprendizado de máquina, algoritmos de grafos e Spark Streaming.

Custo

O Spark é um framework de código aberto, portanto, sua licença é gratuita, assim como é o caso do Hadoop.

No entanto, para que o Spark seja utilizado na prática, são necessários equipamentos que, obviamente, têm custos associados a eles. No caso do Hadoop, seu funcionamento é baseado nas operações de uso de discos, os quais, portanto, devem ser rápidos. Já o Spark pode trabalhar com discos comuns, porém, precisa de uma grande quantidade de memória RAM, o que implica que a sua infraestrutura de hardware é mais cara do que a do Hadoop.

Aplicações 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. Agora, vamos analisar cenários que são adequados e inadequados para o uso do Spark.

Cenários adequados para usar o Spark

Agora, vamos analisar cenários que são adequados e inadequados para o uso do Spark.

Integração de dados

Consiste nos processos de extrair, transformar e carregar os dados – *Extract Transform Load* (ETL). Trata-se de uma situação rotineira em que é necessário realizar um tratamento dos dados antes de incorporá-los às bases de dados da organização. É comum que os dados originais não sejam consistentes e, por isso, o Spark pode ser usado para reduzir o custo e o tempo necessários para esse processo ETL.

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. O Spark pode fazer o processamento desses fluxos de dados e identificar potenciais operações fraudulentas.

Aprendizado de máquina

Os modelos de aprendizado de máquina funcionam melhor quando são treinados por grande volume e variedade de dados. É um cenário adequado para utilizar o Spark, pois ele é capaz de armazenar dados na memória e executar consultas eficientemente.

Análise interativa

Uma das características do Spark é a capacidade de gerar respostas rapidamente. Portanto, em vez de executar consultas predefinidas, podemos lidar com os dados de maneira interativa.

Cenários inadequados para usar o Spark

Existem algumas situações em que o uso do Spark é inadequado, sendo melhor, portanto, utilizar outra ferramenta de Big Data. Podemos conferir a seguir alguns casos em que é preferível usar outra tecnologia em vez do Spark:

Gerenciamento de cluster

Fazer o ajuste e a manutenção do Spark é uma tarefa complexa. Essa dificuldade tem um grande impacto para garantir o alto desempenho do Spark para tratar de grandes volumes de dados, que é uma situação típica para 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. Então, se o cluster ou as máquinas virtuais tiverem pouca capacidade de computação, o Spark não será uma escolha adequada. Em vez dele, pode-se utilizar o Apache Hadoop, por exemplo.

Se o Spark não for bem configurado, é muito provável que ocorram falhas de jobs com erros de falta de memória. Questões que devem ser tratadas explicitamente em uma boa configuração envolvem o tratamento de alocação de memória fixa ou dinâmica, quantos núcleos do cluster é permitido que o Spark use, qual a quantidade de memória que cada executor pode utilizar e quantas partições o Spark deve usar ao embaralhar os dados. Portanto, para utilizar o melhor do potencial do Spark, é necessário um conhecimento sólido da sua arquitetura para fazer todas essas configurações.

Arquitetura do Spark

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. A arquitetura do Spark é baseada em duas abstrações:

- Conjunto de dados distribuídos resilientes (do inglês *Resilient Dataset Distribution* – RDD).
- Gráfico Acíclico Dirigido (do inglês *Directed Acyclic Graph* – DAG).

Vamos analisar essas abstrações e, em seguida, detalhar os principais elementos da arquitetura do Spark e seus componentes.

Conjunto de dados distribuídos resiliente (RDD)

Os conjuntos de dados distribuídos resilientes, mais comumente referenciados por sua sigla em inglês RDD, são grupos de itens de dados que podem ser armazenados na memória dos nós trabalhadores. Resiliente significa que os dados são recuperados em caso de falha. Distribuído quer dizer que os dados são alocados a nós diferentes. Conjunto de dados (datasets) significa que os dados são organizados em grupos. Os RDDs suportam dois tipos de operações:

Transformações

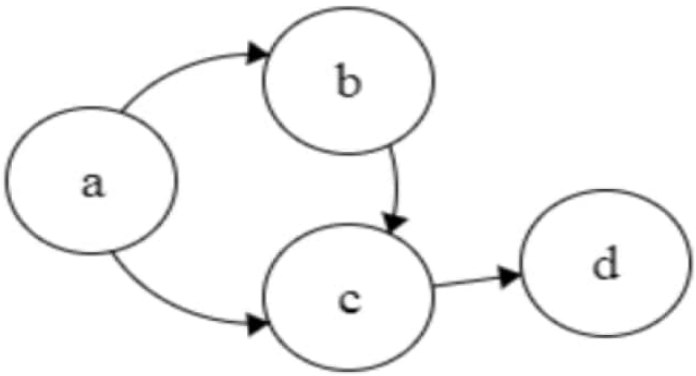
São operações como mapa, filtro, junção e união, que são executadas em um RDD e geram um novo RDD contendo o resultado.

Ações

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

Grafo Acíclico Dirigido (DAG)

Grafo é um modelo matemático formado por nós e arestas, em que as arestas com setas são as ligações entre os nós do grafo. Na figura a seguir, apresentamos um exemplo de um grafo dirigido acíclico.



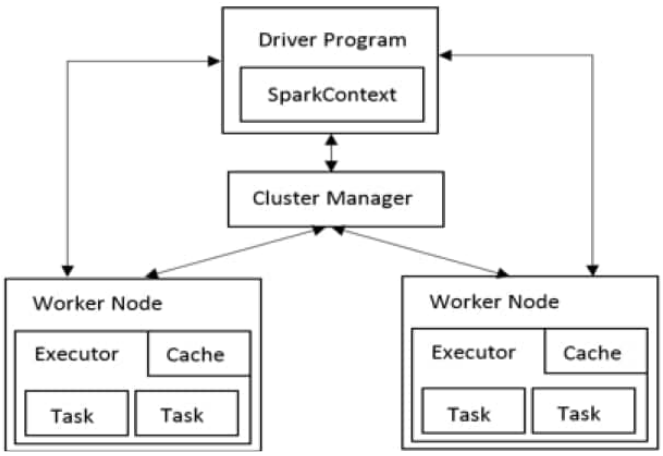
Exemplo de grafo dirigido acíclico

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 grafo acíclico direcionado, mais conhecido por sua sigla em inglês DAG, executa uma sequência de operações nos dados. Cada nó do grafo é uma partição RDD, e a aresta representa uma transformação sobre os dados. O grafo representa um mapa de navegação enquanto as propriedades de “direcionado” e “acíclico” fazem referência ao modo como a navegação é feita.

