Soluções para processamento paralelo e distribuído de dados

Spark – Algumas ações e transformações

Ilustração de Oliver Munday

Spark – Algumas Ações e Transformações

Passando funções:

- Muitas *transformations* e *actions* vão exigir como parâmetros funções que irão ser aplicadas aos dados ;
- Cada uma das linguagens utilizadas no Spark possui formas de passar tais funções para os *ransformations* e *actions;*
- Vamos focar em Python e Scala;

Spark – Algumas Ações e Transformações

Passando funções:

Python

- Para funções curtas podemos utilizar o *lambda*

```
word = rdd.filter(lambda s: "error" in s)

def containsError(s):
    return "error" in s
word = rdd.filter(containsError)
```

Spark – Algumas Ações e Transformações

Passando funções:

Python

- Para funções mais longas use a função

```
def myFunc(s):
    palavras = s.split(" ")
    return len(palavras)

sc = SparkContext(...)
sc.textFile('arquivo.txt').map(MyFunc)
```

- Tome cuidado para não passar métodos de objetos, em alguns caso o objeto será serializado e enviado para o cluster;

Spark – Algumas Ações e Transformações

Passando funções:

Scala

- Para funções mais curtas use comando *inline* :

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

- Para funções longas:

```
object MyFunctions {
  def func1(s: String): String = { ... }
}

myRdd.map(MyFunctions.func1)
```

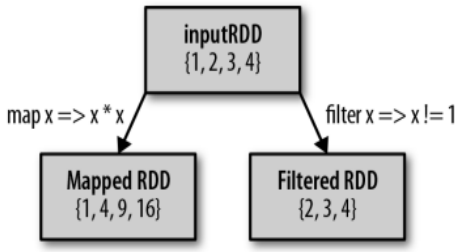
Spark – Algumas Ações e Transformações

Transformações - Básico:

- Duas transformações muito comuns são:

  - *Map()* : aplica uma função a cada elemento do RDD, e os elementos gerados pela função formam o novo RDD

  - *Filter():* aplica uma função e retorno o conjunto de elementos filtrados com um novo RDD



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Duas transformações muito comuns são:
  - *Map()*
    - Pode ser utilizada para um grande variedade de tarefas: buscar o conteúdo de uma URL ou retornar um número elevado ao cubo;
    - O RDD de entrada não precisa ser do mesmo tipo do RDD de saída:

```
txt_number = sc.parallelize([u'1',u'2',u'3',u'4',u'1',u'2'])
int_number = txt_number.map(lambda x:int(x))
sum(int_number.collect())
```

Spark – Algumas Ações e Transformações

Transformações - Básico:

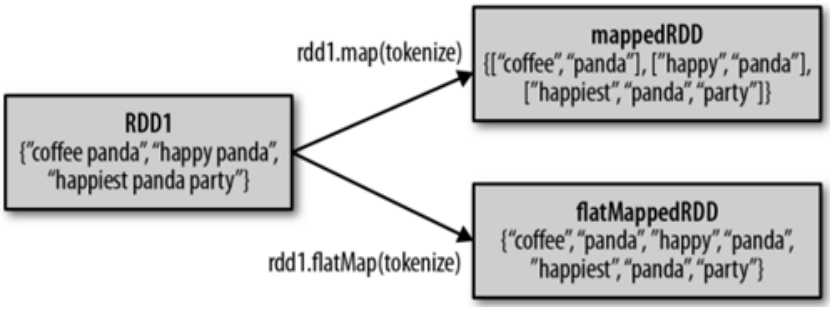
- Outras transformações:
  - *flatMap()*
    - Similar ao *Map* com a diferença que para cada item de entrada pode ser gerada um mais itens de saída;
    - Retorna um *iterator* para o resultado;
    - Exemplo clássico de *flatMap* é quebrar um linha em várias palabras

```
linhas = sc.parallelize(["hello world", "hi"])
palavra = linhas.flatMap(lambda linha: linha.split(" "))
palavra.take(3) # retorno ['hello', 'world', 'hi']
```

