



Spark

11/05/2018

Everton Osnei Cesario

Apresentação

- Everton Osnei Cesario
 - Sistemas de Informação / BI / Big Data / Mestrando IA – Saúde
 - Modelos Preditivos / KDD / Data Mining
- Desenvolvedor
 - Qlikview / MicroStrategy /C++/Java
- Professor Pós Positivo
 - Business Intelligence
 - Inteligência Artificial
 - Big Data
- Celepar
 - 8 anos - 2 Pentaho - 6 Qlikview / MicroStrategy
 - 2 Big Data – Analytics / IA
- Administração de ambientes
 - Dimensionamento e disponibilidade.
 - 4 Servidores Qlikview
 - Cluster 4 Servidores Big Data – HortonWorks
 - 2 Servidores Microstrategy
 - 2 Servidores Analytics – Weka/SPSS/Jupyter

Celepar

- Clientes
 - Secretarias
 - Fazenda, Educação, Justiça, Cultura, Segurança, Planejamento, Saúde, Identificação, Tribunal de Contas, Agricultura
 - Detran
 - Agência de Fomento
 - Portos
 - Bombeiros
- 19 Bases de dados Distintas, 486 Bi's desenvolvidos e publicados
- Média de 6.000 acessos/mês

Agenda

- Aulas

- 11/05 - Teórica
- 12/05 - Teórica / Prática
- 25/05 - Prática
- 26/05 - Prática / Trabalho

- Trabalho

Agenda

- ◉ Oque é
- ◉ Características
- ◉ Arquitetura
- ◉ Terminologia
- ◉ Instalação e Configuração
 - ◉ Windows
- ◉ Rdd's
- ◉ Transformações e Ações
- ◉ Funções
- ◉ SparkSQL
 - ◉ Hive
- ◉ Spark Streaming
- ◉ Anaconda + Jupyter Notebook

Objetivos

- ◉ Instalação e configuração do Spark + Anaconda + Jupyter Notebook
 - ◉ Windows
 - ◉ VM Virtualbox – Pronta
 - ◉ VM Cloudera 5.10 ou 5.5
 - ◉ Testar Spark – spark-shell
- ◉ Wordcount
 - ◉ Scala
 - ◉ Python
 - ◉ Eclipse - Scala e Java
- ◉ Mtos exercícios
- ◉ Trabalho

Quem somos?

- ◉ Nome?
- ◉ Formação?
- ◉ Onde trabalha?
 - ◉ Possuí sistema de BI ou BigData?
- ◉ Oque faz?
 - ◉ Como gostaria que fosse feito?
 - ◉ Resultados?
- ◉ Hobbies
- ◉ Oque mais achar necessário =)

Oque é?

- Big Data
- Hadoop
 - Maquinas baratas
 - Premissa que máquinas darão problemas
 - Mover código entre maquinas é mto mais eficiente que mover dados.
 - Desenvolvimento focado no problema
 - cluster manager, distributed compute engine, distributed file system
 - Yarn, MapReduce e HDFS
- GFS
- HDFS
- Nosql
- Business Intelligence
- Inteligência Artificial

Spark – Oque é?

- Mecanismo de computação de uso geral para **processamento de dados em larga escala**.
- “uma estrutura de computação em cluster na memória para processar e analisar grandes quantidades de dados.”
- Open Source*
- AMPLab doação para Apache
- Oferece uma camada de abstração de dados
 - Resilient Distributed Datasets – RDD
 - Analise de dados em paralelo
- Alternativa para o Hadoop MapReduce

Spark – Oque é?

- 100x Faster*****
- Escrito em Scala/Java
- Fornece Api's – Scala, Java, Python e R