Detalhamento da Arquitetura do Spark

A arquitetura do Spark é composta por três componentes principais: o Driver Program, o Cluster Manager e os Executors. Na figura a seguir, podemos visualizar para a arquitetura básica do Spark.



Arquitetura do Spark

Elementos estruturais da Arquitetura do Spark

Agora, vamos ver com mais detalhes os elementos estruturais da arquitetura do Spark e aprender mais sobre o fluxo de execução de uma aplicação.

Driver Program

É o responsável por converter uma aplicação de usuário em unidades de execução menores chamadas de tarefas. Em seguida, essas tarefas são programadas, ou agendadas, para serem aplicadas com um gerenciador de cluster (Cluster Manager) nos executores. Cabe também ao Driver Program executar a aplicação Spark e retornar o status ou o resultado para o usuário. Para executar em um cluster, o SparkContext se conecta ao gerenciador de cluster e, em seguida, efetua as seguintes tarefas:

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

Cluster Manager ou **gerenciador de cluster**

É um componente opcional que somente é necessário se o Spark for executado de maneira distribuída. Ele é responsável por administrar as máquinas que serão utilizadas como workers. Os gerenciadores de cluster podem ser classificados em três tipos:

- Independente (standalone): Trata-se de um gerenciador de cluster simples que facilita a configuração de um cluster.
- Apache Mesos: É um gerenciador geral de cluster que também pode executar Hadoop MapReduce.
- Hadoop YARN: É o gerenciador de recursos no Hadoop.

Além desses três casos, há o suporte experimental para Kubernetes, que são uma plataforma de código aberto para fornecer infraestrutura centrada em contêineres.

Worker Nodes (nós de trabalho)

Worker Nodes (nós de trabalho): trata-se de máquinas que executarão as tarefas enviadas pelo Driver Program. Se o Spark for executado no modo local, a máquina desempenhará tanto o papel de Driver Program como de Worker. Portanto, os workers são nós escravos e sua função é executar o código do aplicativo no cluster. Além disso, os Workers são compostos por:

- Executor (Executores): São processos de nós de trabalho encarregados de executar tarefas individuais em determinado trabalho do Spark. O ciclo de vida deles começa no início de uma aplicação Spark. São executados durante toda a vida útil da aplicação. Depois de executar a tarefa, enviam os resultados ao Driver Program. Também fornecem armazenamento na memória para RDDs que são armazenados em cache por programas do usuário por meio do gerenciador de blocos (Block Manager).
- Tasks (Tarefas): As unidades de trabalho que são enviadas a um executor.

Fluxo de execução da aplicação

Agora que já conhecemos os papéis dos elementos estruturais da arquitetura do Spark, vamos analisar o ciclo de vida de uma aplicação no Spark. O fluxo de execução de uma aplicação tem os passos a seguir:

1. Quando uma aplicação inicia, o Driver Program converte o seu fluxo em um grafo acíclico direcionado (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 para outros executores para serem processadas novamente.

Ecossistema do Spark

O ecossistema do Spark é formado pelos recursos de linguagem de programação, permitindo que os projetos sejam programados em diferentes linguagens e por diferentes tipos de componentes totalmente integrados à plataforma. De maneira resumida, o Spark é uma ferramenta computacional voltada para aplicações de Big Data, em que podemos fazer o agendamento, a distribuição e o monitoramentos de várias aplicações distribuídas.

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

Scala

O framework do Spark foi desenvolvido em Scala. Isso implica que 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

A linguagem de programação R também é utilizada em contextos de ciência de dados e aprendizado de máquina.

Java

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

Agora vamos analisar os componentes do Spark. Esses componentes têm como objetivo facilitar o desenvolvimento de projetos com finalidades específicas, como os de aprendizado de máquina. Os principais componentes do ecossistema do Spark são (CHELLAPPAN; GANESAN, 2018):

Spark Core

Esse componente contém a funcionalidade básica do Spark, sendo que todas as demais bibliotecas são construídas sobre ele. 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).

Spark SQL

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

Spark Streaming

Oferece suporte ao processamento escalonável e com tolerância a falhas de dados de streaming. É usado para processar dados que são transmitidos em tempo real. Por exemplo, ocorre 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 Internet das Coisas (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 de aprendizado de máquina que contém diversos algoritmos de aprendizado de máquina, tais como correlações e testes de hipóteses, classificação e regressão, agrupamento e análise de componentes principais.

GraphX

É uma biblioteca usada para manipular e realizar cálculos paralelos aos grafos. Por meio dela, podemos visualizar dados na forma de grafo, além de fornecer vários operadores para manipulação de grafos e combiná-los com RDDs.

Instalação do Spark

Para aprendermos a instalar o Spark, vamos trabalhar com o ambiente Google Colab. Para isso, teremos de realizar a sequência de passos a seguir:

- **Instalar o Java 8.**
- **Instalar o Apache Spark com Hadoop.**
- **Instalar o Findspark.**
- **Configurar as variáveis de ambiente.**
- **Testar a instalação.**

Comentário

Para trabalhar com o ambiente Google Colab, é preciso possuir uma conta Google.