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Outras transformações:
  - Diferença entre *Map()* e *flatMap()*



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:
  - Os RDD´s não são exatamente conjuntos(sets ou tabelas), mas há um bom conjunto de transformações para lidar com conjuntos;
  - *Distinct*() = para gerar elementos distintos de um RDD;
    - Tome cuidado com esta operação pois ela é, computacionalmente, custosa;
  - *Union()* = faz união dos dados dos conjuntos. As repetições são mantidas;

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

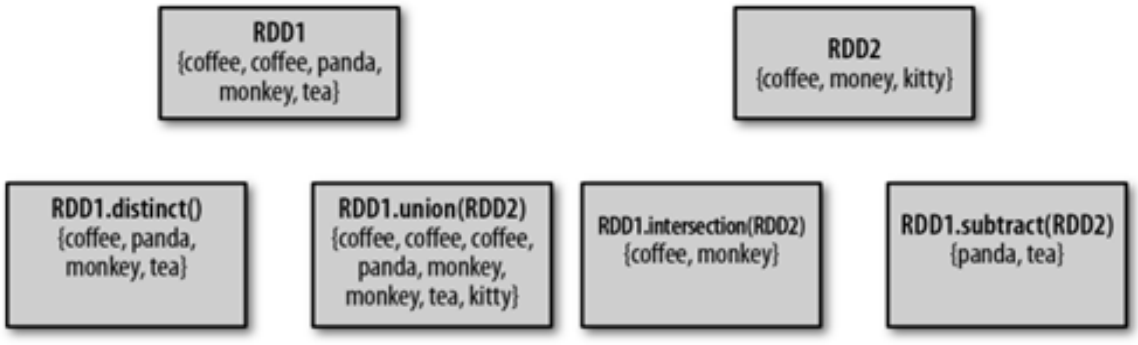
  - *intersection*() = retorna os elementos existente em ambos os RDD´s (elimina os duplicados = atua como o *distinct*(), mantendo sua obsevação quanto ao desempenho);

  - subtract*()* = retorna os valores existentes apenas no primeiro RDD e que não existem no segundo RDD (tome cuidado com o desempenho);

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

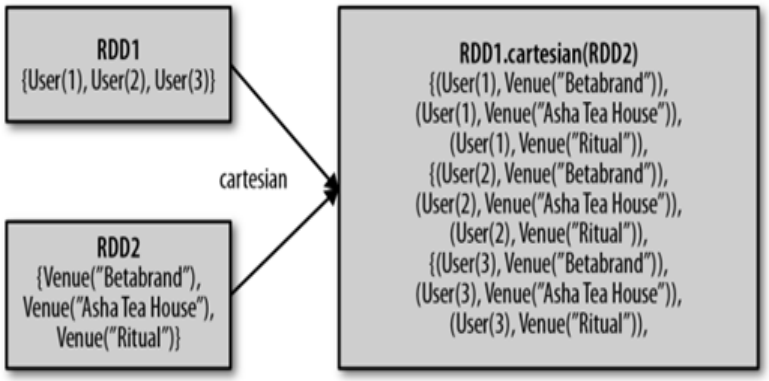


Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

  - *cartesian*() = realiza um produto cartesiano entre os dois RDD´s envolvidos. Veja o exemplo:



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

  - *parallelize()*

    - É um trasnformation

    - Não é muito utilizado na prática, pois os seus dados tem caber na memória de uma única máquina

Spark – Algumas Ações e Transformações

Ações - Básico:

- *reduce*()
  - Talvez seja a ação mais comum;
  - Aplica uma função em dois elementos (tipo de dados) do RDD gerando um resultado do mesmo tipo;
  - O exemplo mais comum de função é “+”(somar o RDD);
  - Em geral o reduce é utilizado para agregar elementos;

```
sum = rdd.reduce(lambda x, y: x + y)
```

- *fold*()
  - Faz a mesma coisa que o Reduce, mas atribui um valor inicial para cada elemento na soma;

```
sc.parallelize([1, 2, 3, 4, 5]).fold(0, add)
```

