



Minicurso

Introdução ao Processamento
Paralelo e Distribuído Utilizando o
Apache Spark

M.Sc. Fernando Pereira Gonçalves de Sá

Overview



1. Apresentação
2. Introdução
3. Ecossistema do Big Data
4. Apache Spark
5. Sistemas Paralelos e Distribuídos
6. Arquitetura do Apache Spark
7. Transformações
8. APIs do Apache Spark
9. Conclusão

Apresentação

Quem sou eu?



Sobre o instrutor:

Linkedin: <https://www.linkedin.com/in/fpgdesa/>

E-mail: fpgdesa@gmail.com

Acadêmico:

Atualmente, aluno do curso de doutorado em Computação (POSGRAD-IC/UFF). Tema de

pesquisa: **Detecção de Anomalias...**

- ▶ Engenheiro de Produção (UFF)
- ▶ Mestre em Engenharia Elétrica (COPPE/UFRJ)
- ▶ Mestre em Ciência da Computação (CEFET/RJ)

Profissional:

- ▶ Cientista de Dados CEPEL/Eletrobras

Introdução

Introdução



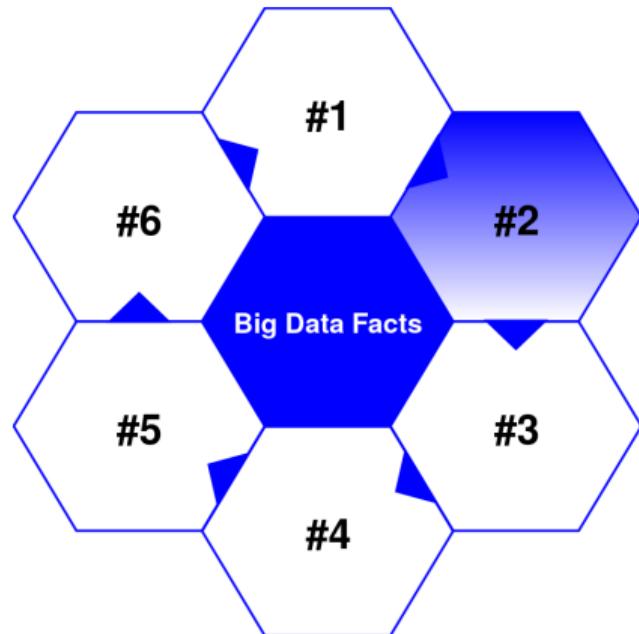
A cada 2 dias, a humanidade cria mais informações do que a quantidade que foi criada desde o seu início até 2003.



Introdução



Cerca de 90 % dos dados no mundo
foram criados nos últimos 2 anos.



Introdução



Espera-se que, em 2020, a quantidade de informações digitais armazenadas crescerá dos atuais 3.2 zettabytes para cerca de 40 zettabytes.



Introdução



A quantidade de dados sendo adquiridos e armazenados pela indústria dobra a cada 1.2 anos.



Introdução



A cada minuto, enviamos 204 milhões de *e-mails*, geramos 1,8 milhões de *likes* no Facebook, enviamos 278 mil *tweets*, e carregamos 200,000 fotos para o Facebook.



Introdução



O Google processa, em média,
mais de 40 mil consultas por
segundo, acumulando mais de 3.5
bilhões em um único dia.



Introdução



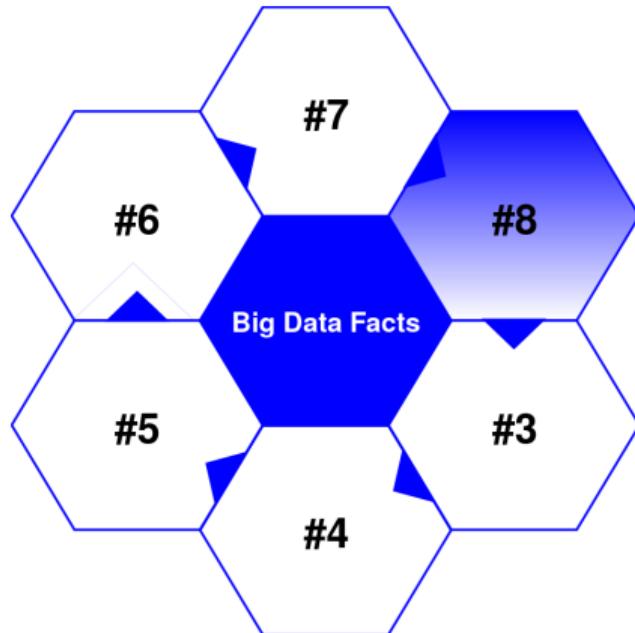
Cerca de 100 horas de vídeo são carregados no YouTube a cada minuto. Demoraríamos 15 anos para assistir a todo o conteúdo carregado na plataforma em um único dia.



