

Entendendo o Big Data

(baseado em Guller (2015) - Big Data Analytics with Spark A Practitioner's Guide to Using Spark for Large-Scale Data Processing, Machine Learning, and Graph Analytics, and High-Velocity Data Stream Processing)



Falando sobre o Cenário de Big Data - HDFS (Hadoop Distributed File System)

HDFS, como o nome indica, é um sistema de arquivos distribuído. Ele armazena um arquivo em um cluster de servidores.

Ele foi projetado para armazenar e fornecer acesso rápido a grandes arquivos e grandes conjuntos de dados. É escalonável e tolerante a falhas.



O que é o HDFS

HDFS é um sistema de arquivos estruturado em blocos.

Assim como os sistemas de arquivos Linux, o HDFS divide um arquivo em blocos de tamanho fixo, também conhecidos como partições ou divisões.

O tamanho do bloco padrão é 128 MB, mas é configurável.

Deve ficar claro a partir do tamanho dos blocos que o HDFS não foi projetado para armazenar arquivos pequenos. Se possível, o HDFS espalha os blocos de um arquivo em diferentes máquinas.

Portanto, um aplicativo pode paralelizar as operações de leitura e gravação em nível de arquivo, tornando muito mais rápido ler ou gravar um arquivo HDFS grande distribuído por vários discos em computadores diferentes do que ler ou gravar um arquivo grande armazenado em um único disco.

Características e Arquitetura do HDFS

Distribuir um arquivo para várias máquinas aumenta o risco de um arquivo ficar indisponível se uma das máquinas em um cluster falhar.

O HDFS reduz esse risco replicando cada bloco de arquivo em várias máquinas.

O fator de replicação padrão é 3.

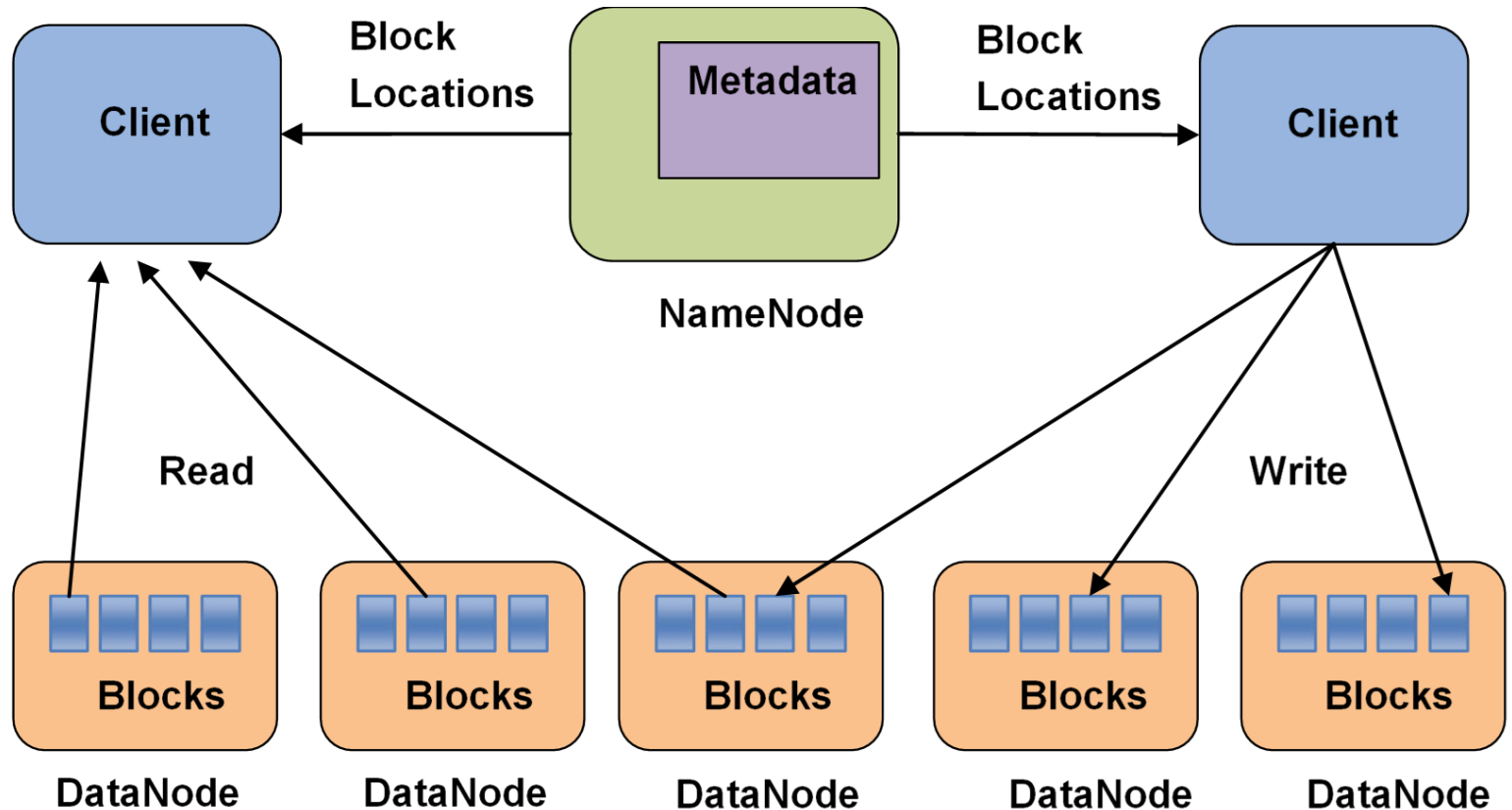
Portanto, mesmo se uma ou duas máquinas que atendem a um bloco de arquivos falharem, esse arquivo ainda poderá ser lido. O HDFS foi projetado com a suposição de que as máquinas podem falhar regularmente. Portanto, ele pode lidar com a falha de uma ou mais máquinas em um cluster.