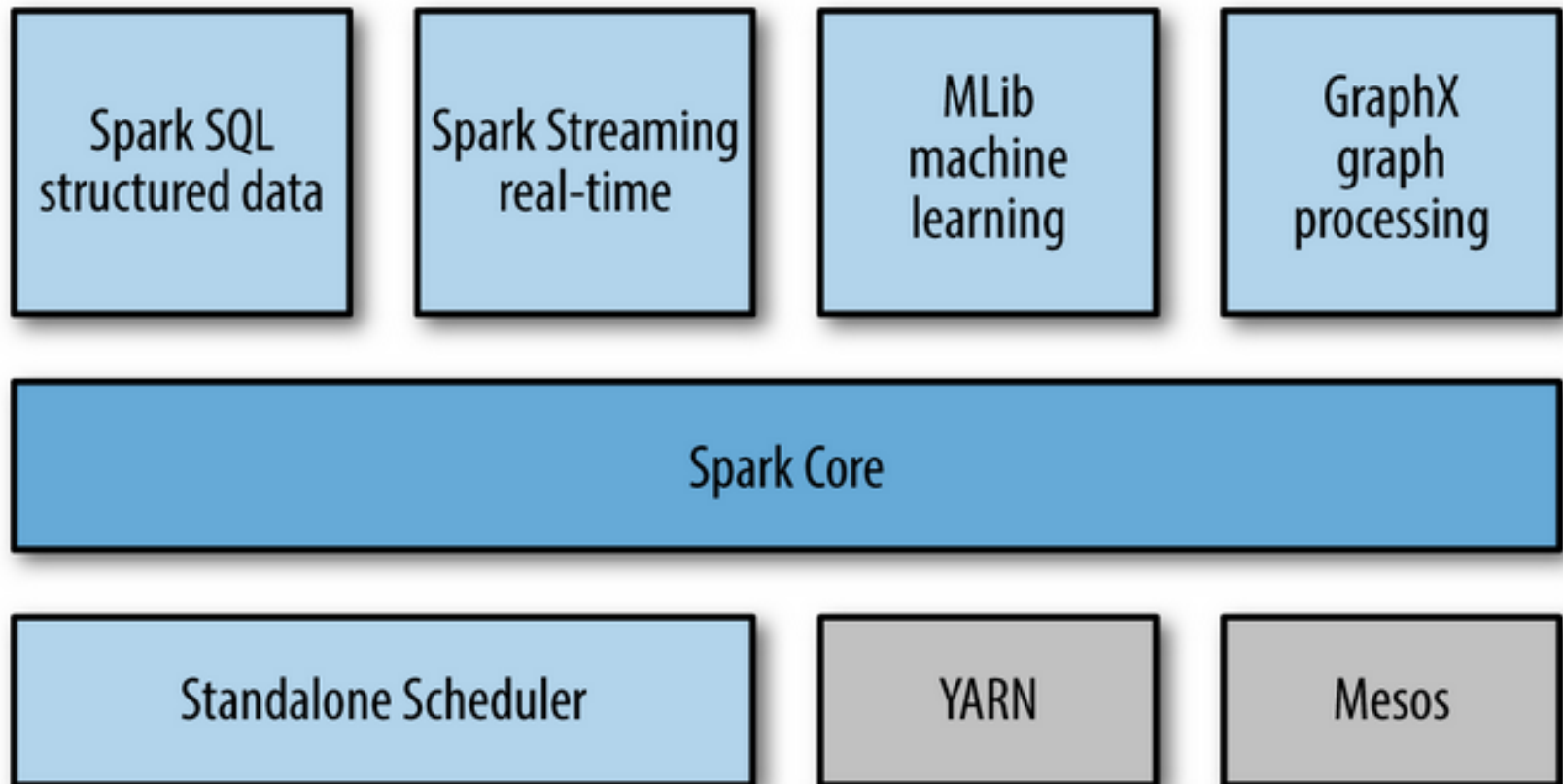
Características

- Fácil Uso;
 - Desenvolver uma aplicação distribuída com o Spark é mais fácil que desenvolver uma rotina de MapReduce;
- Fornece API's prontas
- Mais Rápido que as rotinas de MapReduce
 - Permite processamento em memória
 - 100x mais rápido se o volume de dados couber na memória, senão 10x
- Directed Acyclic Graph (DAG)
- Uso de Propósito Geral
 - Fornece uma plataforma integrada de diferentes tipos de Jobs

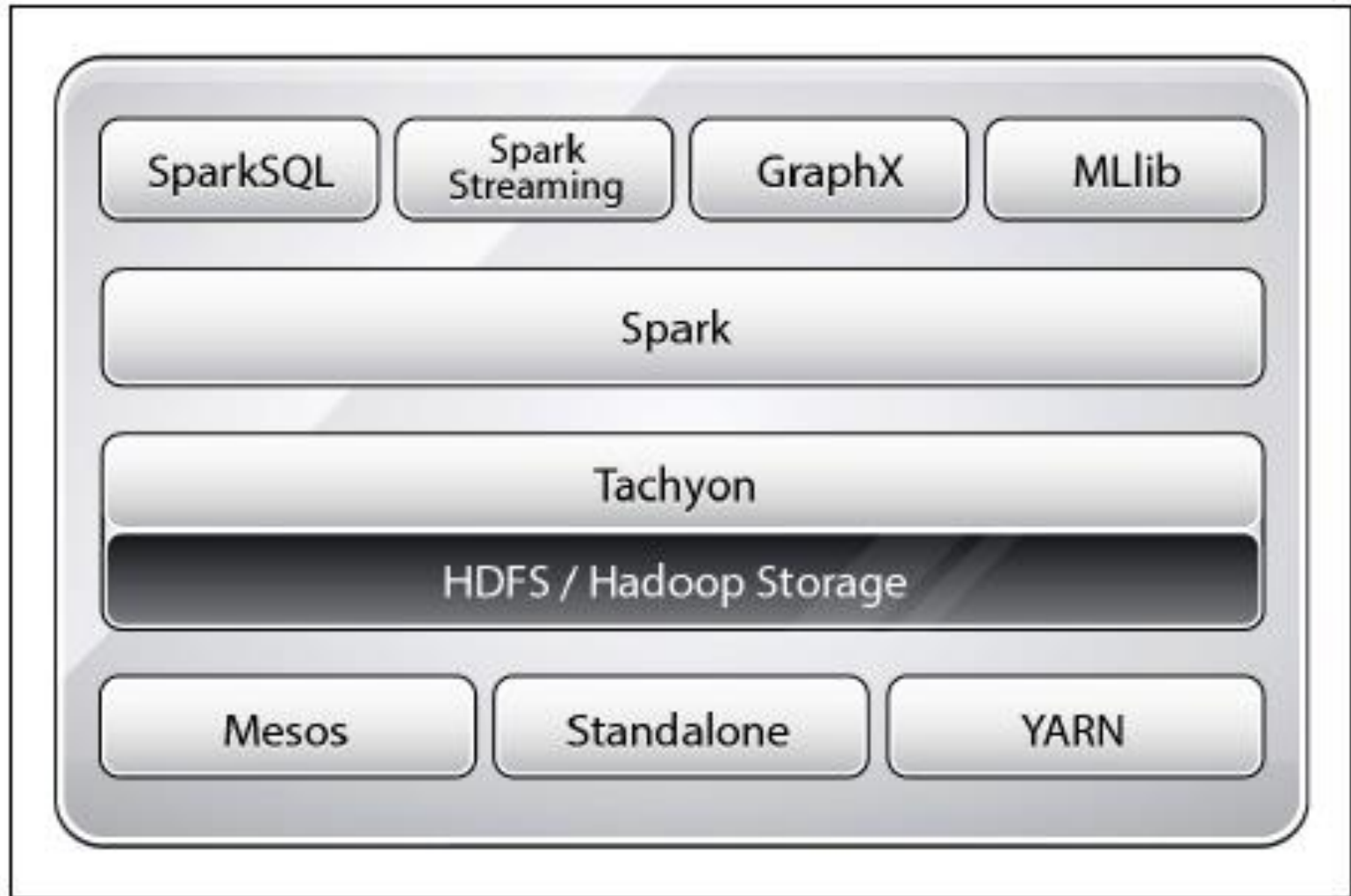
Características

- Escalável
- Tolerante à falhas
 - Pode ser que baixe a performance, porém o processo não para.
 - Não existe a necessidade de se preocupar com as falhas.