Introdução



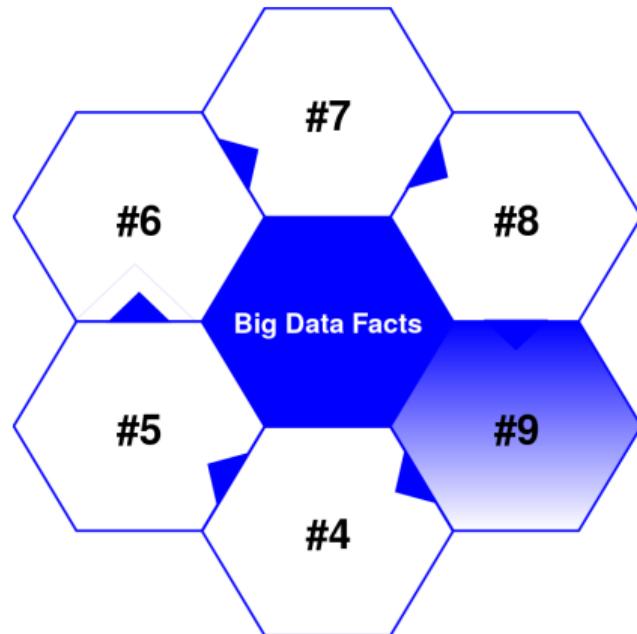
Se gravássemos em DVDs toda informação digital gerada em único dia, a quantidade empilhada de discos teria uma altura equivalente a duas vezes a distância da Terra à Lua.



Introdução



Os data centers ocupam uma área equivalente a 6,000 campos de futebol.



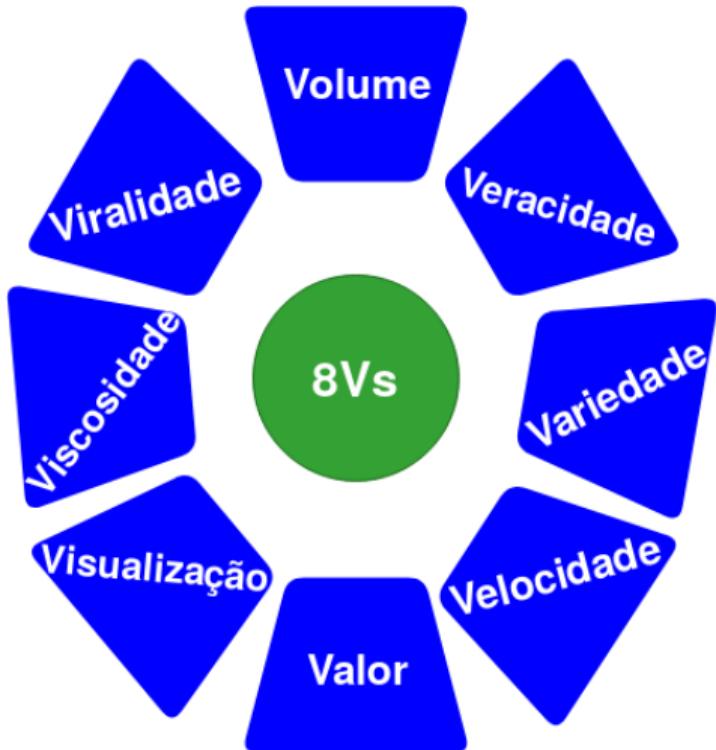
Introdução



O número de *bits* de informação armazenada excedeu a quantidade conhecida de estrelas em 2007.