Um cluster HDFS consiste em dois tipos de nós: NameNode e DataNode (veja a Figura 3).

Um NameNode gerencia o namespace do sistema de arquivos. Ele armazena todos os metadados de um arquivo. Por exemplo, ele rastreia nomes de arquivo, permissões e locais de bloqueio de arquivo. Para fornecer acesso rápido aos metadados, um NameNode armazena todos os metadados na memória.

Um DataNode armazena o conteúdo real do arquivo na forma de blocos de arquivo.

Arquitetura HDFS



Detalhes da Arquitetura HDFS

O NameNode recebe periodicamente dois tipos de mensagens dos DataNodes em um cluster HDFS.

Um é chamado de Heartbeat e o outro é Blockreport.

Um DataNode envia uma mensagem de pulsação para informar ao NameNode que está funcionando corretamente.

Um Blockreport contém uma lista de todos os blocos de dados em um DataNode.

Quando um aplicativo cliente deseja ler um arquivo, ele primeiro contata um NameNode.

O NameNode responde com as localizações de todos os blocos que compõem aquele arquivo.

Um local de bloco identifica o DataNode que contém dados para aquele bloco de arquivo.

Um cliente então envia diretamente uma solicitação de leitura aos DataNodes para cada bloco de arquivo.

Um NameNode não está envolvido na transferência de dados real de um DataNode para um cliente.

Mais Detalhes da Arquitetura HDFS

Da mesma forma, quando um aplicativo cliente deseja gravar dados em um arquivo HDFS, ele primeiro contata o NameNode e pede a ele para criar uma nova entrada no namespace HDFS.

O NameNode verifica se já existe um arquivo com o mesmo nome e se o cliente tem permissão para criar um novo arquivo. Em seguida, o aplicativo cliente pede ao NameNode para escolher DataNodes para o primeiro bloco do arquivo.

Ele cria um pipeline entre todos os nós de réplica que hospedam esse bloco e envia o bloco de dados para o primeiro DataNode no pipeline.

O primeiro DataNode armazena o bloco de dados localmente e o encaminha para o segundo DataNode, que o armazena localmente e encaminha para o terceiro DataNode.

Depois que o primeiro bloco de arquivo foi armazenado em todos os DataNodes atribuídos, o cliente pede ao NameNode para selecionar os DataNodes para hospedar réplicas do segundo bloco.

Este processo continua até que todos os blocos de arquivo tenham sido armazenados nos DataNodes.

Finalmente, o cliente informa ao NameNode que a gravação do arquivo foi concluída.

MapReduce

MapReduce é um mecanismo de computação distribuído fornecido pelo Hadoop.

Enquanto o HDFS fornece um sistema de arquivos distribuído para armazenar grandes conjuntos de dados, o MapReduce fornece uma estrutura de computação para processar grandes conjuntos de dados em paralelo em um cluster de computadores.

Ele abstrai a computação em cluster e fornece construções de alto nível para escrever aplicativos de processamento de dados distribuídos.

Ele permite que programadores sem experiência em escrever aplicativos distribuídos ou paralelos escrevam aplicativos que podem ser executados em um cluster de computadores comuns.



Estrutura do MapReduce

A estrutura MapReduce programa automaticamente a execução de um aplicativo em um conjunto de máquinas em um cluster. Ele lida com balanceamento de carga, falhas de nó e comunicação complexa entre nós. Ele cuida dos detalhes complicados da computação distribuída e permite que um programador se concentre na lógica de processamento de dados.

Os blocos de construção básicos de um aplicativo MapReduce são duas funções: mapear e reduzir. Ambos os primitivos são emprestados da programação funcional.

Todos os trabalhos de processamento de dados em um aplicativo MapReduce são expressos usando essas duas funções.

A função de Map recebe como entrada um par de valores-chave e produz um conjunto de pares de valores-chave intermediários.