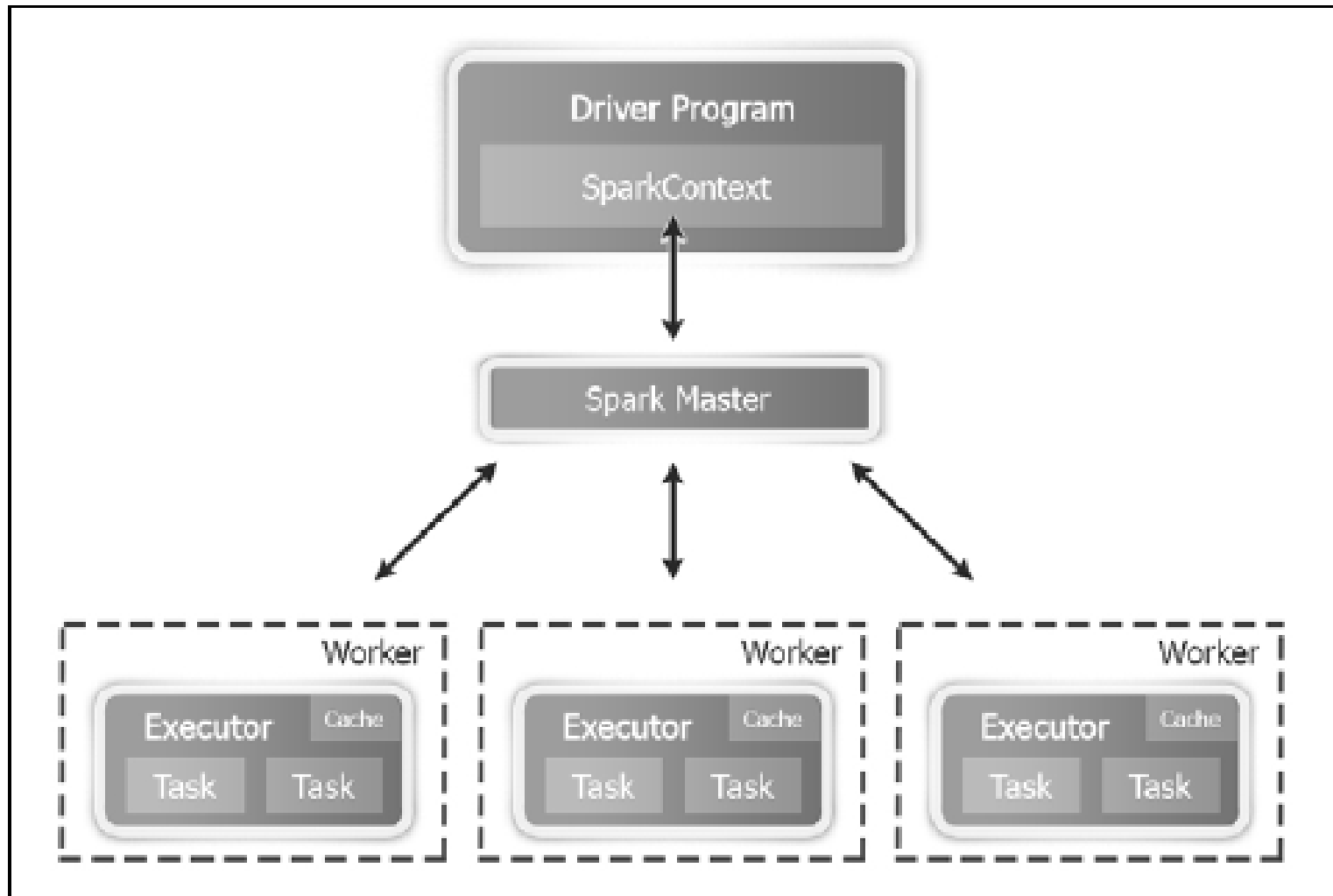
Arquitectura



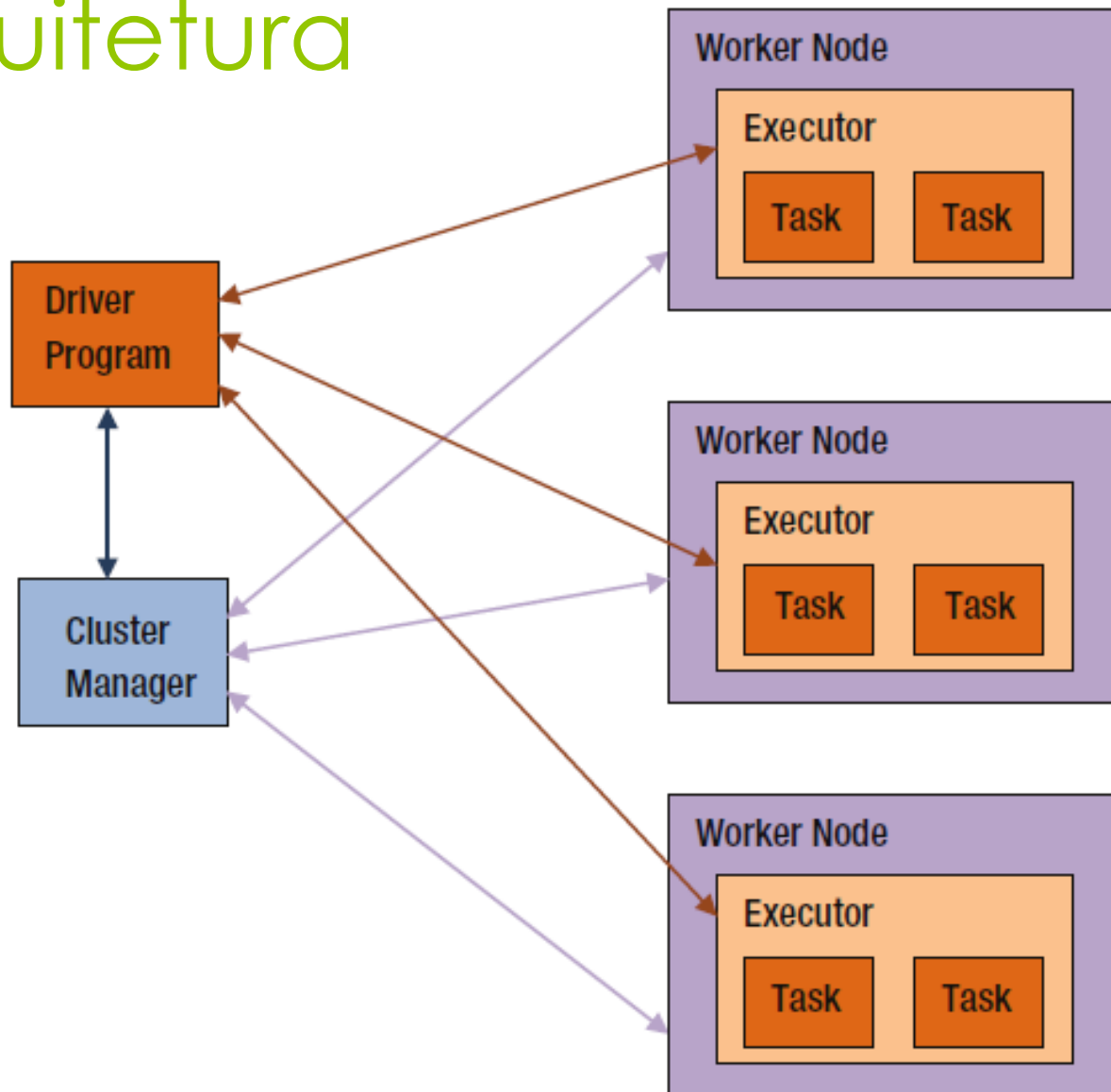
Arquitetura



Arquitetura



Arquitetura



Arquitetura

- **Driver Program**

- É um aplicativo que usa o Spark como uma biblioteca. Ele fornece o código de processamento de dados que o Spark executa nos nós (Worker).
- O Driver Program pode iniciar um ou mais trabalhos em um cluster.