Introdução



Introdução



- ▶ “*Big Data* é determinado por conjuntos de dados cujo tamanho excede a habilidade dos típicos gerenciadores de banco de dados para realizar a captura, o armazenamento, o gerenciamento e a análise” McKinsey & Company

Introdução



- ▶ “*Big Data* é determinado por conjuntos de dados cujo tamanho excede a habilidade dos típicos gerenciadores de banco de dados para realizar a captura, o armazenamento, o gerenciamento e a análise” **McKinsey & Company**
- ▶ “Atualmente, *Big Data* pode ser definido como um tópico que, além de envolver dados, se refere a um amplo conjunto de técnicas, ferramentas e frameworks de processamento” [Samadi et al. 2016]

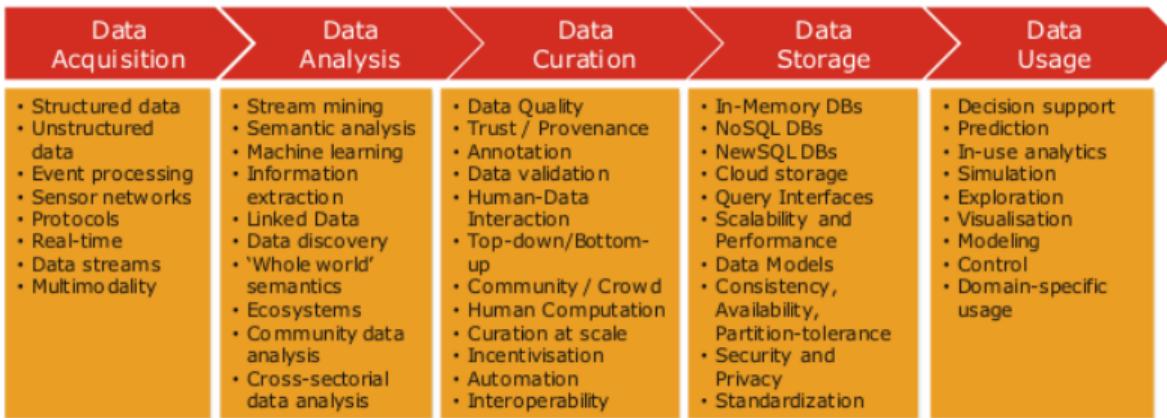
Introdução

Qual o valor que o *Big Data* gera?



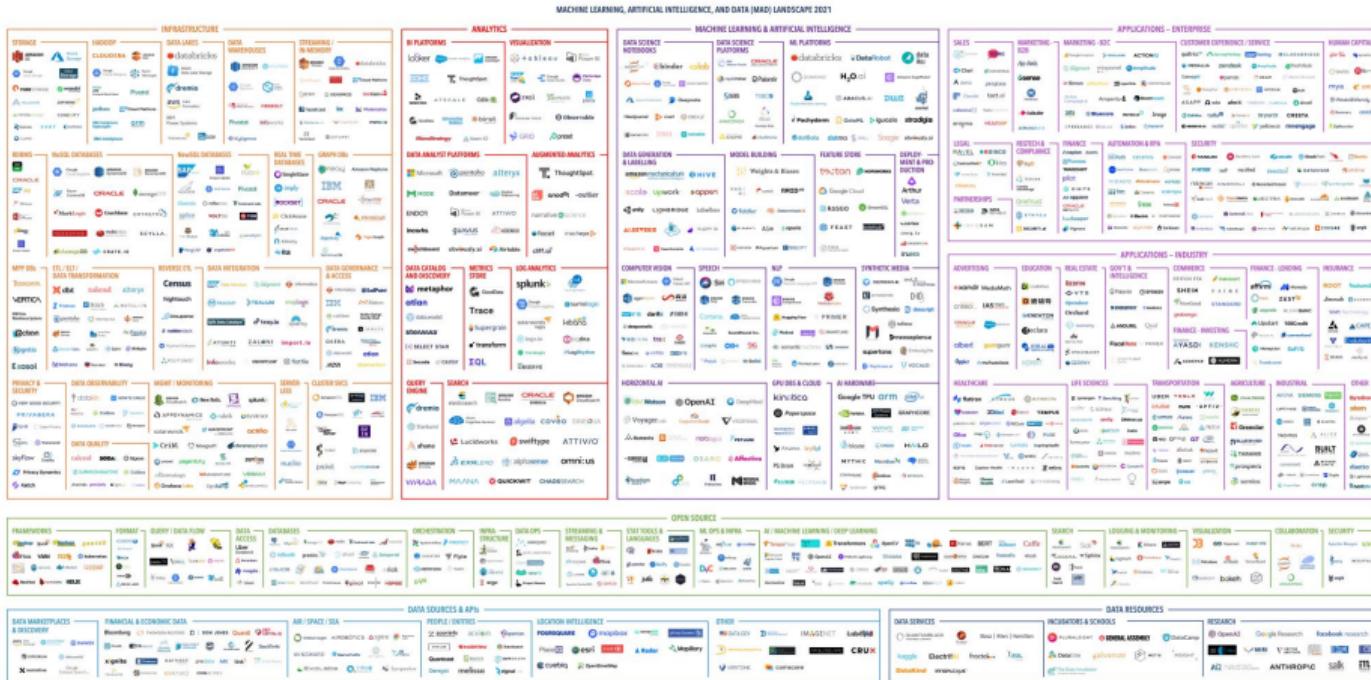
Introdução

Qual o valor que o *Big Data* gera?