A estrutura MapReduce chama a função de map uma vez para cada par de valores-chave no conjunto de dados de entrada. Em seguida, ele classifica a saída das funções de mapa e agrupa todos os valores intermediários associados à mesma chave intermediária. Em seguida, ele os passa como entrada para a função de redução.

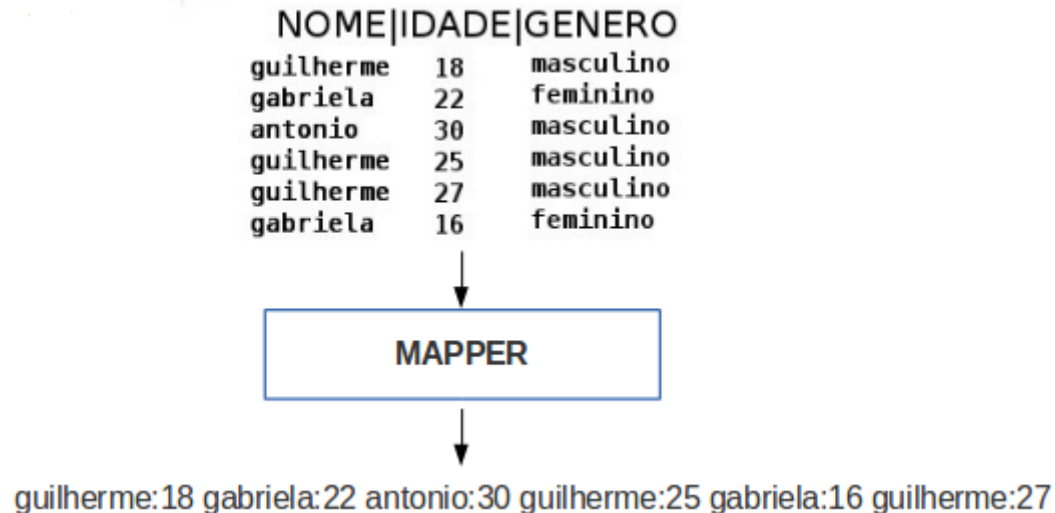
A função Reduce agrega esses valores e produz o valor agregado junto com a chave intermediária que recebeu como sua entrada.

O Spark, que estudamos neste curso, é considerado um sucessor do MapReduce.

Ele oferece muitas vantagens em relação ao MapReduce. Isso é discutido mais a frente.

A Primeira Camada do Mapreduce - Map

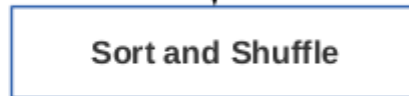
Primeiramente temos o Input(dados os de entrada) como podemos ver na imagem acima, logo após o Input temos o Mapper, que separa os dados da entrada em chave:valor.Os Mappers são distribuídos entre as máquinas do cluster, para o processamento ser realizado mais rápido.



Sort and Shuffle

Logo após temos o Sort and Shuffle, primeiro vem o Sort que organiza os dados em ordem numérica ou alfabética, após o Sort organizar os dados o Shuffle entra em ação realizando um "Group By" dos dados como por exemplo: chave:valor1,valor2,valor3

guilherme:18 gabriela:22 antonio:30 guilherme:25 gabriela:16 guilherme:27

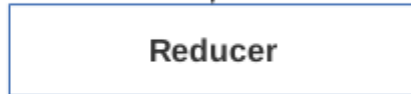


antonio:30 gabriela:22,16 guilherme:18,25,27

A Segunda Camada do Mapreduce - Reduce

Após o Sort and Shuffle entramos na fase final do processamento, que é o Reducer, que consiste em reduzir os dados que o Shuffle produziu para apenas um único valor (esse valor consiste na quantidade de vezes que o valor está se repetindo no Shuffle), como por exemplo: Shuffle: chave:valor1,valor1,valor Reducer: chave:3

antonio:30 gabriela:22,16 guilherme:18,25,27



antonio:1 gabriela:2 guilherme:3

Hive (Colmeia)

Hive é um software de armazenamento de dados que fornece uma linguagem semelhante a SQL para processar e analisar dados armazenados em HDFS e outros sistemas de armazenamento compatíveis com Hadoop, como Cassandra e Amazon S3.

Embora o Hadoop tenha facilitado a escrita de aplicativos de processamento de dados que podem utilizar os recursos em um cluster de computadores, o pool de programadores que podem escrever tais aplicativos ainda é muito menor em comparação com o pool de pessoas que conhecem SQL.



E o que é SQL

SQL é uma das linguagens de processamento de dados mais amplamente utilizadas. É uma linguagem declarativa. Parece extremamente simples, mas é uma linguagem poderosa.

SQL é mais fácil de aprender e usar do que Java e outras linguagens de programação usadas para escrever um aplicativo MapReduce. O Hive trouxe a simplicidade do SQL para o Hadoop e o tornou acessível a uma base de usuários mais ampla.



mais sobre SQL neste [link \(https://www.tecmundo.com.br/software/146482-sql-que-ele-serve.htm\)](https://www.tecmundo.com.br/software/146482-sql-que-ele-serve.htm).