Spark – Algumas Ações e Transformações

Ações - Básico:

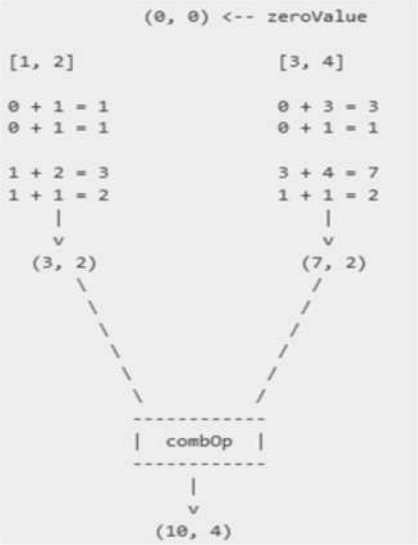
- *Aggregate*()
  - Tanto fold quanto reduce exigem que o retorno seja igual ao tipo do RDD trabalhado;
  - Para evitar este problema o que poderia ser feito é criar uma função *map* e depois uma funcão *reduce*;
  - É exatamente isto que a função *aggregate* faz, para tal devemos:
    - Fornecer valores iniciais (* atenção use sempre os valores = 0);
    - Função que será aplicada em cada elemento de cada partição para gerar o resultado intermediário de cada partição
    - Função para fazer *merge* dos resultados de parciais de cada partição

Spark – Algumas Ações e Transformações

Ações - Básico:

- *Aggregate*()

```
nums = sc.parallelize([1,2,3,4], 2)
seqOp = (lambda x, y: (x[0] + y, x[1] + 1))
combOp = (lambda x, y: (x[0] + y[0], x[1] + y[1]))
nums.aggregate((0, 0), seqOp, combOp)
```



Spark – Algumas Ações e Transformações

Ações - Básico:

- *collect*()
  - Forma mais simples de obter todo conteúdo de um RDD. Obs.: cuidado no uso para grandes bases de dados ;
- *take*(n)
  - Retorna os *n* elementos do RDD;
- *takeSample (withReplacement, num, seed)*
  - Faz uma amostragem com ou sem reposição (assume uma distruibuição uniforme dos dados);

Spark – Algumas Ações e Transformações

Ações - Básico:

- *foreach*()
  - Executa uma função em cada elemento do RDD, mas não retorna nada para o Driver ;
  - Pense no caso, por exemplo, de ler um arquivo Json e armazená-lo em um outro banco de dados;
- *count*()
  - Retorna a quantidade de elementos do RDD;
- countByValue *()*
  - Retorna um mapa com a quantidade de elementos, por valor, do RDD;

Spark – Algumas Ações e Transformações

Ações - Básico:

- Exemplos

Entrada	Função	Saída
{1, 2, 3, 3}	rdd.collect()	{1, 2, 3, 3}
{1, 2, 3, 3}	rdd.count()	4
{1, 2, 3, 3}	rdd.countByValue()	{(1, 1),(2, 1),(3, 2)}
{1, 2, 3, 3}	rdd.top(2)	{3, 3}
{1, 2, 3, 3}	rdd.takeSample(false, 1)	1 – estocástico
{1, 2, 3, 3}	rdd.foreach(func)	Não retorna nada

Spark – Algumas Ações e Transformações

Prática

- Trabalhando com algumas funções do Spark

Soluções para processamento paralelo e distribuído de dados

Spark – SQL

Ilustração de Oliver Munday

Spark – SQL

Introdução

- É a proposta Spark para lidar com dados estruturados e semi-estruturados (dados com algum tipo de esquema);
- Características
  - Carga de vários tipos de dados(Json, Hive, Parquet);
  - Consultas são realizadas utilizando SQL (através de um programa spark ou através conexões externas: JDBC/ODBC);
  - Forte integração para ser utilizada com as linguagens spark: Java, Scala e Python. Permitir mesclar um RDD com dados do SQL Spark