Instalar o Java 8

O Apache Spark tem como pré-requisito que o Java 8 esteja instalado. Como utilizaremos o Google Colab, então, precisamos criar uma célula de código e executar o comando a seguir:

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

Instalar o Apache Spark com Hadoop

Agora vamos instalar o Apache Spark com o Hadoop. Para isso, precisamos executar o comando a seguir:

```
python
!wget -q https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
```

Isso significa que fizemos o download do arquivo abaixo, da versão 3.1.2 do Spark e 3.2 do Hadoop.

spark-3.1.2-bin-hadoop3.2.tgz

Para escolher outras versões, basta acessarmos a página do Spark no site da Fundação Apache:

downloads.apache.org/spark

O próximo passo é descompactar o arquivo, abrindo outra célula de comando e executando a linha abaixo:

```
python
!tar xf spark-3.1.2-bin-hadoop3.2.tgz
```

Instalar o findspark

Um pouco adiante, vamos precisar utilizar o pacote PySpark que é essencial para o desenvolvimento das nossas aplicações, mas ele não está no caminho do sistema por padrão (sys.path). Então, precisamos resolver isso adicionando o PySpark ao “sys.path” em tempo de execução por meio do FindSpark. O FindSpark torna o PySpark uma biblioteca regular. Para instalarmos o FindSpark, basta executarmos o comando abaixo:

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

Instalar o pyspark

O PySpark é uma interface do Python para o Apache Spark. Nós vamos estudá-lo com mais detalhes posteriormente, mas, neste momento, precisamos saber que ele é muito importante para os nossos

programas no Spark e, para instalá-lo, basta executar a linha de comando abaixo em uma célula do Google Colab:

```
python

!pip install -q pyspark
```

Configurar as variáveis de ambiente

Depois de baixar, instalar e configurar os arquivos, pacotes e variáveis de ambiente, precisamos verificar se o Spark está funcionando corretamente. Para isso, vamos realizar os seguintes passos:

```
python

import os
 os.environ['JAVA_HOME'] = '/usr/lib/jvm/java-8-openjdk-amd64'
 os.environ['SPARK_HOME'] = '/content/spark-3.1.2-bin-hadoop3.2'
```

Na primeira linha do código, importamos o pacote “os”, que é uma interface para o sistema operacional. No caso do nosso programa, associamos as variáveis de ambiente aos caminhos em que o Java e o Spark estão instalados no Google Colab. Outro ponto de atenção está relacionado aos nomes dos arquivos. Eles precisam fazer referência às versões que baixamos, caso contrário, a configuração estará errada.

Testar a instalação

Nesse passo, teremos de configurar as variáveis de ambiente JAVA_HOME e SPARK_HOME, pois o Spark precisa acessar os recursos delas para funcionar. Para isso, precisamos escrever os comandos abaixo em uma célula de código e executá-lo:

1. Iniciar o FindSpark
2. Criar uma seção no Spark
3. Ler e exibir dados

Iniciar o FindSpark

Para iniciar o FindSpark e, posteriormente, utilizar o PySpark, precisamos escrever e executar o programa abaixo em uma célula de código do Google Colab:

```
python

import findspark
 findspark.init()
```

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

Criar uma seção no Spark

Agora vamos criar uma seção do Spark que nos permite trabalhar com RDD, Dataframe e DataSet. Para isso, precisamos escrever o código abaixo em uma célula de código do Google Colab e, em seguida, executá-lo:

```
python

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

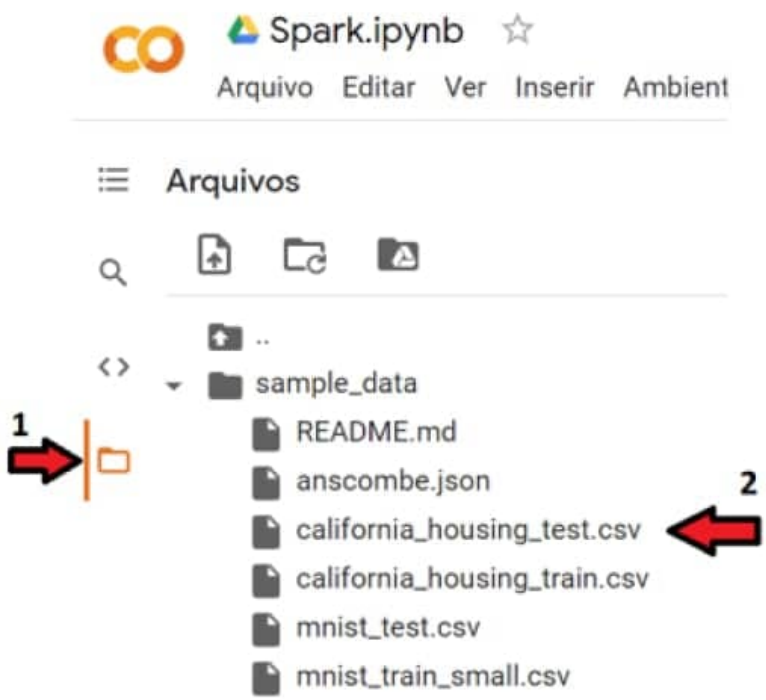
Na segunda linha, usamos o método getOrCreate para criar uma seção no Spark ou, simplesmente, obter uma seção, caso já exista.

Ainda em relação à seção do Spark, precisamos analisar o master. Existem dois casos possíveis:

- Execução no cluster: Precisamos usar um nome como yarn ou mesos como um argumento para master, dependendo da configuração do cluster.
- Execução no modo autônomo: Devemos usar **local [parâmetro]**, sendo que 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.

Ler e exibir dados

Depois de criar a seção do Spark, vamos utilizá-lo para acessarmos dados. Vamos utilizar os dados que já estão disponíveis no Google Colab, no caso, um dataset sobre residências na Califórnia. Para ver esses dados, basta acessar um símbolo de diretório no lado esquerdo do Google Colab. Na figura abaixo, podemos ver os detalhes desse processo.



Dados do Google Colab

Agora criamos outra célula de código, implementando e executando o comando abaixo:

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

Assim, temos os dados do arquivo “california_housing_test.csv” disponíveis na variável “dataset”. Para visualizar a estrutura desses dados, basta executarmos a linha abaixo em uma célula de código do Google Colab:

```python
dataset.printSchema()
```

Cuja 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, executamos o nosso exemplo completo do Spark no Google Colab.

Fechar a seção no Spark

Para concluir, devemos fechar a seção do Spark escrevendo o código abaixo em uma célula de código do Google Colab e, em seguida, executá-lo:

```python
spark.stop()
```

Visão geral dos conceitos do Spark

Neste vídeo, apresentaremos os principais conceitos sobre o Spark e sua arquitetura.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Aplicações do Spark

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Arquitetura do Spark

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

As aplicações de Big Data são dinâmicas e, cada vez, mais comuns. Então, é necessário utilizar recursos computacionais que auxiliem na criação de aplicações eficientes para atender a essa demanda, tais como frameworks e plataformas de desenvolvimento. O Spark é um framework de alto desempenho usado com sucesso exatamente para esse tipo de situação. Nesse sentido, selecione a opção correta a respeito do Spark em relação aos cenários de aplicação:

A

Para que o Spark possa ser utilizado de modo a obter alto desempenho, é 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 tratamento dos dados, situações essas que são tratadas 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, por outro lado não é adequado para operações analíticas em tempo real.

A alternativa A está correta.

O Spark é um framework para aplicações de Big Data que possui alto desempenho e, por isso, é a principal tecnologia atualmente para essa aplicação. Ele pode ser utilizado para aplicações de Internet das Coisas, Redes Sociais, portais de notícias e em muitas outras situações similares. No entanto, a sua configuração é um desafio, pois é necessário um profundo conhecimento da plataforma e da aplicação para configurar os parâmetros adequadamente.

Questão 2

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

A

Uma das abstrações da arquitetura do Spark é o conjunto de dados de distribuição resiliente, ou RDD, que viabiliza a execução paralela dos processos com tolerância a falhas.

B

A arquitetura do Spark é baseada no modelo mestre-escravo onde o mestre-componente Executor realiza o papel do mestre e o Worker Node desempenha o papel do escravo.

C

O Spark Context é o componente da arquitetura do Spark responsável por controlar a execução das tarefas.

D

O Cluster Manager é o componente da arquitetura do Spark que faz a transformação das aplicações em tarefas.

E

A arquitetura do Spark é formada por componentes que separam os processos em blocos que são tratados como tarefas pelo Spark Context.

A alternativa A está correta.

As abstrações fundamentais da arquitetura do Spark são o conjunto de dados de distribuição resiliente, ou RDD, e o grafo acíclico dirigido, DAG. O RDD é um conjunto de dados que é tolerante a falhas e pode ser executado em paralelo, sendo que todos os dados particionados em um RDD são distribuídos e imutáveis. E o DAG é a forma como as operações sobre os dados são executadas.

Introdução e Contextualização

O Apache Spark foi desenvolvido na linguagem de programação Scala. Isso significa que desenvolver aplicações em Scala no Spark é vantajoso, embora também seja possível desenvolver aplicações com outras linguagens de programação, como, por exemplo, o Python. Para podermos desenvolver com o Python no Spark, precisamos utilizar o PySpark.

Atenção

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.

Algumas das principais aplicações do PySpark é o desenvolvimento de soluções de ciência de dados e aprendizado de máquina. Isso ocorre porque o Python possui diversas bibliotecas voltadas para esse tipo de aplicação, e o ambiente do Spark favorece essas aplicações por causa da eficiência do processamento de grandes volumes de dados. Entre as principais vantagens do PySpark estão:

1. Ter um mecanismo de processamento distribuído em memória de uso geral que permite processar dados com eficiência de maneira distribuída.
2. As aplicações executadas no PySpark são muito mais eficientes do que os sistemas tradicionais.
3. Muito eficiente para pipelines de ingestão de dados.
4. Pode processar dados HDFS do Hadoop e muitos outros sistemas de arquivos.
5. Pode ser usado para processar dados em tempo real.
6. O PySpark possui bibliotecas de grafos e para aprendizado de máquina.

Utilização do PySpark

O primeiro passo para utilizar o Spark é conectar-se a um cluster. Para haver essa 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 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, os quais enviam seus resultados de volta ao mestre.

Para utilizarmos o PySpark, precisamos fazer a instalação do Spark e de suas dependências, além de configurarmos as variáveis de ambiente. Nós já fizemos esses passos na seção “Instalação do Spark”. Agora, precisamos realizar os seguintes passos:

1. Conectar-se a um cluster Spark do PySpark.
2. Realizar operações com Spark DataFrames.

Conectar-se a um cluster Spark do PySpark

Para nos conectarmos a um cluster Spark do PySpark, precisamos criar uma instância da classe SparkContext. Para isso, devemos executar estas etapas:

- Importar a biblioteca pyspark.
- Instanciar um objeto SparkContext.

Apresentamos abaixo o código que faz exatamente a sequência de passos anterior:

```
python

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

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”. Abaixo, mostramos a saída do programa:

```
python


 3.1.2
```

Realizar Operações com Spark DataFrames

A principal estrutura de dados do Spark é o RDD - *Resilient Distributed Dataset*. Essa estrutura permite que o Spark seja muito eficiente, fazendo a divisão dos dados em vários nós no cluster. Para simular o uso dos RDDs, vamos utilizar a abstração Spark DataFrame construída sobre RDDs.

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 a nossa responsabilidade realizar operações de modo eficiente, enquanto o DataFrame já faz esse processo otimizado automaticamente.

Já realizamos o primeiro passo para 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 abaixo para criar uma seção no Spark (SparkSession):

```
python

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

Na primeira linha, importamos o SparkSession a partir do pyspark.sql. Na segunda linha, criamos uma seção do Spark. Na terceira linha, imprimimos o objeto Spark. A saída do programa é dada por:

```
python
```

Essa saída nos oferece informações de que a seção foi instanciada com sucesso. Agora, vamos ler os dados de um arquivo disponível no diretório “sample_data” do Google Colab. No nosso caso, utilizamos o arquivo “california_housing_test.csv”. A seguir, podemos analisar o código de leitura dos dados do arquivo:

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

Para visualizarmos os dados de “dataset”, executamos o código a seguir, que vai exibir o cabeçalho das colunas e o conteúdo da primeira linha:

```python
dataset.head()
```

Esse código produz a seguinte saída:

```python
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)
```

Para obtermos a quantidade de linhas no dataset, basta executarmos o código abaixo:

```python
dataset.count()
```

Esse código produz a saída:

3000

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

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

A segunda linha imprime as tabelas no catálogo. A saída da execução é dada por:

```python
[Table(name='tabela_temporaria', database=None, description=None,
 tableType='TEMPORARY', isTemporary=True)]
```

O próximo passo consiste em fazer uma consulta SQL. Para isso, vamos selecionar apenas três registros com os dados referentes às colunas “longitude” e “latitude”, executando o seguinte código:

```
python

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

O resultado desse código é :

+---------+--------+

|longitude|latitude|

+---------+--------+

| -122.05| 37.37|

| -118.3| 34.26|

| -117.81| 33.78|

+---------+--------+

Utilização do PySpark com o pacote Pandas

Um dos pacotes mais importantes do Python para manipulações de dados é o Pandas, que possui funções voltadas para manipulações básicas de dados até limpeza e análise exploratória dos dados. Em especial, para aplicações de Big Data, o Pandas facilita o estudo estatístico para identificarmos padrões e descobrirmos conhecimento por meio de funcionalidades, a fim de obtermos valores máximos e mínimos de um conjunto de dados, correlações entre colunas, média, desvio-padrão e outros recursos úteis no estudo da relevância estatística dos dados.