Voltando ao Hive

O Hive fornece uma linguagem de consulta semelhante a SQL chamada Hive Query Language (HiveQL) para processar e analisar dados armazenados em qualquer sistema de armazenamento compatível com Hadoop.

Ele fornece um mecanismo para projetar uma estrutura em dados armazenados em HDFS e consultá-los usando HiveQL.

Ele traduz consultas HiveQL em trabalhos MapReduce.

Ele também oferece suporte a UDFs (funções definidas pelo usuário) e UDAFs (funções agregadas definidas pelo usuário), que podem ser usados para processamento de dados complexos que não podem ser expressos com eficiência em HiveQL.

O Spark SQL, é considerado um sucessor do Hive. No entanto, o Spark SQL oferece mais do que apenas uma interface SQL.

**Na proxima aula continuamos com explicando
conceitos e e mais componentes do cenário Big Data**



<https://hackersandslackers.com/learning-to-use-apache-spark-pyspark>
(<https://hackersandslackers.com/learning-to-use-apache-spark-pyspark>).

Aprendendo Apache Spark com PySpark e Databricks

Algo que apenas começamos a abordar até agora é o benefício de utilizar o Apache Spark em pipelines de dados em maior escala. O Spark é uma parte essencial da pilha de dados do Apache: construído sobre o Hadoop, o Spark destina-se a lidar com tarefas que consomem muitos recursos, como streaming de dados e processamento de gráfico.

Muito do fascínio do Spark vem do fato de ele ser escrito em Scala e Java. Java e suas linguagens derivadas são notórias por rodar com muita memória em tempo de execução, o que pode ser usado a nosso favor. Como nossos jobs tornam-se previsivelmente intensivos em recursos, já que tudo é armazenado na memória, isso nos permite dimensionar nossos Spark Clusters para acomodar adequadamente.

Sobre Spark vs. Hadoop

É crucial para nós entendermos onde o Spark se encaixa no grande ecossistema Apache. Tecnicamente, o Spark é construído sobre o Hadoop: o Spark pega muito emprestado do sistema de arquivos distribuído do Hadoop, portanto, comparar "Spark vs. Hadoop" não é uma comparação precisa de 1 para 1. Em vez disso, é melhor pensar no Spark como uma alternativa ao MapReduce do Hadoop.

Quando comparado ao MapReduce do Hadoop, o Spark é executado mais rápido. O Spark armazena DataFrames na memória até que seja informado o contrário, dando a ele um bônus de velocidade em relação ao MapReduce, que grava no disco. Claro, o Spark vem com o bônus de ser acessível por meio da biblioteca Python do Spark: [PySpark](https://spark.apache.org/docs/2.3.1/api/python/index.html).
(<https://spark.apache.org/docs/2.3.1/api/python/index.html>)

Se você estiver interessado em aprender como esses dois funcionam juntos, recomendo este [vídeo](https://www.youtube.com/watch?v=hrBFdVYtHI0) (<https://www.youtube.com/watch?v=hrBFdVYtHI0>), que analisa isso muito bem. Se você ainda está um pouco verde para o MapReduce do Hadoop, não procure mais além deste [vídeo](https://www.youtube.com/watch?v=MfF750YVDxM) (<https://www.youtube.com/watch?v=MfF750YVDxM>), que é a melhor explicação que encontrei até agora. Os vídeos são em inglês mas dá para acompanhar!

Usando Databricks para começar

Ativar um cluster do Spark é um tópico complexo - é um processo não trivial que varia por provedor de nuvem e não é necessariamente o lugar certo para começar para aqueles que estão apenas aprendendo o Spark (também não tenho desejo de abordar isso) É aqui que o Databricks entra.

A [Databricks \(https://databricks.com/\)](https://databricks.com/) é uma empresa privada co-fundada do criador original do Apache Spark. O Databricks fornece uma interface de notebook limpa (semelhante ao Jupyter) que é pré-configurada para se conectar a um cluster Spark. Assim, podemos evitar a configuração inicial associada à criação de um cluster. O nível de [comunidade gratuita \(https://databricks.com/try-databricks?utm_campaign=701610000008nLSAAY&gclid=Cj0KCQjw2IrmBRCJARIsAJZDdxB4j7tL3B\)](https://databricks.com/try-databricks?utm_campaign=701610000008nLSAAY&gclid=Cj0KCQjw2IrmBRCJARIsAJZDdxB4j7tL3B) do Databricks é talvez a melhor maneira para qualquer pessoa começar a aprender o Spark.

Cmd 1

Overview

This notebook will show you how to create and query a table or DataFrame that you uploaded to DBFS. [DBFS](#) is a Databricks File System that allows you to store data for querying inside of Databricks. This notebook assumes that you have a file already inside of DBFS that you would like to read from.

This notebook is written in **Python** so the default cell type is Python. However, you can use different languages by using the `%LANGUAGE` syntax. Python, Scala, SQL, and R are all supported.