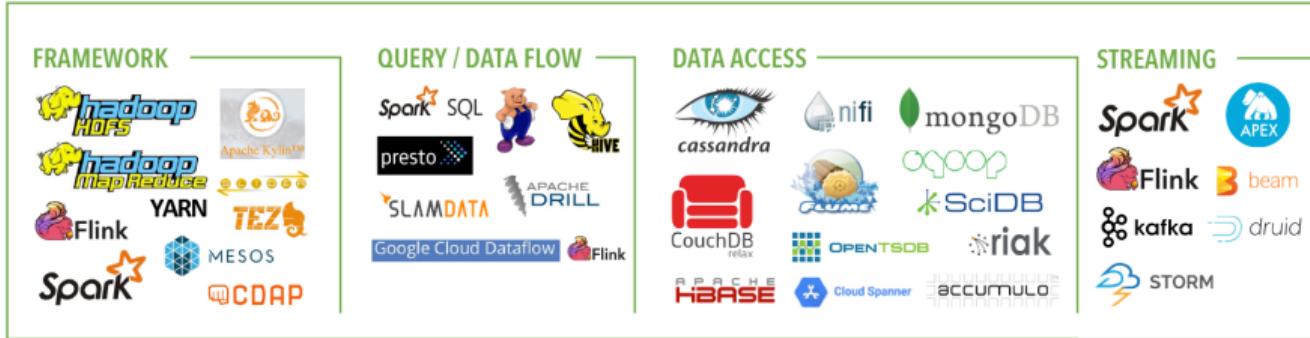
Ecossistema do Big Data

Ecossistema Qual a estrutura que compõe o *Big Data*?



Ecossistema

Qual a estrutura que compõe o *Big Data*?