Arquitetura

o Cluster Manager

- o O Spark usa um gerenciador de cluster para **receber os recursos de cluster** para executar um job.
- o Fornece um agendamento de baixo nível dos recursos do cluster através das aplicações.
- o Permite que vários aplicativos compartilhem recursos de cluster e executem as tarefas nos mesmos nós de trabalho (Workers).
- o O Spark atualmente suporta três gerenciadores de cluster: Standalone, Mesos e YARN.
- o Mesos e YARN permitem que você execute aplicativos Spark e Hadoop simultaneamente nos mesmos nós de trabalho.

Arquitetura

◉ Workers

- ◉ Fornece recursos de CPU, memória e armazenamento para o aplicativo.
- ◉ Responsáveis por tornar uma aplicação em um processo distribuído no Cluster.

◉ Executors

- ◉ O executor é o processo JVM (Java Virtual Machine) que o Spark cria em cada worker para executar um aplicativo.
- ◉ Executa o código do aplicativo simultaneamente em várias threads.
- ◉ Também pode armazenar dados em cache na memória ou no disco.
- ◉ Um executor tem o mesmo tempo de vida que o aplicativo para o qual ele é criado. Quando um aplicativo termina, todos os executores criados para ele também terminam.

Arquitetura

○ Tasks

- A task é a menor unidade de trabalho que o Spark envia para um executor.
- Ele é executado por um thread em um executor em um nó de trabalho(workers).
- Cada tarefa executa alguns cálculos para retornar um resultado ao driver program ou particiona sua saída para shuffle.
- O Spark cria uma task por partição de dados.
- Um executor executa uma ou mais tarefas simultaneamente.
- Quantidade de paralelismo é determinado pelo número de partições. Mais partições significam mais task's processando dados em paralelo.

Terminologia

◉ Shuffle

- ◉ O shuffle redistribui dados entre um cluster de nós.
- ◉ É uma operação cara porque envolve a movimentação de dados através de uma rede.
- ◉ Note que o que um shuffle faz não é redistribuir dados aleatoriamente;
- ◉ Ele agrupa os conjuntos de dados em intervalos com base em alguns critérios.
- ◉ Cada conjunto de dados forma uma nova partição

Terminologia

o Job

- o Um job é um conjunto de cálculos que o Spark executa para retornar resultados para um program driver.
- o Essencialmente, é uma execução de um algoritmo de processamento de dados em um cluster.
- o Um aplicativo pode iniciar vários jobs.

Terminologia

◉ Stage

- ◉ É uma coleção de tasks.
- ◉ O Spark divide um trabalho em um DAG de estágios.
- ◉ Um stage pode depender de outro.
- ◉ Por exemplo, um trabalho pode ser dividido em dois estágios, estágio 0 e estágio 1, em que o estágio 1 não pode começar até que o estágio 0 seja concluído.
- ◉ Spark agrupa as task's em stages usando limites do shuffle.
- ◉ Tasks que não exigem um shuffle são agrupados no mesmo stage. Uma tarefa que requer que seus dados de entrada sejam embaralhados começa um novo estágio.

Ta e dae???

- Quando um aplicativo é executado, o Spark se conecta a um **cluster manager** e adquire **executors** nos **worker's**.
- A aplicação envia o algoritmo de processamento de dados como um **job**.
- O Spark divide um trabalho em **stages** gerando um gráfico acíclico direcionado (DAG - directed acyclic graph) .
- Após, ele agenda a execução desses stages nos executors usando um planejador de baixo nível fornecido pelo cluster manager.
- Enfim os executors executam as tarefas enviadas pelo Spark em paralelo.

Ta e dae 2???

- Cada aplicativo obtém seu próprio conjunto de executores nos nós do worker.
- Este design fornece alguns benefícios.
 - Primeiro, as tasks de diferentes aplicativos **são isoladas umas das outras**, uma vez que elas são executadas em diferentes processos da JVM. Uma task com mal funcionamento não pode travar outro aplicativo.
 - Segundo, agendamento de tarefas torna-se mais fácil porque o spark precisa agendar as tarefas pertencentes a apenas um aplicativo de cada vez. Ele não precisa lidar com as complexidades das tarefas de agendamento a partir de vários aplicativos em execução simultaneamente.

Hummmmm!!!!

- No entanto, esse design também tem uma desvantagem.
- Como os aplicativos são executados em processos separados da JVM, **eles não conseguem compartilhar dados com facilidade.**
- Mesmo que eles possam estar em execução nos mesmos nós de trabalho, eles não podem compartilhar dados sem gravá-lo em disco.
- Lembrando que escrever e ler dados do disco são operações caras.
- Portanto, os aplicativos que compartilham dados por meio do disco terão problemas de desempenho.

Instalação e Configuração

Spark Context

- SparkContext é uma classe definida na biblioteca sendo o principal ponto de entrada para uso das bibliotecas Spark.
- Ele representa uma conexão com um cluster Spark.
- Também é necessário para criar outros objetos importantes fornecidos pela API Spark.
- Um aplicativo deve criar uma instância da classe SparkContext.
- Normalmente um aplicativo pode ter apenas uma instância ativa do SparkContext.
- Para criar outra instância, primeiro deve parar a instância ativa.
- A classe SparkContext fornece vários construtores. O mais simples não aceita argumentos.

RDD

- Resilient Distributed Datasets
- O RDD representa uma coleção de elementos de dados particionados que podem ser operados em paralelo em diferentes nós do cluster.
- É o mecanismo primário de abstração, definido como uma classe abstrata na biblioteca Spark.
- Conceitualmente, o RDD é semelhante a uma coleção de dados, exceto pelo fato de representar um conjunto de dados distribuídos e suportar operações do tipo **lazy**.
- Quando um arquivo é lido usando o método `textFile` do `SparkContext`, o Spark não lê imediatamente o arquivo do disco. Da mesma forma, as transformações RDD, que retornam um novo RDD são computadas.

Lazy

- O Spark faz apenas uma anotação de como um RDD foi criado e as transformações aplicadas a ele
Ele usa essas informações para construir ou reconstruir um RDD quando necessário.