Cmd 2

```
1 # File location and type
2 file_location = "/FileStore/tables/fake.csv"
3 file_type = "csv"
4
5 # CSV options
6 infer_schema = "true"
7 first_row_is_header = "true"
8 delimiter = ","
9
10 # The applied options are for CSV files. For other file types, these will be ignored.
11 df = spark.read.format(file_type) \
12     .option("inferSchema", infer_schema) \
13     .option("header", first_row_is_header) \
14     .option("sep", delimiter) \
15     .load(file_location)
16
17 display(df)
```

▶ (3) Spark Jobs

▼ df: pyspark.sql.dataframe.DataFrame

```
id: integer
initiated: timestamp
hiredate: string
```

Navegando na interface do Databricks

Se quiséssemos, poderíamos muito bem permitir que o Databricks possuísse toda a nossa pilha Spark, exceto talvez onde guardamos nossos dados finais. Confira as várias guias que compõem a interface do Databricks:

Dados (Data): aqui é onde controlamos a entrada de dados. O Databricks é fortemente integrado com AWS e Azure.

Clusters: Opções para dimensionar nosso cluster de servidores. Todas as alterações feitas aqui são refletidas diretamente na AWS.

Jobs: O legal dos notebooks do Databricks é que eles não precisam ser apenas notebooks. Ao vincular nossos notebooks a tarefas agendadas, podemos usar nossos notebooks como scripts reais para execução regular. Assim, nossos notebooks se tornam um código de produção totalmente funcional como resultado.


Importação de Bibliotecas ([Import Library](https://community.cloud.databricks.com/?import-library) (<https://community.cloud.databricks.com/?import-library>))


[o=2841344480934877#libraries/create/3752117266494682](#)): Esta é um pouco difícil de encontrar ... ela está em Databricks> Biblioteca de importação. Esta é a ferramenta que usamos para importar bibliotecas Python (ou Java) de terceiros para uso em nossos notebooks. A maneira mais fácil de fazer isso é selecionar “PyPI” e fornecer o nome da biblioteca (ou seja: pandas). Isso instalará a biblioteca em seu cluster e a disponibilizará para uso.


Carregando dados


A maneira mais fácil de adicionar alguns dados para trabalhar no Databricks é por meio de sua IU. Voltando à guia de dados, temos as opções de conectar-se a um bucket do S3, fazer upload de um CSV ou até mesmo selecionar fontes como Amazon Redshift ou Apache Kafka. Infelizmente, o Google Cloud ainda não é compatível com Databricks 😞


Manteremos as coisas simples e enviaremos um CSV para começar:



databricks



Home



Workspace


Recents


Data


Clusters


Jobs


Search

Create New Table (Python)

Data source ?

Upload File

S3

DBFS

Other Data Sources


Upload to DBFS ?

/FileStore/tables/ (optional)

Select

File ?

posts.csv




5.8 MB

[Remove file](#)

✓ File uploaded to /FileStore/tables/posts.csv

Create Table with UI

 Create Table in Notebook



Após o upload, obtemos imediatamente a opção de **Criar Tabela no Notebook (Create Table in Notebook)** . É melhor seguir em frente com essa opção, pois ela nos mostra como carregar dados programaticamente no Spark para o futuro. Selecionar esta opção criará um novo bloco de notas com alguns clichês para nós.

Tipos de dados estruturados no Spark

Antes de carregar nossos dados, devemos estar cientes dos diferentes tipos de dados com os quais podemos trabalhar no Spark. O Spark começou com uma única estrutura de dados chamada **RDDs** e se expandiu para dados bidimensionais ao adicionar **DataFrames** e **Datasets** (Conjuntos de dados):

- Um **RDD** é um conjunto de dados preenchido em nossos muitos clusters Spark, daí seu nome (Resilient Distributed Datasets). Os dados RDD são armazenados como uma lista ou um armazenamento de chave / valor. Os **RDDs** servem a muitos propósitos, como analisar arquivos de texto em tipos de dados viáveis que podem ser reaproveitados em DataFrames.
- Os **DataFrames** são uma evolução dos **RDDs** projetados para serem mais fáceis de usar, semelhante a como podemos esperar que as tabelas SQL funcionem. Como resultado, DataFrames criados apenas a partir de dados estruturados. Ao contrário dos **RDDs**, os **DataFrames** têm automaticamente uma série de otimizações aplicadas a eles que tornam o trabalho com dados estruturados mais rápido e fácil. Ao contrário dos **RDDs**, no entanto, os **DataFrames** não podem ser modificados após sua criação inicial.
- **Datasets** são uma forma de dados tabulares limitados a Scala (booo). Os conjuntos de dados são semelhantes aos DataFrames, mas adicionam uma camada adicional de segurança de tipo para impedir que dados do tipo errado caiam nas colunas erradas.