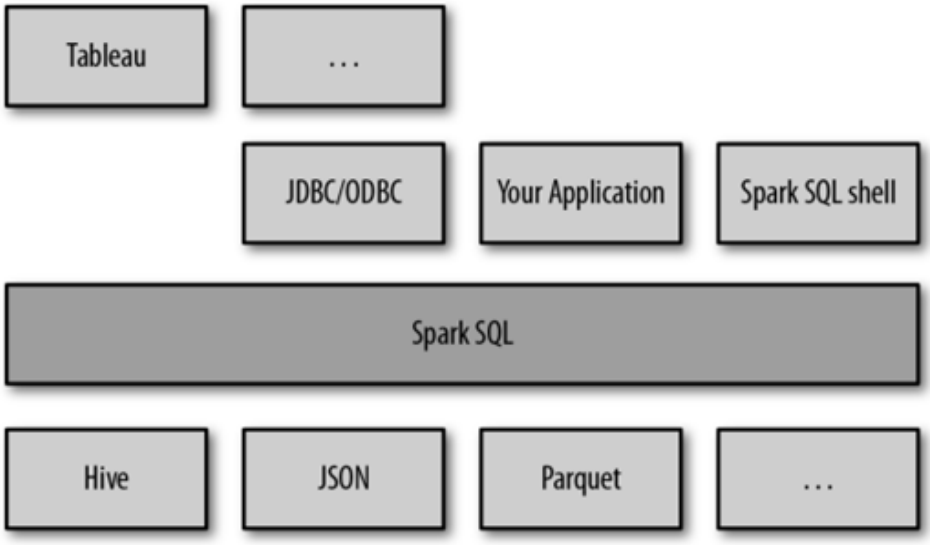
Spark – SQL

Introdução

- Os dados carregados com SQL ainda são RDD´s;
- No entanto, do tipo SchemaRDD;
- SchemaRDD:
  - Conjunto de objetos do tipo *Row*;
  - SchemaRDD conhece seu esquema (campos);
  - Apesar de ser um RDD, possui otimizações internas para lidar com este tipo de dado;
  - Possui novas operações que não existem nos outros RDD´s
  - Podem ser carregados a partir de várias fontes externas

Spark – SQL

Introdução



Spark – SQL

Introdução

-Pode possuir forte integração com o Hive (recomenda-se o seu uso com esta integração habilitada);

Spark – SQL

Exemplo inicial

-O exemplo abaixo mostra como é uma conexão utilizando o pyspark:

```
from pyspark.sql import HiveContext, Row # ou
from pyspark.sql import SQLContext, Row

hiveCtx = HiveContext(sc)
arqinput = hiveCtx.jsonFile('/user/hue/pessoasval.json')
arqinput.registerTempTable("pessoas")

allPessoas = hiveCtx.sql("""SELECT * FROM pessoas ORDER BY 1 LIMIT 10""")
allPessoas.collect()
```

Spark – SQL

SQL

-Para fazer consultas SQL utilize o método Sql() do HiveContext ou SQLContext;

- Antes da consulta os dados precisam ser 'conhecidos' pelo SparkSQL, no caso anterior, usamos um arquivo json;

- Uma tabela temporária é registrada com um metadado com nome;

- Consultas SQL podem ser feitas nesta estrutura;

Spark – SQL

SchemaRDD – versão Spark 1.x

- São similares a tabelas em bancos de dados relacionais;

- Os resultados das operações de carga dos dados ou SQL retornam SchemaRDD´s;

- Internamente os SchemaRDD´s são um conjunto de objetos do tipo Row;

- Cada objeto do tipo Row é uma especie de vetor com campos definidos pelo Schema;

- Como o SchemaRDD é ainda um RDD, todos os métodos utilizados para o RDD podem ser usados no SchemaRDD

Spark – SQL

SchemaRDD – versão Spark 1.x

- As tabelas temporárias registradas pelo método registerTempTable são alocadas temporariamente e assim que o Driver program termina elas são liberadas;

- Em python cada uma linhas pode ser acessada:

```
row[i]
row.datafieldname # veja que datafieldname é o nome real da coluna

#Exemplo
topTweetText = topTweets.map(lambda row: row.text)
```