Aplicativos Spark

- Imutável – Uma vez criado, não pode ser modificado no local. Basicamente, uma operação que modifica um RDD retorna um novo RDD.
- Os aplicativos do Spark processam dados usando os métodos definidos na classe RDD ou classes derivadas dele. Esses **métodos** também são conhecidos como **operações**.
- Processamento independente do volume de dados
- Processamento independente se é local ou distribuído
- Podem ser categorizadas de duas maneiras, **Transformação e Ação**.
 - Transformação cria um novo RDD
 - Ação retorna um valor para o Manager Driver

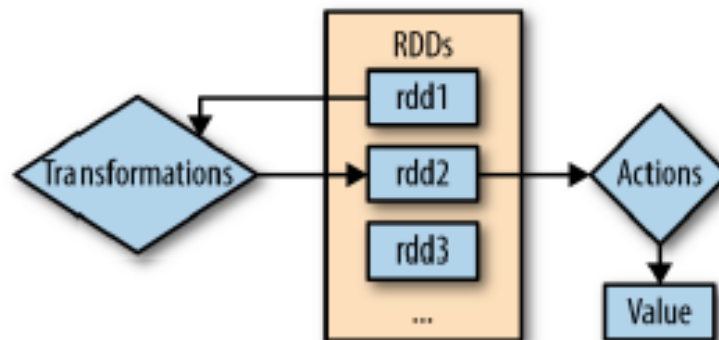
Transformações e Ações

- Transformações

- Retornam um novo RDD baseado no original. API do Spark incluem `map ()`, `filter ()`, `sample ()` e `Union()`.

- Ações

- Este tipo de operação retorna um valor baseado em algum tipo de processamento, sendo executada em um RDD. As ações disponíveis através da API Spark incluem `reduce ()`, `count ()`, `first ()` e `foreach ()`.



Spark-shell

- ◉ Spark-shell
- ◉ :help
- ◉ :quit
- ◉ :paste
 - ◉ Ctrl-D

- ◉ `val xs = (1 to 1000).toList`
- ◉ `val xsRdd = sc.parallelize(xs)`
- ◉ `val evenRdd = xsRdd.filter{ _ % 2 == 0 }`
- ◉ `val count = evenRdd.count`
- ◉ `val first = evenRdd.first`

Transformações

- `val xs = (1 to 10000).toList`
- `val rdd = sc.parallelize(xs)`
- Leitura de um arquivo
 - `val rdd = sc.textFile("C:/Dados/movies.txt")`
 - `rdd.collect().foreach(println)`
- HDFS
 - `val rdd = sc.textFile("hdfs://namenode:9000/path/to/directory/*.gz")`
 - `val rdd = sc.wholeTextFiles("path/to/my-data/*.txt")`

Transformações

- map

- `val rdd = sc.textFile("C:/Dados/movies.txt")`
- `val lengths = rdd map { l => l.length }`
- `lengths.collect().foreach(println)`
- `val lengths = rdd.map(w => (w,1))`

- Filter

- `val rdd = sc.textFile("C:/Dados/movies.txt")`
- `val lines = rdd filter { l => l.length < 27 }`
- `val one= rdd filter { l => l.contains("One") }`
- `val year= rdd filter { l => l.contains("1993") }`

Transformações

- FlatMap

- val rdd = sc.textFile("C:/Dados/movies.txt")
- val words = rdd.flatMap { l => l.split(",") }
- val wordsf = rdd.flatMap(_.split("\\W+"))
- val wordsMap = wordsf.map(w => (w,1))

- reduceByKey

- val words = rdd.flatMap { line => line.split(" ") }
- val wordPairs = words.map { word => (word, 1) }
- val wordCounts = wordPairs.reduceByKey(_ + _)

- Sort

- sortByKey(true)
- sortBy(_._2, false)

Transformações

- Union

- `val rdd = sc.textFile("C:/Dados/movies.txt")`
- `val rdd2 = sc.textFile("C:/Dados/movies2.txt")`
- `val linesFromBothFiles = rdd.union(rdd2)`

- Intersection

- `val linesPresentInBothFiles = rdd.intersection(rdd2)`

- Subtract

- `val linesInFile1Only = rdd.subtract(rdd2)`

- Distinct

- `val uniqueNumbers = rdd2.distinct`

Transformações

Function name	Purpose	Example	Result
<code>map()</code>	Apply a function to each element in the RDD and return an RDD of the result.	<code>rdd.map(x => x + 1)</code>	{2, 3, 4, 4}
<code>flatMap()</code>	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	<code>rdd.flatMap(x => x.to(3))</code>	{1, 2, 3, 2, 3, 3, 3}
<code>filter()</code>	Return an RDD consisting of only elements that pass the condition passed to <code>filter()</code> .	<code>rdd.filter(x => x != 1)</code>	{2, 3, 3}
<code>distinct()</code>	Remove duplicates.	<code>rdd.distinct()</code>	{1, 2, 3}
<code>sample(withReplacement, fraction, [seed])</code>	Sample an RDD, with or without replacement.	<code>rdd.sample(false, 0.5)</code>	Nondeterministic

Function name	Purpose	Example	Result
<code>union()</code>	Produce an RDD containing elements from both RDDs.	<code>rdd.union(other)</code>	{1, 2, 3, 3, 4, 5}
<code>intersection()</code>	RDD containing only elements found in both RDDs.	<code>rdd.intersection(other)</code>	{3}
<code>subtract()</code>	Remove the contents of one RDD (e.g., remove training data).	<code>rdd.subtract(other)</code>	{1, 2}
<code>cartesian()</code>	Cartesian product with the other RDD.	<code>rdd.cartesian(other)</code>	{(1, 3), (1, 4), ..., (3, 5)}

Exercicio 1

- ◉ Ler da pasta Dados\E1\texto.txt
- ◉ Identificar quais palavras mais ocorrem no texto
 - ◉ Quebrar o texto em um array
 - ◉ atribuir o valor 1 para cada item do array
 - ◉ Agrupar por chave
 - ◉ Ordenar
 - ◉ Ordenar de novo
 - ◉ Serializar os resultados

Ações

- Ações são métodos RDD que retornam um valor para o Driver Manager.
- Collect****
 - `val rdd = sc.parallelize((1 to 10000).toList)`
 - `val filteredRdd = rdd filter { x => (x % 1000) == 0 }`
 - `val filterResult = filteredRdd.collect`
- Count
 - `val rdd = sc.parallelize((1 to 10000).toList)`
 - `val total = rdd.count`
- CountByValue
 - `val rdd = sc.parallelize(List(1, 2, 3, 4, 1, 2, 3, 1, 2, 1))`
 - `val counts = rdd.countByValue`