Os **DataFrames** e os **Dataset** juntos formam a parte do Spark frequentemente referida como "Spark SQL", que não deve ser confundida com as funções SQL que podemos executar no Spark. Na verdade, existem muitas convenções de nomenclatura que deixam muito a desejar no Spark.

Anatomia de um Spark DataFrame

Os RDDs podem ter sido a maneira original de trabalhar com dados no Spark, mas atualmente os DataFrames são o pão com manteiga. Isso não é para subestimar a importância dos RDDs - DataFrames são construídos na mesma lógica dos RDDs, e nós absolutamente devemos conhecer os meandros dos RDDs se quisermos nos considerar usuários respeitáveis do Spark. Dito isso, se você deseja aprender o máximo possível sobre o Spark em um curto período de tempo, devemos nos concentrar nos DataFrames.

Aqui estão três propriedades de DataFrames das quais devemos estar cientes:

- **Imutável:** Spark DataFrames gosta de ser criado uma vez com antecedência, sem ser modificado após o fato.
- **Distribuído:** Spark DataFrames são tolerantes a falhas e altamente disponíveis, bem como vanilla Hadoop. Portanto, corremos pouco risco de algo dar terrivelmente errado e limpar nosso DataFrame devido a fatores externos - se um nó em nosso cluster Spark cair, o Spark pode avançar independentemente.
- **Avaliação lenta (Lazy evaluation):** as tarefas DataFrame não são executadas até que sejam explicitamente executadas. Este não deve ser um novo conceito; aqueles que estão familiarizados com o guia de estilo PEP sabem que a maioria das tarefas relacionadas a dados em Python são executadas dessa maneira. Ótimo, mas como é trabalhar com um Spark DataFrame? A estrutura geral de modificação de um Spark DataFrame normalmente se parece com esta:

```
new_df = original_df. \
    transformation_1(original_df). \
    transformation_2(original_df). \
    transformation_3(original_df)
```

Como mencionamos antes, Spark DataFrames são imutáveis, então precisamos criar um novo DataFrame do nosso original toda vez que quisermos fazer ajustes (AKA: `new_df = original_df.etc().etc()`).

A partir daí, podemos encadear transformações para garantir que não criaremos um novo DataFrame por transformação, o que seria um desperdício de memória.

Criando Spark DataFrames

Podemos criar Spark DataFrames a partir de várias fontes diferentes, como CSVs, arquivos JSON ou até mesmo unindo RDDs. Já carregamos um CSV, então vamos começar por aí.

Criação de DataFrames a partir de CSVs

Se você selecionou a opção **Criar Tabela no Notebook (Create Table in Notebook)** ao fazer o upload do seu CSV, deverá ver um bloco de notas com uma primeira célula semelhante a esta:

```
# File location and type
file_location = "/FileStore/tables/political_social_media-1.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema) \
    .option("header", first_row_is_header) \
    .option("sep", delimiter) \
    .load(file_location)

display(df)
```

Esta célula está nos mostrando o método preferido para criar um DataFrame a partir de um arquivo CSV. A primeira coisa que acontece é `spark.read.format("csv")` : isso informa ao Spark que estamos prestes a ler os dados (sim, podemos escrever dessa forma também ... aguarde).

Dizemos ao Spark que o arquivo que passaremos será um CSV usando `.format()`. Também podemos especificar txt, JSON, etc...

Observe como temos algumas `option()` linkadas a esta instrução `read` ? Cada "option" especifica a configuração que determina como nosso CSV deve ser criado:

- **`infer_schema`** tentará observar os padrões nos dados que carregamos e reduzir automaticamente cada coluna para o tipo de dados adequado. Isso provou ser surpreendentemente preciso; vamos visualizar essa funcionalidade definindo o valor como "true".
- **`first_row_is_header`** respeitará a primeira linha de nosso CSV como os nomes de cabeçalho de nosso DataFrame quando a opção for definida como "true".
- **`delimiter(delimitador)`** nos permite separar colunas por um caractere diferente de uma vírgula. Podemos deixar como ";", a menos que seus "Valores separados por vírgula" sejam separados por algo diferente de vírgulas ...

Você já pode estar confuso com algumas das coisas estranhas de sintaxe que acontecem aqui:

- Os valores booleanos no PySpark às vezes são definidos por strings ("true" ou "false", em oposição a True ou False). Na verdade, também há momentos em que esse não é o caso (os argumentos de palavra-chave no PySpark normalmente aceitam True e False). Esta é uma das muitas escolhas desagradáveis feitas no design do PySpark.
- Você reconhecerá "df" como uma abreviação para DataFrame, mas não vamos nos deixar levar - trabalhar com PySpark DataFrames é muito diferente de trabalhar com Pandas DataFrames na prática, embora sejam conceitualmente semelhantes. A sintaxe e a tecnologia subjacente são bastante diferentes entre as duas.
- O Databricks nos fornece uma função `display()` super organizada, que é uma forma muito mais poderosa de visualizar nossos DataFrames do que `print()`. Com certeza veremos algumas das vantagens de `display()` conforme prosseguirmos.