Comentário

O Pandas facilita trabalharmos com dados relacionais de várias estruturas de dados e operações que nos ajudam a manipular dados numéricos e séries temporais.

Ele foi construído sobre a biblioteca Numpy – talvez a biblioteca 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.

Em nosso estudo, temos especial interesse no Pandas DataFrame. Trata-se de uma estrutura de dados tabular bidimensional que pode mudar o tamanho durante a execução – desde que queiramos que isso ocorra. Essa estrutura de dados também pode ser heterogênea, ou seja, os dados podem ser de tipos diferentes. As linhas e as colunas do DataFrame são rotuladas. Portanto, o Pandas DataFrame consiste em três componentes principais: **dados, linhas e 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 por conterem 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.

Converter Spark SQL para Pandas

Ainda na sequência do exemplo que já havíamos iniciado, queremos obter a localização de residências com a maior quantidade de quartos. Portanto, o resultado da consulta deve apresentar a latitude e longitude do bloco residencial com a maior quantidade de quartos.

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

1. Implementar a consulta SQL na tabela “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.

Abaixo, podemos ver o código da nossa solução que reproduz exatamente os passos que descrevemos:

```
python

query1 = 'SELECT MAX(total_rooms) as maximo_quartos FROM tabela_temporaria'
 q_maximo_quartos = spark.sql(query1)
 pd_maximo_quartos = q_maximo_quartos.toPandas()
 print('A quantidade máxima de quartos é: {}'.format(pd_maximo_quartos['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 necessária porque o retorno é um DataFrame. Depois de executar o programa, obtemos a saída abaixo:

A quantidade máxima de quartos é: 0 30450.0

Name: maximo_quartos, dtype: float64

Devemos notar 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 linha de baixo, aparece o campo “Name” com o rótulo “máximo_quartos”, nome que associamos 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 implementar os seguintes passos:

1. Implementar a consulta SQL para retornar a latitude e 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.

Abaixo, apresentamos o nosso código:

```
python

query2 = 'SELECT longitude, latitude FROM tabela_temporaria WHERE total_rooms = '+str(qtd
 _maximo_quartos)
 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, convertemos a variável “qtd_maximo_quartos” para String e, em seguida, fizemos a concatenação dela. Outro ponto que precisamos observar é a conversão do DataFrame do

Spark para o DataFrame do Pandas através da função “toPandas”. Por fim, na última linha, exibimos o resultado através da função “head” do Pandas DataFrame. A saída do programa é:

longitude latitude

0 -117.2 33.58

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

Converter Pandas DataFrame para Spark DataFrame

Agora, vamos estudar um exemplo que converte um DataFrame do Pandas para um DataFrame no Spark. O nosso exemplo é composto dos seguintes passos:

1. Geração de dados aleatórios que seguem a distribuição normal com média e desvio-padrão que nós fornecemos. Usamos a biblioteca Numpy para gerar os dados e a 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.

Abaixo, apresentamos o código:

```
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:

[Table(name='tabela_temporaria', database=None, description=None, tableType='TEMPORARY', isTemporary=True), Table(name='temp', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

São listadas apenas as tabelas que já estavam no catálogo. Depois de executar a função “createOrReplaceTempView”, obtemos a seguinte saída:

[Table(name='nova_tabela_temporaria', database=None, description=None, tableType='TEMPORARY', isTemporary=True), Table(name='tabela_temporaria', database=None, description=None, tableType='TEMPORARY', isTemporary=True), Table(name='temp', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

Essa saída inclui informações sobre a última tabela que criamos.

Para concluir, precisamos fechar a seção usando o seguinte código:

```
python

spark.stop()
```

Dicas

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, é mais fácil detectar e corrigir.

Uma fonte de erros comum é não observar a indentação 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. Quando rodarmos os exemplos, não podemos nos esquecer de instalar os pacotes e as dependências, exatamente como foi apresentado.

Utilização do PySpark

Neste vídeo, apresentamos os conceitos fundamentais do PySpark e alguns exemplos práticos.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Utilização do PySpark

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Utilização do PySpark com o pacote Pandas

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

Uma das vantagens de trabalhar com o Spark é que ele oferece suporte para diversas linguagens de programação. Uma dessas linguagens é o Python, que pode ser acessado pela AP PySpark. Essa API possui diversas características que precisam ser compreendidas para desenvolver aplicações de qualidade do Python no Spark. Nesse sentido, selecione a opção 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

Um dos limitantes 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.

A alternativa E está correta.

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

Questão 2

A linguagem de programação Python é utilizada em diversos contextos de aplicação. Em especial, ela é uma referência para aplicações de Ciência de Dados e de Aprendizado de Máquina que, por sua vez, tem 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, selecione a opção 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 por meio do uso direto dos RDDs com o PySpark.

C

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

D

Os DataFrames são um recurso do Python que permitem trabalhar com dados heterogêneos que facilita 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.

A alternativa D está correta.

Os DataFrames são estruturas de dados do Python que pertencem à biblioteca Pandas, muito utilizada para manipular dados heterogêneos, 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.

Introdução e Contextualização

Atualmente, usamos diversas aplicações distribuídas em nosso dia a dia. Alguns exemplos são serviços na internet, computação móvel e Internet das Coisas. Com a popularização da internet, as aplicações distribuídas ganharam protagonismo, pois começaram a surgir diversas demandas que não precisavam, nem podiam passar por uma abordagem sequencial. Desse modo, 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**.

Divisão

Na etapa de divisão, os blocos de dados são separados em pequenas partes gerenciáveis.

Conquista

Em seguida, vem a etapa da conquista, na qual 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, introduzido pela empresa Google por volta de 2004 (HAMSTRA; KARAU; ZAHARIA; KONWINSKI; WENDELL, 2015). A parte de divisão é chamada de ação **Mapeamento**, e a recombinação é chamada de ação **Redução**.

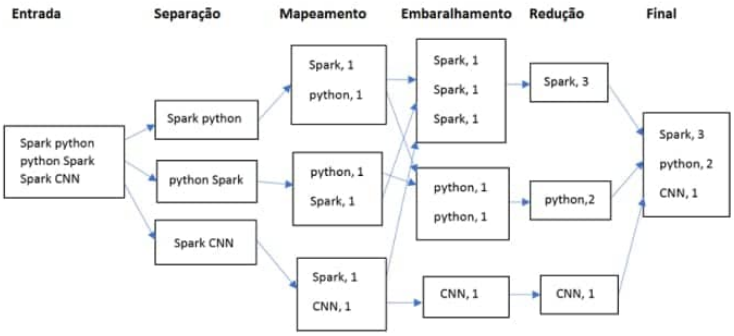
> O MapReduce é uma estratégia de computação com capacidade de processar grandes conjuntos de dados, de maneira distribuída, em várias máquinas.

A 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 processamento é muito eficiente para aplicações distribuídas, pois quase todos os dados podem ser mapeados em pares (chave, valor). Importante ressaltar que chaves e valores podem ser de qualquer tipo: strings, inteiros, tipos abstratos e até os próprios pares (chave, valor).

Um exemplo clássico para utilização de MapReduce é a contagem da frequência de palavras em um texto grande. Essa estratégia pode ser utilizada para fazer classificações e pesquisas distribuídas, aprendizado de máquina e em muitas outras situações práticas que apenas confirmam a importância dessa técnica utilizada pelo Spark.

Paradigma MapReduce

Como vimos, o MapReduce pode ser usado para 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 e, no final, o algoritmo produz uma lista com as palavras e a quantidade de vezes que elas apareceram. Na figura abaixo, podemos ver um exemplo de como o MapReduce trabalha no exemplo de contagem de palavras.



Exemplo de MapReduce Elaborado por: Sergio Assunção Monteiro

Analisando o esquema, observamos algumas etapas distintas:

Entrada de dados

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 maneira independente ao longo das fases de processamento até que o processo conclua.

Separação

Os dados são separados em listas, que serão processadas de modo independente.

Mapeamento

A ideia dessa etapa é aplicar uma função a cada elemento de uma lista e coletar o resultado. Em nosso exemplo, o objetivo é mapear cada palavra no arquivo de entrada em um par chave e valor, considerando que a chave é a palavra e que o número de ocorrências é o valor.

Embaralhamento

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

Redução

Precisamos apenas contar 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

O processo apresenta as palavras com as suas respectivas frequências de repetição.

A seguir, vamos analisar alguns exemplos práticos de implementações de MapReduce com o PySpark.

Exemplos Práticos

Depois de termos conhecido os aspectos teóricos do paradigma MapReduce para programação distribuída, vamos estudar alguns exemplos práticos. Cada um desses exemplos corresponde a uma unidade lógica que deve ser executada em uma célula do Google Colab.

Antes de implementar o exemplo, precisamos instanciar uma seção. Para isso, precisamos escrever o código abaixo:

```
python

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

Exemplo 1 - Vetores

O objetivo desse exemplo é receber um vetor numérico e, para cada elemento do vetor, aplicar a seguinte função matemática:

$$f(x) = x^2 + x$$

Para criar o vetor, vamos utilizar o pacote Numpy, importado pelo seguinte código:

```
python

import numpy as np
```

Em seguida, entramos com os dados do vetor, conforme o código abaixo:

```
python

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

Agora vamos criar um RDD por meio de um SparkContext com o seguinte código:

```
python

paralelo = spark_contexto.parallelize(vetor)
```

O código a seguir nos ajuda a fazer uma rápida verificação do conteúdo da variável “paralelo”:

```
python

print(paralelo)
```

Que produz a seguinte saída:

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

Essa saída indica que criamos o RDD com sucesso. Logo, o vetor já está no Spark e podemos aplicar o Mapeamento. Para isso, vamos usar o código abaixo:

```
python

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

lambda x : x**2+x é uma função lambda que corresponde 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.

Nosso próximo passo é coletar os dados, ou seja, verificar o resultado. Fazemos isso usando o código abaixo:

```
python

mapa.collect()
```

Esse código produz exatamente a saída que queremos, sendo ela:

[110, 420, 930, 1640, 2550]

Exemplo 2 - Listas

Nesse exemplo, o objetivo é contar a frequência das palavras de uma lista. Então, o primeiro passo é entrar com uma lista de palavras, conforme o código abaixo:

```
python

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

Com isso, a variável “paralelo” faz referência ao RDD. O próximo passo é implementar uma função lambda que simplesmente associa o número “1” a uma palavra. O código fica exatamente assim:

```
python

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

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

Em seguida, vamos aplicar o MapReduce, mas primeiro vamos analisar o código a seguir:

```
python

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

Na primeira linha, importamos o operador “add” que será usado para somar as ocorrências das palavras mapeadas. Já na segunda linha, muitas coisas estão ocorrendo:

1. Fazemos o mapeamento dos dados da variável “paralelo” para a função lambda, ou seja, para cada palavra criamos um par (palavra, 1).
2. Em seguida, aplicamos a redução com a função “reduceKey”, que soma as ocorrências e as agrupa pela chave, no caso, pelas palavras.

3. Na etapa final, dada pela função “collect”, fazemos a coleta dos dados em uma lista que chamamos de “mapa”.

Para visualizarmos o resultado, usamos o código abaixo:

```
python

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. Abaixo, apresentamos a saída:

distribuida: 2

spark: 3

rdd: 1

Para concluir, precisamos fechar a seção com o seguinte código:

```
python

spark_contexto.stop()
```

Operações de MapReduce com PySpark

No vídeo a seguir, apresentaremos os conceitos do paradigma MapReduce no PySpark com exemplos práticos.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Paradigma MapReduce

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Aspectos de Programação Funcional

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

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, selecione a opção correta a respeito da computação distribuída no contexto do Spark:

A

O Spark utiliza a técnica de dividir para conquistar para processar grandes volumes de dados, assim obtém um alto desempenho.