Spark – SQL

SchemaRDD – versão Spark 1.x

- Tipos de dados:

Spark SQL/HiveQL type	Scala type	Java type	Python
TINYINT	Byte	Byte/byte	int/long (in range of –128 to 127)
SMALLINT	Short	Short/short	int/long (in range of –32768 to 32767)
INT	Int	Int/int	int or long
BIGINT	Long	Long/long	long
FLOAT	Float	Float/float	float
DOUBLE	Double	Double/double	float
DECIMAL	Scala.math.BigDecimal	Java.math.BigDecimal	decimal.Decimal

Spark – SQL

SchemaRDD – versão Spark 1.x

- Tipos de dados:

Spark SQL/HiveQL type	Scala type	Java type	Python
STRING	String	String	string
BINARY	Array[Byte]	byte[]	bytearray
BOOLEAN	Boolean	Boolean/boolean	bool
TIMESTAMP	java.sql.TimeStamp	java.sql.TimeStamp	datetime.datetime
ARRAY<DATA_TYPE>	Seq	List	list, tuple, or array
MAP<KEY_TYPE, VAL_TYPE>	Map	Map	dict
STRUCT<COL1: COL1_TYPE, ...>	Row	Row	Row

Spark – SQL

SchemaRDD – versão Spark 1.x

- A partir do Spark SQL 1.3 o schemaRDD foi renomeado para DataFrame (nesta versão não herda mais diretamente do RDD, mas possui os métodos que o mesmo tem)

Spark – SQL

Caching

- Como o SchemaRDD conhece os tipos de dados, o processo de carregar os dados para a memória é mais otimizada;

- Utiliza uma forma colunar de armazenamento dos dados;

- O método *cacheTable*(“nometabela”) garante que o método otimizado é utilizado ao invés do objeto completo;

- Os dados ficam na memória durante toda a execução do *Driver program;*

- Da mesma forma que os RDD, faz sentido manter em memória se for utilizado várias vezes;

Spark – SQL

Caching

- As tabelas também pode ser armazenadas na memória por comandos HIVE QL :

```
CACHE TABLE tablename
UNCACHE TABLE tablename
```

-Este comando é mais utilizado com comando enviados por clientes, através de um JDBC *Server*, por exemplo ;

- As tabelas (*SchemaRDD* ou *DataFrames*) são apresentados com um RDD típico no Spark UI

Spark – SQL

Carregando e Salvando Dados

- As fontes para carga de dados são: tabelas *Hive* , arquivos *Json* e arquivos Parquet;

- Os RDD´s existentes também pode ser transformados em SchemaRDD´s

- A recomendação de uso do SparkSQL é quando existem vários cálculos a serem feitos na mesma estrutura (média, desvio padrão, soma, quantidade e etc....). Melhor Desempenho;

- O processo de junção do SQL também simplifica algumas tarefas;

Spark – SQL

Carregando e Salvando Dados

Hive

- Quando lê usando o Hive consegue utilizar todos os formatos de arquivos suportados pelo Hive (arquivos textos, Parquet, ORC);

- Para conectar o Spark SQL com o Hive é necessário prover uma configuração. Em essência: copiar o arquivo *hive-site.xml* para o *./conf* do Spark;

-Exemplo Python:

```
from pyspark.sql import HiveContext

hiveCtx = HiveContext(sc)
rows = hiveCtx.sql("SELECT key, value FROM mytable")
keys = rows.map(lambda row: row[0])
```

Spark – SQL

Carregando e Salvando Dados

Parquet

- É um estrutura de arquivos orientado a coluna e bastante utilizada no Hadoop;

- Suporta todos os tipos do Spark SQL

-Exemplo Python:

```
rows = hiveCtx.parquetFile(parquetFile)
tbl = rows.registerTempTable("people")
pandaFriends = hiveCtx.sql("SELECT name FROM people WHERE favouriteAnimal = \"panda\"")
pandaFriends.saveAsParquetFile("hdfs://...")
```