Nossa primeira célula agora está pronta para funcionar. Vamos ver o que isso faz:

databricks

Home

Workspace

Recents

Data

Clusters

Jobs

Search

2019-04-26 - DBFS Example (1) (Python)

Attached: My Cluster File View: Code Permissions Run All Clear

Run Publish Comments Revision history

Cell 1

Overview

This notebook will show you how to create and query a table or DataFrame that you uploaded to DBFS. **DBFS** is a Databricks File System that allows you to store data for querying inside of Databricks. This notebook assumes that you have a file already inside of DBFS that you would like to read from.

This notebook is written in **Python** so the default cell type is Python. However, you can use different languages by using the `%LANGUAGE` syntax. Python, Scala, SQL, and R are all supported.

Cell 2

```
1 # File location and type
2 file_location = "/FileStore/tables/sample_data.csv"
3 file_type = "csv"
4
5 # CSV options
6 infer_schema = "true"
7 first_row_is_header = "true"
8 delimiter = ","
9
10 # The applied options are for CSV files. For other file types, these will be ignored.
11 df = spark.read.format(file_type) \
12     .option("inferSchema", infer_schema) \
13     .option("header", first_row_is_header) \
14     .option("sep", delimiter) \
15     .load(file_location)
16
17 display(df)
```

▶ (2) Spark Jobs
▶ df: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 16 more fields]

_c0	_c1	_c2	_c3	_c4	_c5	_c6	_c7
id	key	assignee_name	assignee_url	summary	status	priority_name	priority_url
5936	HACK-938	Todd Birchard	https://avatar-cdn.atlassian.com/9eb3868db428fb602e03b3059608199b?s=48&d=https%3A%2F%2Fsecure.gravatar.com%2Favatar%2F9eb3868db428fb602e03b3059608199b%3Fd%3Dmm%26s%3D48%26noRedirect%3Dtrue	Update Jira ETL script to work with mongo	In Progress	null	https://hackersandslackers.atlassian.net/images/icon
5937	HACK-937	Todd Birchard	https://avatar-cdn.atlassian.com/9eb3868db428fb602e03b3059608199b?s=48&d=https%3A%2F%2Fsecure.gravatar.com%2Favatar%2F9eb3868db428fb602e03b3059608199b%3Fd%3Dmm%26s%3D48%26noRedirect%3Dtrue	Remake footer logo	Done	null	https://hackersandslackers.atlassian.net/images/icon
5938	HACK-145	Todd Birchard	https://avatar-cdn.atlassian.com/9eb3868db428fb602e03b3059608199b?s=48&d=https%3A%2F%2Fsecure.gravatar.com%2Favatar%2F9eb3868db428fb602e03b3059608199b%3Fd%3Dmm%26s%3D48%26noRedirect%3Dtrue	Creative	To Do	null	https://hackersandslackers.atlassian.net/images/icon
5939	DEV-119	Todd Birchard	https://avatar-cdn.atlassian.com/9eb3868db428fb602e03b3059608199b?s=48&d=https%3A%2F%2Fsecure.gravatar.com%2Favatar%2F9eb3868db428fb602e03b3059608199b%3Fd%3Dmm%26s%3D48%26noRedirect%3Dtrue	Move hackersandslackers tutorials to GitHub orb	In Progress	null	https://hackersandslackers.atlassian.net/images/icon
5940	DEV-25	Todd Birchard	https://avatar-cdn.atlassian.com/9eb3868db428fb602e03b3059608199b?s=48&d=https%3A%2F%2Fsecure.gravatar.com%2Favatar%2F9eb3868db428fb602e03b3059608199b%3Fd%3Dmm%26s%3D48%26noRedirect%3Dtrue	Github	To Do	null	https://hackersandslackers.atlassian.net/images/icon

Command took 5.58 seconds -- by i.am.todd.birchard@gmail.com at 4/26/2019, 2:27:36 PM on My Cluster

```
In [0]: df1 = spark.read.format("csv").load("dbfs:/FileStore/shared_uploads/d.napolitano@uni9.pr
o.br/all_seasons-1.csv", header="true", inferSchema="true")
display(df1)
```

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_ro
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
1	Dwayne Schintzius	LAC	28.0	215.9	117.93392	Florida	USA	1990	1
2	Earl Cureton	TOR	39.0	205.74	95.25432	Detroit Mercy	USA	1979	3
3	Ed O'Bannon	DAL	24.0	203.2	100.697424	UCLA	USA	1995	1
4	Ed Pinckney	MIA	34.0	205.74	108.86208	Villanova	USA	1985	1
5	Eddie Johnson	HOU	38.0	200.66	97.522280000000001	Illinois	USA	1981	2
6	Eddie Jones	LAL	25.0	198.12	86.18248	Temple	USA	1994	1
7	Elden								

In [0]: display(df1)