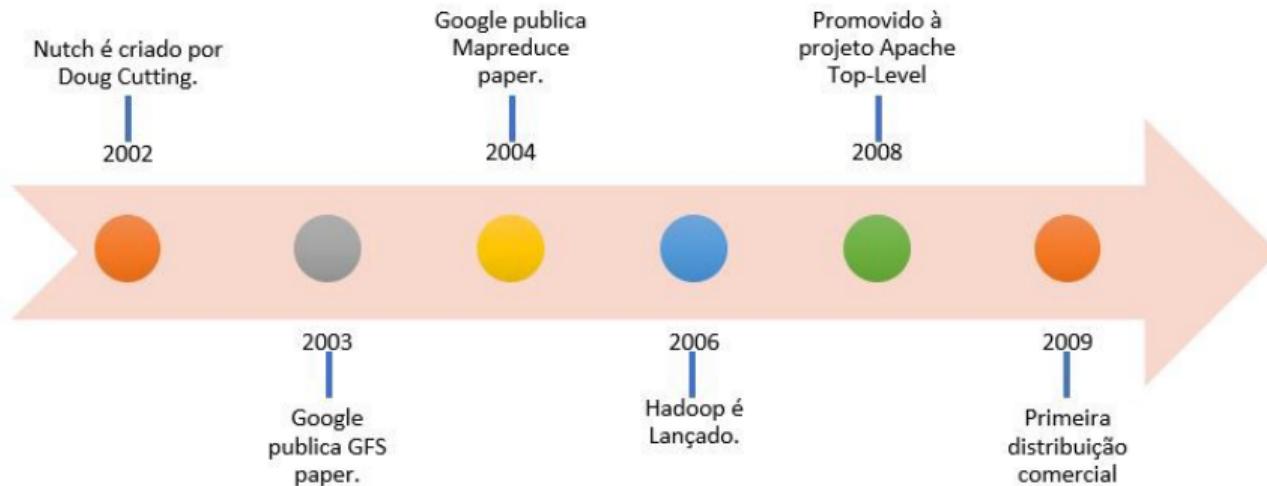


AI / MACHINE LEARNING / DEEP LEARNING



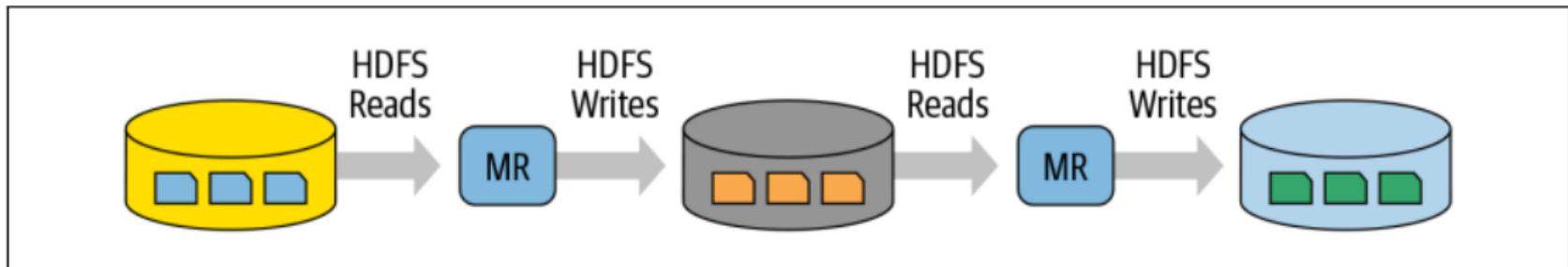
Apache Spark

Primórdios



Primórdios

Hadoop



Apache Spark



- ▶ O Apache Spark é um poderoso *framework* de computação paralela e distribuída, desenvolvido em 2009, no AMPLab da Universidade de Berkeley (EUA), hoje RISELab

Apache Spark



- ▶ O Apache Spark é um poderoso *framework* de computação paralela e distribuída, desenvolvido em 2009, no AMPLab da Universidade de Berkeley (EUA), hoje RISELab
- ▶ **O Spark herdou importantes funcionalidades do Apache Hadoop, como o processamento baseado no paradigma MapReduce**

Apache Spark



- ▶ O Apache Spark é um poderoso *framework* de computação paralela e distribuída, desenvolvido em 2009, no AMPLab da Universidade de Berkeley (EUA), hoje RISELab
- ▶ O Spark herdou importantes funcionalidades do Apache Hadoop, como o processamento baseado no paradigma MapReduce
- ▶ **A principal característica do Spark é a realização do processamento em memória, utilizando RDDs (*Resilient Distributed Datasets*), que são imutáveis e tolerantes a falhas**

Apache Spark

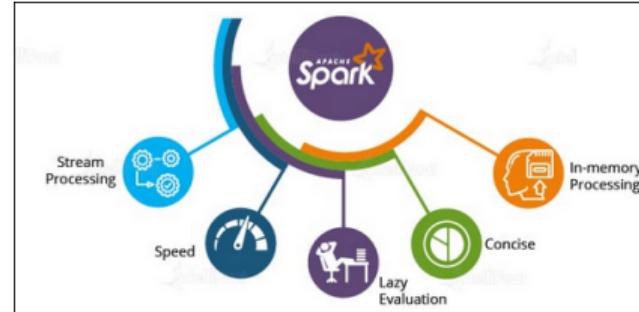


- ▶ O Apache Spark é um poderoso *framework* de computação paralela e distribuída, desenvolvido em 2009, no AMPLab da Universidade de Berkeley (EUA), hoje RISELab
- ▶ O Spark herdou importantes funcionalidades do Apache Hadoop, como o processamento baseado no paradigma MapReduce
- ▶ A principal característica do Spark é a realização do processamento em memória, utilizando RDDs (*Resilient Distributed Datasets*), que são **imutáveis e tolerantes a falhas**
- ▶ **As RDDs fornecem processamento multiestágio, representado como um DAG (grafo acíclico direcionado), avaliado preguiçosamente (*Lazy Evaluation*)**

Apache Spark



- ▶ O Apache Spark é um poderoso *framework* de computação paralela e distribuída, desenvolvido em 2009, no AMPLab da Universidade de Berkeley (EUA), hoje RISELab
- ▶ O Spark herdou importantes funcionalidades do Apache Hadoop, como o processamento baseado no paradigma MapReduce
- ▶ A principal característica do Spark é a realização do processamento em memória, utilizando RDDs (*Resilient Distributed Datasets*), que são **imutáveis** e **tolerantes a falhas**
- ▶ As RDDs fornecem processamento multiestágio, representado como um DAG (grafo acíclico direcionado), avaliado preguiçosamente (*Lazy Evaluation*)



Sistemas Paralelos e Distribuídos

Apache Spark

Qual o contexto?



- ▶ O Apache Spark é um *framework* de processamento paralelo e distribuído. Mas o que isso significa?