Ações

- first

- `val rdd = sc.parallelize(List(10, 5, 3, 1))`
- `val firstElement = rdd.first`

- max

- `val rdd = sc.parallelize(List(2, 5, 3, 1))`
- `val maxElement = rdd.max`

- min

- `val rdd = sc.parallelize(List(2, 5, 3, 1))`
- `val minElement = rdd.min`

- top

- `val rdd = sc.parallelize(List(2, 5, 3, 1, 50, 100))`
- `val largest3 = rdd.top(3)`

Ações

- mean

- val numbersRdd = sc.parallelize(List(2, 5, 3, 1))
 - val mean = numbersRdd.mean

- stdev

- val numbersRdd = sc.parallelize(List(2, 5, 3, 1))
 - val stdev = numbersRdd.stdev

- sum

- val numbersRdd = sc.parallelize(List(2, 5, 3, 1))
 - val sum = numbersRdd.sum

- variance

- val numbersRdd = sc.parallelize(List(2, 5, 3, 1))
 - val variance = numbersRdd.variance

Ações

- Salvando os resultados
 - `val numbersRdd = sc.parallelize((1 to 10000).toList)`
 - `val filteredRdd = numbersRdd filter { x => x % 1000 == 0 }`
 - `filteredRdd.saveAsTextFile("c:/Dados/output")`

Ações

Function name	Purpose	Example	Result
<code>collect()</code>	Return all elements from the RDD.	<code>rdd.collect()</code>	<code>{1, 2, 3, 3}</code>
<code>count()</code>	Number of elements in the RDD.	<code>rdd.count()</code>	4
<code>countByValue()</code>	Number of times each element occurs in the RDD.	<code>rdd.countByValue()</code>	<code>{(1, 1), (2, 1), (3, 2)}</code>

Ações

<code>take(num)</code>	Return num elements from the RDD.	<code>rdd.take(2)</code>	{1, 2}
<code>top(num)</code>	Return the top num elements the RDD.	<code>rdd.top(2)</code>	{3, 3}
<code>takeOrdered(num)(ordering)</code>	Return num elements based on provided ordering.	<code>rdd.takeOrdered(2)(myOrdering)</code>	{3, 3}
<code>takeSample(withReplacement, num, [seed])</code>	Return num elements at random.	<code>rdd.takeSample(false, 1)</code>	Nondeterministic
<code>reduce(func)</code>	Combine the elements of the RDD together in parallel (e.g., sum).	<code>rdd.reduce((x, y) => x + y)</code>	9
<code>fold(zero)(func)</code>	Same as <code>reduce()</code> but with the provided zero value.	<code>rdd.fold(0)((x, y) => x + y)</code>	9
<code>aggregate(zeroValue)(seqOp, combOp)</code>	Similar to <code>reduce()</code> but used to return a different type.	<code>rdd.aggregate((0, 0))((x, y) => (x._1 + y, x._2 + 1), (x, y) => (x._1 + y._1, x._2 + y._2))</code>	(9, 4)
<code>foreach(func)</code>	Apply the provided function to each element of the RDD.	<code>rdd.foreach(func)</code>	Nothing

Funções

◉ Sintaxe

```
def functionName ([list of parameters]) : [return type] = {  
    function body  
    return [expr]  
}
```

◉ :paste

```
def addInt( a:Int, b:Int ) : Int = {  
    var sum:Int = 0  
    sum = a + b  
    return sum  
}
```

- ◉ Ctrl+d

- ◉ addInt(1,2)

Spark Jobs

- Um Job é um conjunto de cálculos que o Spark realiza para retornar os resultados de uma Action a um Manage Driver. Um aplicativo pode iniciar um ou mais Jobs.
- Ele lança um Job chamando uma Action de um RDD. Portanto, uma Action aciona uma Job.
- Se uma ação é chamada para um RDD que não está em cache ou um descendente de outro RDD, o Job começa com a leitura de dados de um sistema de armazenamento.
- No entanto, se uma ação for solicitada um RDD armazenado em cache ou um descendente de um RDD em cache, um job começa a partir do ponto em que o RDD ou seu ancestral RDD foi armazenado em cache.
- Em seguida, o Spark aplica as transformações necessárias para criar o RDD cuja ação método foi chamado.
- Finalmente, executa os cálculos especificados pela ação.
- Um Job é concluído quando um resultado é retornado para um Manage Driver.
- Quando um aplicativo chama uma Action RDD, o Spark cria um DAG de Stages de tarefas.
- Agrupa tarefas em Stages usando limites de shuffle.
- Tarefas que não exigem shuffle são agrupadas no mesmo Stage.
- Uma tarefa que requer que seus dados de entrada sejam embaralhados começa um novo Stage.
- Um Stage pode ter uma ou mais tarefas. O Spark envia tarefas para os executores, que executam as tarefas em paralelo.
- As tarefas são agendadas nos nós com base na localidade dos dados. Se um nó falhar durante o Job em uma tarefa, o Spark reenviará tarefa para outro nó.

Exercicio 2

- Ler da pasta Dados\E2\logs.txt
- Passar todas palavras para minúsculas
- Pegar apenas as linhas que contém a palavra “erro”
- Pegar apenas as linhas que iniciam com a palavra “erro”
 - `val errorLogs = logs.filter{_.split("\\s+")(0) }`
- Salvar os resultados

Pratica Movies

- `val rawLogs = sc.textFile("c:/Dados/movies.txt")`
- `val lengths = rawLogs.map{line => line.size}`
- `val maxLen = lengths.reduce{ (a, b) => if (a > b) a else b }`
- Qual seria a outra maneira de conseguir o valor máximo do array?
- `val wordCounts = rawLogs map {line => line.split(" "\s+").size}`
- `ll.saveAsTextFile("c:/Dados/movielogs")`

Exercício 3

- Utilizando as informações da Pasta Dados/E2/registrosPlacas.txt, levantar as seguintes informações:
 - Preparar os dados
 - Identificar o veículo que tem mais registros
 - Identificar qual veículo teve a maior velocidade
 - Identificar qual veículo teve a maior velocidade média