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_ro
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
1	Dwayne Schintzius	LAC	28.0	215.9	117.93392	Florida	USA	1990	1
2	Earl Cureton	TOR	39.0	205.74	95.25432	Detroit Mercy	USA	1979	3
3	Ed O'Bannon	DAL	24.0	203.2	100.697424	UCLA	USA	1995	1
4	Ed Pinckney	MIA	34.0	205.74	108.86208	Villanova	USA	1985	1
5	Eddie Johnson	HOU	38.0	200.66	97.52228000000001	Illinois	USA	1981	2
6	Eddie Jones	LAL	25.0	198.12	86.18248	Temple	USA	1994	1
7	Elden		33.0	210.04	110.000	St.	USA	1980	1

In [0]: display(df1)

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_ro
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
1	Dwayne Schintzius	LAC	28.0	215.9	117.93392	Florida	USA	1990	1
2	Earl Cureton	TOR	39.0	205.74	95.25432	Detroit Mercy	USA	1979	3
3	Ed O'Bannon	DAL	24.0	203.2	100.697424	UCLA	USA	1995	1
4	Ed Pinckney	MIA	34.0	205.74	108.86208	Villanova	USA	1985	1
5	Eddie Johnson	HOU	38.0	200.66	97.52228000000001	Illinois	USA	1981	2
6	Eddie Jones	LAL	25.0	198.12	86.18248	Temple	USA	1994	1
7	Elden		33.0	210.04	110.000	St.	USA	1980	1

In [0]: display(df1)

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_ro
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
1	Dwayne Schintzius	LAC	28.0	215.9	117.93392	Florida	USA	1990	1
2	Earl Cureton	TOR	39.0	205.74	95.25432	Detroit Mercy	USA	1979	3
3	Ed O'Bannon	DAL	24.0	203.2	100.697424	UCLA	USA	1995	1
4	Ed Pinckney	MIA	34.0	205.74	108.86208	Villanova	USA	1985	1
5	Eddie Johnson	HOU	38.0	200.66	97.52228000000001	Illinois	USA	1981	2
6	Eddie Jones	LAL	25.0	198.12	86.18248	Temple	USA	1994	1
7	Elden		33.0	210.04	110.000	St.	USA	1980	1

In [0]: display(df1)

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_ro
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
1	Dwayne Schintzius	LAC	28.0	215.9	117.93392	Florida	USA	1990	1
2	Earl Cureton	TOR	39.0	205.74	95.25432	Detroit Mercy	USA	1979	3
3	Ed O'Bannon	DAL	24.0	203.2	100.697424	UCLA	USA	1995	1
4	Ed Pinckney	MIA	34.0	205.74	108.86208	Villanova	USA	1985	1
5	Eddie Johnson	HOU	38.0	200.66	97.52228000000001	Illinois	USA	1981	2
6	Eddie Jones	LAL	25.0	198.12	86.18248	Temple	USA	1994	1
7	Elden		33.0	210.04	110.000	St.	USA	1980	1


```
In [0]: filter_df = df1.filter(df1.team_abbreviation == "CHI")
#filter_df = df1.filter(df1.age >= 30)
display(filter_df)
```

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	draft_round
0	Dennis Rodman	CHI	36.0	198.12	99.79024	Southeastern Oklahoma State	USA	1986	2
40	Dickey Simpkins	CHI	25.0	205.74	112.490816	Providence	USA	1994	1
71	Jason Caffey	CHI	24.0	203.2	116.119552	Alabama	USA	1995	1
117	Bill Wennington	CHI	34.0	213.36	111.13004	St. John's (NY)	USA	1985	1
120	Bison Dele	CHI	28.0	210.82	117.93392	Arizona	USA	1991	1
234	Scottie Pippen	CHI	31.0	200.66	103.418976	Central Arkansas	USA	1987	1
249	Ron Harper	CHI	33.0	198.12	97.975872	Miami (OH)	USA	1986	1
251	Randy	CHI	33.0	197.04	97.40016	New Mexico	USA	1987	1

```
In [0]: sorted_df=df1.sort(df1.pts)
display(sorted_df)
```

_c0	player_name	team_abbreviation	age	player_height	player_weight	college	country	draft_year	dra
10910	Stanton Kidd	UTA	27.0	198.12	97.975872	Colorado State	USA	Undrafted	Unc
5758	Martell Webster	POR	22.0	200.66	106.59411999999999	None	USA	2005	1
10936	Quinndary Weatherspoon	SAS	23.0	190.5	92.98635999999999	Mississippi State	USA	2019	2
10952	Talen Horton-Tucker	LAL	19.0	193.04	106.140528	Iowa State	USA	2019	2
2305	Randy Brown	BOS	34.0	187.96	86.18248	New Mexico State	USA	1991	2
5869	JamesOn Curry	LAC	24.0	190.5	86.18248	Oklahoma State	USA	2007	2
10958	Vic Law	ORL	24.0	200.66	91.171992	Northwestern	USA	Undrafted	Unc
1205	Gary Grant	POR	34.0	190.5	83.91452	Michigan	USA	1988	1