B

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

C

A fase de redução consiste na separação dos dados que, 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, desse modo, eles podem ser particionados em pares chave e valor e, em seguida, tratados na fase de redução.

A alternativa B está correta.

O MapReduce é uma técnica fundamental na computação distribuída em que os dados são submetidos a duas etapas: a de mapeamento e a de redução. Na fase de mapeamento, são gerados pares chave e 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.

Questão 2

O MapReduce é uma técnica clássica de computação distribuída. Ela é utilizada por vários frameworks, inclusive pelo Spark. O MapReduce, apesar de clássico, é uma técnica muito eficiente para processamento distribuído e é composta por diversas etapas. Nesse sentido, selecione a opção correta a respeito das etapas do MapReduce:

A

A fase de entrada de dados consiste no tratamento dos dados de modo a evitar que as próximas etapas sejam sobrecarregadas com a detecção de inconsistências nos dados que poderia levar a dificuldades do processamento.

B

A etapa de mapeamento é a parte mais importante do MapReduce, pois direciona como os dados serão separados e, posteriormente, agrupados para obter o resultado.

C

Na fase de embaralhamento, os pares de dados com a mesma chave são agrupados para serem processados posteriormente pela etapa de redução.

D

Quando o processamento dos dados se aproxima da fase final, passam pela etapa de redução que consiste na contagem de ocorrências de cada par com a mesma chave.

E

Na fase de separação, os dados são separados em grupos de pares com a mesma chave que serão processados pela etapa seguinte de redução.

A alternativa C está correta.

A técnica MapReduce é composta pelas etapas de entrada dos dados, separação, mapeamento, embaralhamento, redução e final. Cada uma dessas etapas é responsável pela computação dos dados, de modo a potencializar a obtenção do processamento distribuído. A ideia básica consiste em separar os dados em pares chave e valor e, em seguida, agrupá-los de tal modo que o processamento de cada etapa seja eficiente. No caso específico da fase de embaralhamento, os pares são agrupados pelos dados com a mesma chave.

Introdução e Contextualização

A estrutura de dados fundamental do Apache Spark é o RDD – conjunto de dados distribuídos resilientes. Como já vimos, o alto desempenho do Spark está diretamente vinculado ao funcionamento do RDD. Os dados no Spark RDD são particionados logicamente entre diversos servidores, para que possam ser calculados em diferentes nós do cluster. Devido à importância do RDD, vamos analisar algumas das suas principais características:

Computação em memória

Podemos armazenar dados no Spark RDD. Esse armazenamento independe do volume dos dados. Isso significa que podemos operar com dados informações na memória de acesso rápido.

Avaliações preguiçosas

Significa que, ao chamar alguma operação no Spark, o processamento não inicia de imediato. Para que uma execução inicie, é necessário realizar uma ação. Logo que essa ação ocorre, os dados dentro do RDD não podem ser transformados ou disponibilizados.

Tolerância a falhas

Os Spark RDDs rastreiam informações para reconstruir, automaticamente, dados perdidos em caso de falha.

Imutabilidade

Isso significa que os RDDs não se modificam ao longo do tempo. Essa propriedade garante a consistência dos dados, quando executamos outras operações. Devido ao fato de não podermos fazer alterações no RDD depois de criado, significa que ele só pode ser transformado em novos RDDs. Para isso, aplicamos os processos de transformação que detalharemos mais adiante.

Particionamento

Todos os conjuntos de dados são particionados logicamente e distribuídos entre os nós do cluster. O objetivo desse particionamento é otimizar o processamento de modo a obtermos paralelismo.

Persistência

Existem duas opções de armazenamento dos dados, persistência em memória RAM ou disco. A melhor opção para otimizar desempenho é armazenar os dados na memória RAM devido à sua alta velocidade de acesso. Além disso, temos a possibilidade de extrai-los diretamente da memória. Isso faz com que os RDDs sejam úteis para cálculos rápidos. Portanto, podemos realizar várias operações nos mesmos dados e reutilizá-los.

Operações de granulação grossa

Significa que podemos realizar uma operação em todo o cluster de uma vez por meio de mapas ou filtros.

Fixação de localização

Os RDDs têm a capacidade de definir a preferência de posicionamento para computar partições, para que a tarefa fique perto dos dados o máximo possível e, assim, acelere a computação.

O Spark possui um conjunto de operações que podem ser executadas em RDD. Uma operação, basicamente, é um método que pode ser aplicado em um RDD para realizar determinada tarefa. O RDD suporta dois tipos de operações:

Transformação

São as operações aplicadas ao RDD para criar um RDD.

Ações

São as operações aplicadas ao RDD com o objetivo de o Spark realizar cálculos e enviar os resultados de volta ao Driver Program.

Por padrão, todas as transformações no RDD são “preguiçosas” – do inglês *lazy*. Isso quer dizer que os cálculos no RDD não serão feitos até que apliquemos uma ação. Como vimos, uma das propriedades dos RDDs é a imutabilidade. Embora não sejamos capazes de alterar o RDD, podemos transformá-lo, ou seja, podemos criar outro RDD e aplicar uma ação.

Transformações

As transformações do Spark RDD são funções que recebem um RDD como entrada e produzem um ou mais RDDs como saída. Eles 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, elas são operações preguiçosas (*lazy evaluations*), ou seja, 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 vai criar um DAG (Grafo Dirigido Acíclico), com as seguintes informações: operação aplicada, a RDD origem e a função usada para a transformação. Apenas quando aplicarmos uma ação, serão realizados os cálculos sobre o RDD. Existem dois tipos de transformações: transformação estreita e transformação ampla. A seguir, vamos analisar cada uma dessas transformações.

Transformações estreitas

São transformações 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 do pipelining é fazer um processamento com a maior quantidade de operações em uma única partição de dados. Como exemplos de funções de transformações estreitas, podemos citar:

map()

É usada para aplicar a transformação em cada elemento de uma fonte de dados e retorna um novo conjunto de dados. Os dados 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.

mapPartitions():

É parecida com a função map(). No caso, executa a função de transformação em cada partição de dados.

sample()

Retorna um subconjunto aleatório dos dados de entrada.

filter

Faz uma filtragem dos registros de uma fonte de dados.