Apache Spark

Qual o contexto?



- ▶ O Apache Spark é um *framework* de processamento paralelo e distribuído. Mas o que isso significa?
- ▶ **O desenvolvimento da computação assistiu um explosivo incremento da capacidade computacional, aliado à miniaturização de circuitos, com aumento da densidade de transístores**

Apache Spark

Qual o contexto?



- ▶ O Apache Spark é um *framework* de processamento paralelo e distribuído. Mas o que isso significa?
- ▶ O desenvolvimento da computação assistiu um explosivo incremento da capacidade computacional, aliado à miniaturização de circuitos, com aumento da densidade de transístores
- ▶ **A Lei de Moore apresenta de forma empírica este fenômeno: a cada 18 meses, a quantidade de transistores aplicados em *chips* aumentaria em um fator de 2, mantendo os custos**

Apache Spark

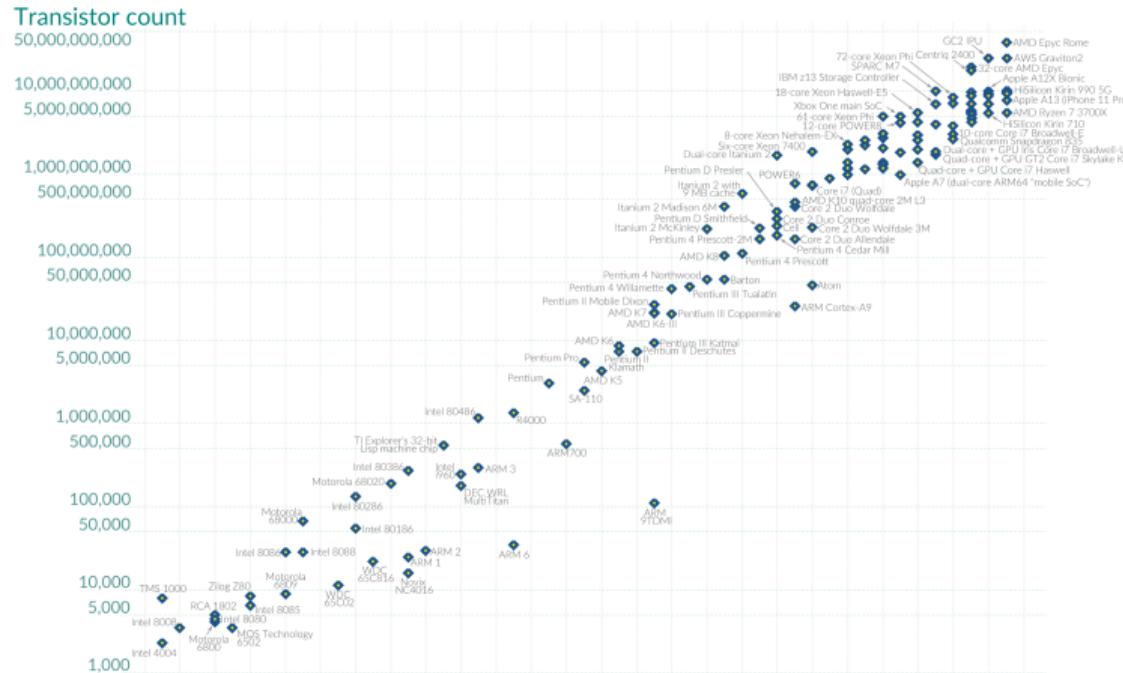
Qual o contexto?



Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World
in Data



Apache Spark

Qual o contexto?



- ▶ A indústria direcionou o incremento do desempenho computacional inserindo mais processadores

Apache Spark

Qual o contexto?



- ▶ A indústria direcionou o incremento do desempenho computacional inserindo mais processadores
- ▶ Assim, o paralelismo de processadores possibilitaria grande impulso na capacidade de processamento das máquinas

Apache Spark

Qual o contexto?



- ▶ A indústria direcionou o incremento do desempenho computacional inserindo mais processadores
- ▶ Assim, o paralelismo de processadores possibilitaria grande impulso na capacidade de processamento das máquinas
- ▶ **Problemas complexos poderiam ser tratados**

Apache Spark

Qual o contexto?



- ▶ Considere um jarro de balas, que representa uma coleção de dados

Apache Spark

Qual o contexto?



- ▶ Considere um jarro de balas, que representa uma coleção de dados
- ▶ **Adotando o paralelismo de dados, os processadores da máquina realizarão alguma tarefa desejada**

Apache Spark

Qual o contexto?