Spark SQL

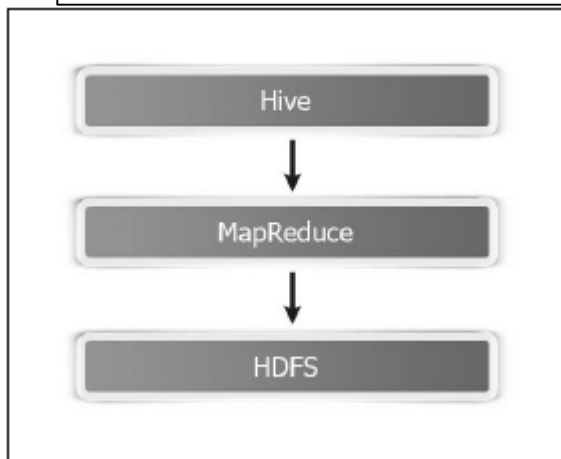
- Um dos principais motivos do Spark se tornar popular
- Existem muito mais pessoas que sabem Sql do que Spark, Scala etc
- Ajuda a criar e executar programas Spark mais rapidamente. Ele permite que os desenvolvedores escrevam **menos código**, que o **programa leia menos dados** e deixe o otimizador do **Catalyst** todo o trabalho pesado.
- Spark Sql é uma biblioteca que roda sobre o Spark, provendo uma camada abstrata para processamento de dados estruturados, nosql...
- Fornece integração com as funções mais conhecidas, **como filtros, agregações, seleções de colunas, joins etc**
- O Spark Sql pode ser utilizada como uma biblioteca para processamento em Scala, Java, Python ou R
- Os formatos de arquivos suportados pelo Spark SQL incluem CSV, JSON, Parquet, ORC e Avro.
- Bancos com conectores JDBC, entre eles:
- PostgreSQL, MySQL, H2, Oracle, DB2, MS SQL Server, HBase,
- Cassandra, Elasticsearch, Druid, e outros NoSQL

Spark SQL

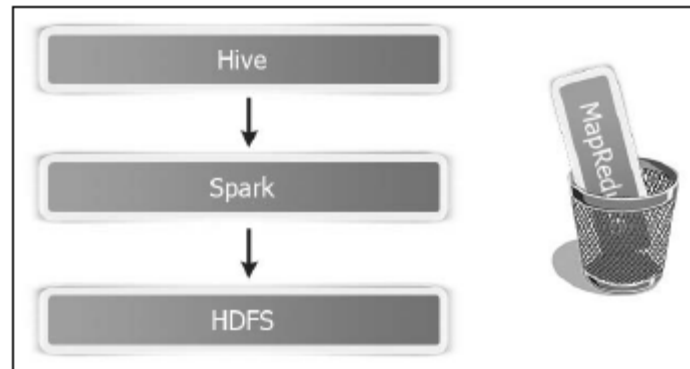
- Integração com o Hive
 - Funciona com ou sem o Hive
- Armazenamento Colunar – Não existe a necessidade de ler todas as colunas.
- Caching In Memory
 - Compressão de dados
- Skip Rows – Faz uso do Statistics dos Gerenciadores de BD para não ler informações desnecessárias
- Predicate Pushdown – Objetivo de reduzir o I/O, aplica filtros e se possível utiliza índices nativos do BD
- Query Optimization – Realiza otimização das queries antes de executá-las
 - Catalyst – Separa a query em 4 etapas
 - Análise
 - Otimização Lógica
 - Planejamento Físico
 - Geração de Código

Spark SQL

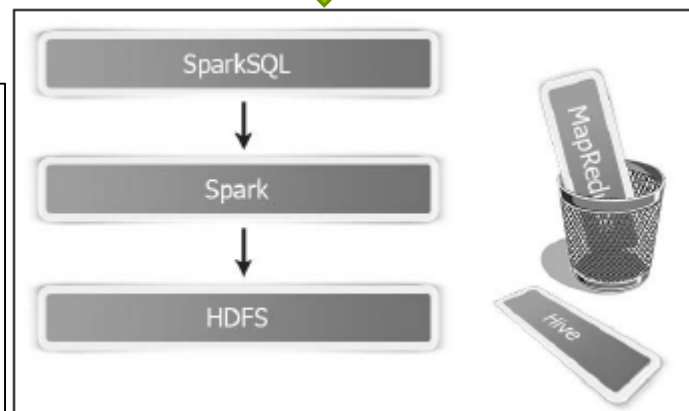
Hive – Conversão de queries SQL em MapReduce



O Shark substituiu a parte MapReduce pelo Spark, mantendo a maior parte da base de código.



Inicialmente, funcionou bem, mas muito em breve, os desenvolvedores do Spark atingiram os limites e não conseguiram otimizar. Finalmente, eles decidiram escrever o SQL Engine do zero e que deu origem ao Spark SQL.



Spark SQL

- Spark Sql compila partes da query diretamente em bytecode Java, utilizando Scala(AST – Abstract Syntax Tree)
- Possibilidade de escrever queries em qq linguagem, o spark irá contruir um plano otimizado.
- O código gerado, normalmente, é mais rapido do que os criados/Tunados manualmente em Scala, Java ou Python
- *One of the benefits of code generation is that you can write a Spark SQL application in any supported language without worrying about performance. **For example, Python applications generally run slower than Java or Scala applications. However, a Spark SQL application written in Python will process data as fast as one written in Scala.***

SchemaRdd

- Tanto o carregamento de dados quanto a execução de consultas retornam SchemaRDDs. Os SchemaRDDs são semelhantes as tabelas em um banco de dados tradicional.
- Superficialmente, um SchemaRDD é um RDD composto de linhas de objetos com informações adicionais do esquema e dos tipos em cada coluna.
- Uma observação importante: em futuras versões do Spark, o nome SchemaRDD = DataFrame.
- Os SchemaRDDs também são RDDs regulares, para que você possa operá-los usando o RDD existente transformações como `map ()` e `filter ()`. No entanto, eles fornecem várias capacidades.
- possibilidade de salvar qualquer SchemaRDD como uma tabela temporária

Spark SQL

- ◉ Comparando os DataFrames com os RDDs.
- ◉ Um RDD é uma coleção de objetos sem ideia sobre o formato dos dados.
- ◉ Os DataFrames têm um esquema associado com eles, sendo assim, também podemos olhar para DataFrames como RDDs com o esquema adicionado a eles.
- ◉ Até o Spark 1.2, havia um artefato chamado SchemaRDD, que agora evoluiu para DataFrame. Eles fornecem funcionalidades muito mais ricas do que os SchemaRDDs.