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. Nessas situações, é necessário realizar movimentos de dados entre as partições para executar as transformações. Por isso, são chamadas de transformações amplas. Elas também são conhecidas como transformações aleatórias, pois podem depender de uma mistura aleatória. Como exemplos de funções de transformações amplas, podemos citar:

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()

Também opera sobre um conjunto de dados de pares (K, V) e retorna um conjunto de dados de pares (K, V) em que 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. Usando isso, podemos diminuir o número de partições.

Ações

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. Alguns exemplos de ações são:

first():

Retorna o primeiro elemento no RDD.

take()

Retorna um vetor com os primeiros n elementos do conjunto de dados, sendo que n é um parâmetro da função.

reduce()

Faz a agregação dos 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.

Um exemplo prático

Agora, vamos desenvolver um exemplo que envolve as operações de transformações e de ações. O primeiro passo é instanciar um SparkContext, utilizando o código abaixo:

```
python

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

Agora vamos fornecer uma lista de entrada e transformá-la em um RDD:

```
python

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

Em seguida, vamos executar a ação de contar os elementos do RDD. Para isso, devemos implementar o seguinte código:

```
python

lista_rdd.count()
```

Essa ação vai produzir na saída o valor: 6.

Agora, vamos criar uma função lambda que recebe um número como parâmetro e retorna um par formador pelo número do parâmetro e pelo mesmo número multiplicado por 10. Abaixo, apresentamos o código:

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

Nesse próximo passo, vamos aplicar a transformação “flatMap” com a ação “collect” da função lambda para a “lista_rdd”:

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

Esse código vai produzir a saída:

[1, 10, 2, 20, 3, 30, 4, 40, 5, 50, 3, 30]

Agora, vamos aplicar a transformação “map” com a ação “collect” da função lambda para a “lista_rdd”:

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

Que vai produzir a saída:

[(1, 10), (2, 20), (3, 30), (4, 40), (5, 50), (3, 30)]

Para concluir, precisamos fechar a seção com o código abaixo:

```
python
spark_contexto.stop()
```

Transformações com PySpark

Neste vídeo apresentamos os conceitos sobre Transformações com PySpark.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vem que eu te explico!

Os vídeos a seguir abordam os assuntos mais relevantes do conteúdo que você acabou de estudar.

Transformações

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Ações

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Verificando o aprendizado

Questão 1

O Apache Spark é um framework de código aberto aplicado para projetos de Big Data. Atualmente, é o mais bem-sucedido framework para Big Data. Isso ocorre por várias características estruturais para armazenar e processar os dados. Em relação ao armazenamento de dados, o Spark utiliza o RDD. Nesse sentido, selecione a opção correta a respeito do RDD do Spark:

A

Uma das limitações da utilização dos RDDs é que somente é possível realizar operações parciais nos clusters, sendo necessário fazer o particionamento dos dados para obter o resultado.

B

Os dados disponíveis nos RDDs são acessados sequencialmente de modo a otimizar a tolerância a falhas e garantir a consistência dos resultados.

C

Devido ao grande de volume de dados que uma aplicação pode trabalhar, o Spark faz uma organização eficiente dos dados por meio de uma combinação dos dados entre as memórias primária e secundária.

D

Os RDDs são caracterizados como estruturas de dados dinâmicas que podem sofrer alterações ao longo do processamento de modo a otimizar o desempenho da aplicação.

E

Uma das características do Spark é o processamento preguiçoso que, ao iniciar, indisponibiliza os dados do RDD para outras aplicações.

A alternativa E está correta.

O RDD é a unidade estrutural fundamental do Spark e é por meio dela que as operações podem ser executadas. Entre suas características estão a computação em memória, as avaliações preguiçosas, a tolerância a falhas, a imutabilidade, o particionamento e a persistência, entre outras. Cada uma delas fornece recursos para que o Spark utilize os dados de modo a otimizar o processamento e garantir a consistência deles ao longo da execução do processo.

Questão 2

As aplicações de Big Data demandam técnicas específicas de tratamento. Além do grande volume e da variedade de dados, existem questões associadas à periodicidade com que esses dados são gerados e à expectativa de tempo que devem ser processados. Nesse sentido, o Spark utiliza duas categorias de operações que se complementam para processar os dados: as transformações e as ações. Selecione a opção correta a respeito das categorias transformações e ações 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, que são responsáveis por criar grupos de pares de chave e valor que serão processados posteriormente pelas transformações.

C

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

D

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

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.

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. No caso das transformações, elas não executam o processamento imediato dos dados, mas criam dados a partir de RDDs existentes que serão utilizados posteriormente pelas ações. Esse processo é chamado de processamento preguiçoso.

Considerações finais

Ao longo do texto, estudamos sobre o framework Spark. Vimos que o conhecimento dos conceitos do Spark é fundamental para entendermos o seu alto desempenho para operar com aplicações de Big Data.

Estudamos aspectos práticos do Spark por meio do PySpark, uma API do Spark que viabiliza o desenvolvimento de projetos na linguagem de programação Python. Vimos, ainda, aplicações de MapReduce para manipularmos conjuntos de dados por meio de programação distribuída.

As aplicações de Big Data estão espalhadas por diversas áreas da sociedade e, as que ainda não estão mapeadas, farão parte desse ecossistema, pois a integração é uma necessidade. Uma das consequências diretas da análise desse cenário é que há muitas oportunidades para desenvolver aplicações para atender a essa demanda. Portanto, investir no conhecimento dessa área é uma ótima escolha que pode criar muitas oportunidades de desenvolvimento profissional.

Podcast

Neste podcast, apresentaremos os principais fundamentos de desenvolvimento de Spark com Python.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Para aprofundar os seus conhecimentos no conteúdo estudado:

Acesse o site oficial do Spark e aprenda mais sobre a arquitetura e os componentes do ecossistema dele.

Acesse o site oficial do Pandas com PySpark, no qual você vai encontrar diversos exemplos práticos que vão ajudá-lo a solidificar os conceitos.

Referências

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

CLUSTER Mode Overview. Consultado na Internet em: 10 out. 2021.

GOOGLE Colab. Consultado na Internet em: 10 out. 2021.

HAMSTRA, M.; KARAU, H.; ZAHARIA, M.; KONWINSKI, A.; WENDELL, P.: **Learning Spark:** Lightning-Fast Big Data Analytics. OReilly Media, 2015.