- ▶ Considere um jarro de balas, que representa uma coleção de dados
- ▶ Adotando o paralelismo de dados, os processadores da máquina realizarão alguma tarefa desejada



Compute Node
(Shared Memory)



Apache Spark Qual o contexto?



Compute Node
(Shared Memory)



Apache Spark

Qual o contexto?



Compute Node
(Shared Memory)



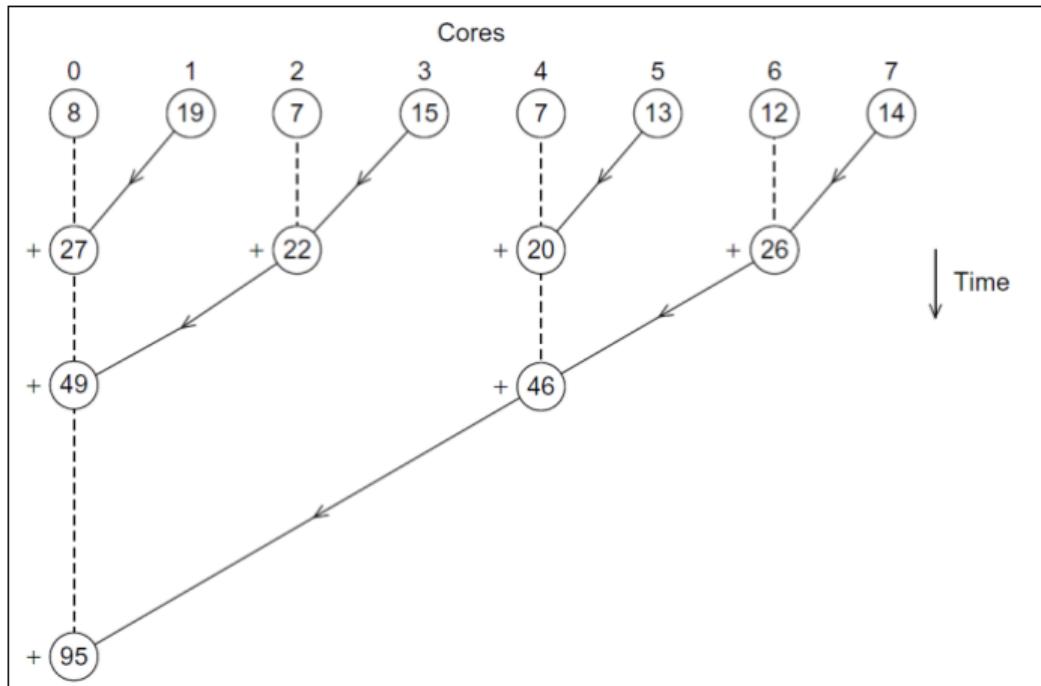
- ▶ Então, aplicamos diferentes processadores para executar a mesma tarefa sobre subconjuntos da coleção original de dados

Apache Spark Qual o contexto?

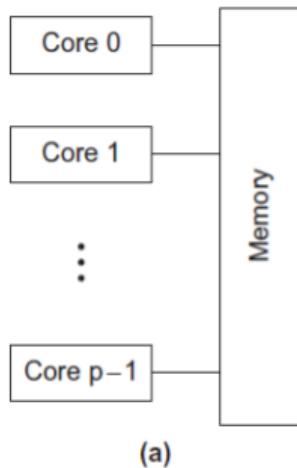


Compute Node
(Shared Memory)

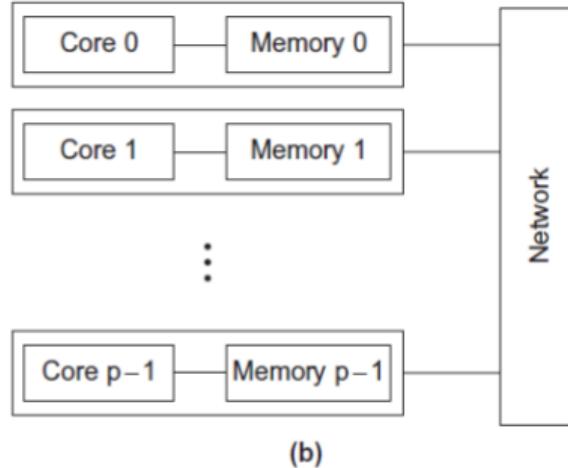
Apache Spark Qual o contexto?



Apache Spark Qual o contexto?