Spark – SQL

Carregando e Salvando Dados

Json

- É necessário que os arquivos tenham o mesmo esquema (caso de uso: diretórios com milhões de arquivos json com o mesmo esquema);

- Spark SQL infere o esquema e permite o acesso aos nomes dos campos ;

-Exemplo Python:

```
{"name":"Sparky The Bear", "lovesPandas":true, "knows":{"friends": ["holden"]}}
```

printSchema()

```
root
 |-- knows: struct (nullable = true)
 |    |-- friends: array (nullable = true)
 |    |    |-- element: string (containsNull = false)
 |-- lovesPandas: boolean (nullable = true)
 |-- name: string (nullable = true)
```

Spark – SQL

Carregando e Salvando Dados

RDD

- A partir de um RDD é possível criar também uma schemaRDD;

-Exemplo Python:

```
happyPeopleRDD = sc.parallelize([Row(name="holden", favouriteBeverage="coffee")])
happyPeopleSchemaRDD = hiveCtx.inferSchema(happyPeopleRDD)
happyPeopleSchemaRDD.registerTempTable("happy_people")
```

Spark – SQL

Acessando dados

JDBC/ODBC

- Provê conexão JDBC para uso, por exemplo, por ferramentas de clássicas de BI ou Data Discovery;

- Servidor JDBC executa como um processo de *background* (é um *Driver program*);

- Permite conexão de vários usuários e compartilha as estruturas de memória entre os mesmos;

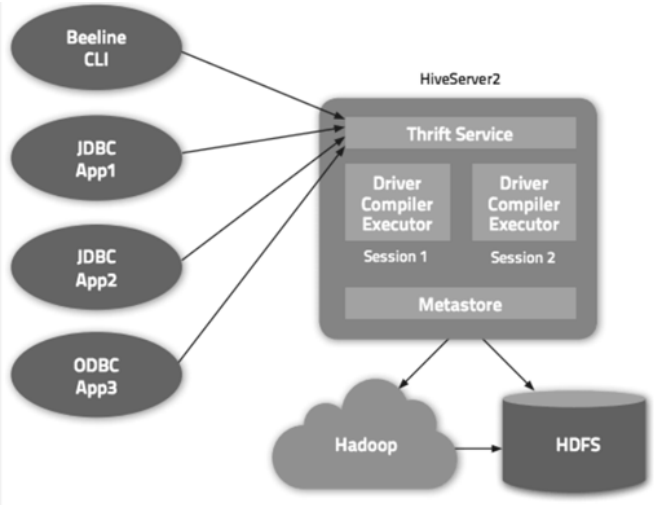
-O servidor Spark SQL JDBC corresponde o HiveServer2 no Hive. Esta conexão JDBC precisa do Hive habilitado para que possa funcionar;

Spark – SQL

Acessando dados

JDBC/ODBC

HiveServer2



Spark – SQL

Acessando dados

JDBC/ODBC

- O servidor pode ser iniciado em *sbin/start-thriftserver.sh*;

```
./sbin/start-thriftserver.sh --master sparkMaster
```

- Também pode ser usado o Beeline um cliente que realiza faz conexões com o servidor JDBC;

- O servidor JDBC gera arquivos de Log que pode ajudar na resolução de algum tipo de problema;

Spark – SQL

Acessando dados

JDBC/ODBC

- Muitos outros clientes também podem fazer conexão via *driver* ODBC;

- O Driver ODBC do Spark SQL é produzido pela Simba®;

- Adicionalmente se a ferramenta possui conexão com o Hive (Hive ODBC HortonWorks) ele poderá se conectar ao Spark SQL