Spark SQL

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
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,	Map	Map	dict

Spark SQL - Pratica

- Utilizando os dados do arquivo movies.txt
- `val hc = new org.apache.spark.sql.hive.HiveContext(sc)`
- `hc.sql("create table if not exists movie(cod int, name string, year int, note float, views int) row format delimited fields terminated by ', '")`
- `hc.sql("load data local inpath \"c:/Dados/movies.txt\" into table movie")`
- `val mov = hc.sql("from movie select name,year,views")`
- `val mov = hc.sql("select * from movie where year=1993").collect().foreach(println)`

Exercício 3.1

- Utilizando as informações da Pasta Dados/E2, levantar as seguintes informações:
 - Tratar o arquivo
 - Identificar o veículo que tem mais registros
 - Identificar qual veículo teve a maior velocidade
 - Identificar qual veículo teve a maior velocidade média

Spark SQL - Hive

- o Cloudera

- o `sudo cp usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/`

- o `val hc = new org.apache.spark.sql.hive.HiveContext(sc)`

- o `val hvdata = hc.sql (" show databases")`

- o `val hvdata = hc.sql (" show tables")`

- o Por padrão, todas as tabelas criadas pelo Spark SQL são tabelas gerenciadas pelo Hive, ou seja, o Hive tem controle completo sobre o ciclo de vida de uma tabela, incluindo a exclusão se os metadados da tabela forem eliminados usando o comando `drop table`. Isso vale apenas para tabelas persistentes.

Exercícios Hive

- ◉ Importar dados p/ HDFS
- ◉ Criar tabela no Hive
- ◉ Tratar os dados
 - ◉ Arquivo CID
 - ◉ Arquivo CSV de 10Gb
- ◉ CREATE DATABASE/SCHEMA, TABLE, VIEW, FUNCTION, INDEX
- ◉ DROP DATABASE/SCHEMA, TABLE, VIEW, INDEX
- ◉ TRUNCATE TABLE
- ◉ ALTER DATABASE/SCHEMA, TABLE, VIEW
- ◉ MSCK REPAIR TABLE (or ALTER TABLE RECOVER PARTITIONS)
- ◉ SHOW DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, VIEWS, PARTITIONS, FUNCTIONS, INDEX[ES], COLUMNS, CREATE TABLE
- ◉ DESCRIBE DATABASE/SCHEMA, table_name, view_name

Hive

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>

Spark Streaming

- O Spark Streaming provê Api para processamento em tempo real de big data - ou seja, análise em tempo real -
- Latência muito baixa, normalmente alguns segundos.
- Streaming é o processo de dividir dados de entrada em fluxo contínuo em unidades discretas, que pode ser processado facilmente.
- Exemplos mais comuns são streaming de vídeo e áudio, dados de vendas, segurança etc
- A solução implementada é transferir dados em pequenos blocos que começam a ser reproduzidos para o usuário, enquanto o restante dos dados estão sendo baixados em segundo plano.
- A taxa na qual um aplicativo de streaming recebe dados é expresso na forma de kilobytes por segundo (kbps) ou megabytes por segundo (mbps).

Spark Streaming

- `import org.apache.spark.SparkConf`
- `import org.apache.spark.streaming.{Seconds, StreamingContext}`
- `import org.apache.spark.storage.StorageLevel`
- `import StorageLevel._`
- `import org.apache.spark._`
- `import org.apache.spark.streaming._`
- `import org.apache.spark.streaming.StreamingContext._`

Spark Streaming

- `val ssc = new StreamingContext(sc, Seconds(5))`
- `val lines = ssc.textFileStream("file:///C:/Dados/E5/")`
- `val wordsFlatMap = lines.flatMap(_.split(" "))`
- `val wordsMap = wordsFlatMap.map(w => (w,1))`
- `val wordCount = wordsMap.reduceByKey((a,b) => (a+b))`
- `wordCount.print`
- `ssc.start`

Exercício Streaming

- Criar um aplicativo com uma funcionalidade diferente
- Ex:
 - Valores Máximos e Mínimos
 - Leitura de outros tipos de arquivos
 - Processo de Leitura do HDFS

Anaconda – Python

Anaconda – Python

- C:\Spark\Projetos
- Pyspark
- Sc

Jupyter - Map

- `map()` é uma função *builtin* de Python, isto é, uma função que é implementada diretamente no interpretador Python, podendo ser utilizada sem a importação de um módulo específico. Essa função, em Python, serve para **aplicar uma função a cada elemento de uma lista**, retornando uma nova lista contendo os elementos resultantes da aplicação da função

Jupyter - Map

- `lista = [2,4,6,8,10]`

```
def soma_dez(num):  
    return num + 10
```

```
for elemento in lista:  
    print(soma_dez(elemento))
```

- **`lista_map = map(soma_dez, lista)`**
- `print(lista_map)`
- `print(list(lista_map))`

Jupyter - Reduce

- `reduce()` é outra função *builtin* do Python (deixou de ser *builtin* e passou a estar disponível no módulo *functools* a partir da versão 3000). Sua utilidade está na aplicação de uma função a todos os valores do conjunto, de forma a **agregá-los todos em um único valor**. Por exemplo, para aplicar a operação de soma a todos os elementos de uma sequência, de forma que o resultado final seja a soma de todos esses elementos, poderíamos fazer o seguinte:

```
import operator
valores = [1, 2, 3, 4, 5]
soma = reduce(operator.add, valores)
print soma
```

Jupyter - Filter

- Como o próprio nome já deixa claro, `filter()` filtra os elementos de uma sequência. O processo de filtragem é definido a partir de uma função que o programador passa como primeiro argumento da função. Assim, `filter()` só “deixa passar” para a sequência resultante aqueles elementos para os quais a chamada da função que o usuário passou retornar `True`.

Jupyter - Filter

- `lista = [1,2,3,4,5,6,7,8,9]`

- `print(lista)`

```
def funpar(num):  
    return num % 2 == 0
```

- `funpar(500)`

- Utilizando Filter, retorna apenas os verdadeiros

- `list(filter(funpar, lista))`

Exercício 4

- ◉ Contexto SQL Jupyter
- ◉ Dados a base E4
- ◉ Identificar:
 - ◉ Quais condutores cometeram mais infrações?
 - ◉ Quais as infrações mais cometidas?
 - ◉ Por ano
 - ◉ Por Gênero
 - ◉ Classe de carteira
 - ◉ Tipo de restrição
- ◉ Quais condutores perderam a carteira nesse período?

Exercício 5

- Contexto SQL Jupyter
- Dados a base grande
 - Tratar no HDFS
 - Acessar com Spark e levantar maiores salários



Muito Obrigado!!!

- ◉ evertoncesario@hotmail.com
- ◉ 41 99958-3361