Spark – Algumas Ações e Transformações

Prática

- Utilizando o Spark-sql

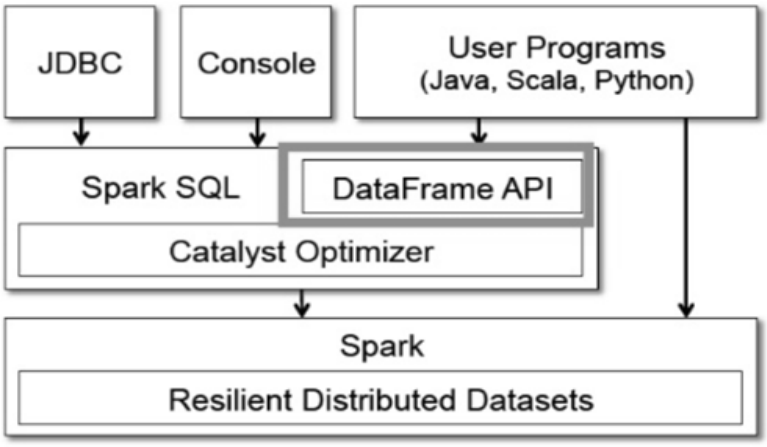
Soluções para processamento paralelo e distribuído de dados

DataFrame - API

Ilustração de Oliver Munday

DataFrame - API

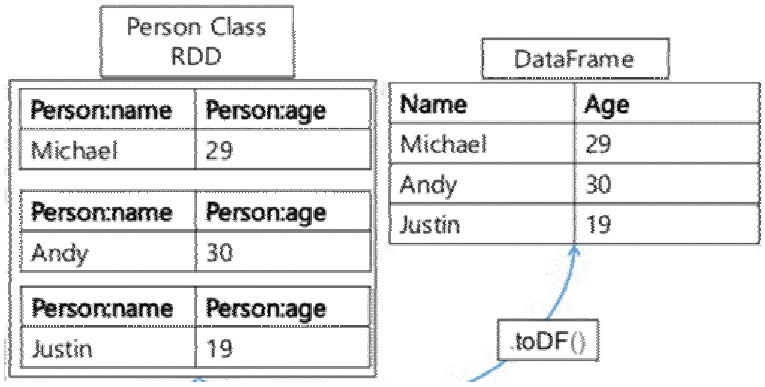
DataFrame API



DataFrame - API

DataFrame API

- Diferenças RDD e DataFrame



Spark – SQL

Alguma operações comuns

- Tipo de dados das colunas

```
DF.printSchema()
```

- Obtenção das primeiras linhas do data frame

Parâmetro com o número de linhas

```
DF.head(5)
```

- Número de linhas

```
DF.count()
```

Número de colunas

```
len(DF.columns)
```

Spark – SQL

Alguma operações comuns

- Estatísticas descritivas para colunas numéricas

```
DF.describe()
```

- Mostrando somente algumas colunas

```
DF.select('Col1','Col2').show(5)
```

- Quantidade de elementos distintos em uma coluna

```
DF.select('Col1').distinct().count()
```

- Geração de tabelas cruzadas(crosstab)

```
DF.crosstab('Col1', 'Col2').show()
```

Spark – SQL

Alguma operações comuns

- Eliminado linhas duplicadas em um data frame

```
DF.select('Col1','Col2').dropDuplicates(
).show()
```

- Excluindo linhas com valores nulos

```
DF.dropna()
```

- Preenchendo valores nulos com outros valores

```
DF.fillna(-1).show(2)
```

- Fitlras linhas

```
DF.filter(DF.Col1 > 15000)
```

Spark – SQL

Alguma operações comuns

- Agrupamentos por colunas

```
DF.groupby('Col1').agg({'Val':
'sum'}).show()
DF.groupby('Col1').count().show()
```

- Amostragem
  - Parâmetros :
    - Reposição = True ou False
    - Fração = 0.2 significa 20% das linhas do data frame
    - seed = para permitir que os resultados sejam reproduzidos

```
DF.sample(False, 0.2, 1234)
```

Spark – SQL

Alguma operações comuns

- Aplicando operações map para uma coluna

```
DF.select('Col1').map(lambda x:(x,1))
```

- Ordenação da tabela base em uma coluna

```
DF.orderBy(DF.Col1.desc()).show(10)
```

- Adicionando colunas calculadas
  - Parâmetros :
    - Nome da coluna = nova ou existente
    - Expressão

```
DF.withColumn('New_Col1',DF.Col1 *
0.2).select('Col1','New_Col1').show(5)
```