(a)



(b)

Memória compartilhada

Memória distribuída

Apache Spark

Qual o contexto?



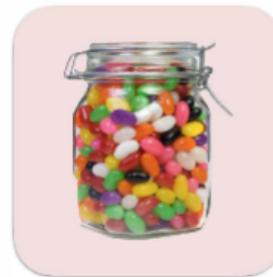
	Memória Compartilhada	Memória Distribuída
Prós	<ul style="list-style-type: none">perspectiva de programação amigávelcompartilhamento de dados é rápido e uniforme devido à proximidade da memória com as CPUs	<ul style="list-style-type: none">memória é escalável com o número de processadorescada processador pode acessar sua própria memória sem <i>overhead</i> para manter a coerência do cachecusto-benefício: pode usar <i>commodities</i>
Contras	<ul style="list-style-type: none">falta de escalabilidade entre memória e CPUsprogramador é responsável pela sincronização do acesso "correto" à memória globaldifícil e caro projetar e produzir máquinas de memória compartilhada com número maior de processadores	<ul style="list-style-type: none">programador é responsável por muitos dos detalhes de comunicação de dados entre processadores.pode ser difícil mapear estruturas de dados existentes (global → distribuída)tempos de acesso não uniforme à memória

Apache Spark

Qual o contexto?



Considere agora um sistema distribuído, com a coleção de dados distribuída em diferentes máquinas

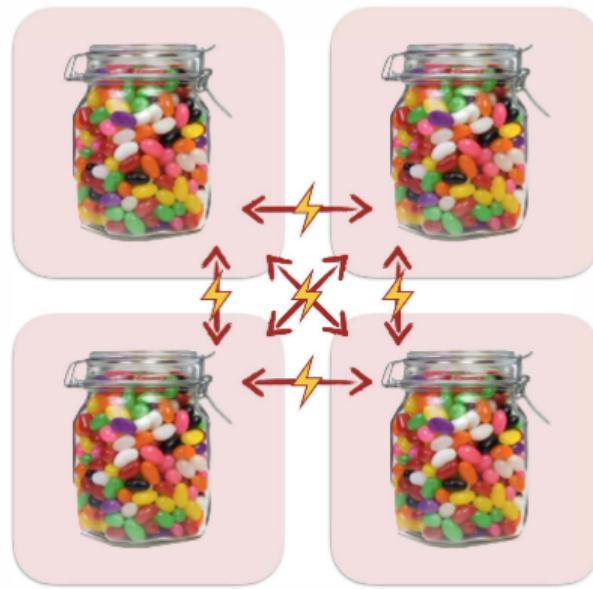


Apache Spark

Qual o contexto?



Estas máquinas comunicam-se utilizando protocolos de rede



Apache Spark

Qual o contexto?



Podemos combinar o princípio de processamento com memória compartilhada, estabelecendo comunicação entre os diferentes nós que compõem a arquitetura

Shared memory:



Distributed:

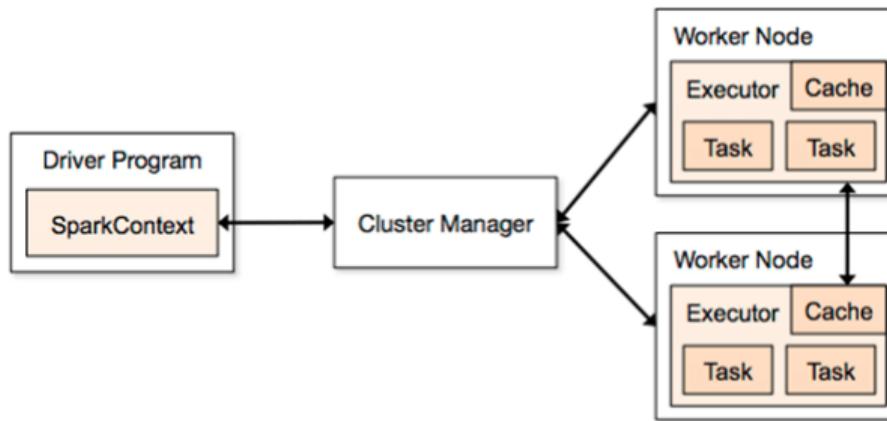


Arquitetura do Apache Spark

Apache Spark Qual o contexto?



A arquitetura do Apache Spark acompanha esta abordagem apresentada. A partir do *Spark Context*, localizado em uma máquina principal, a aplicação é executada nas demais máquinas com auxílio do *cluster manager*

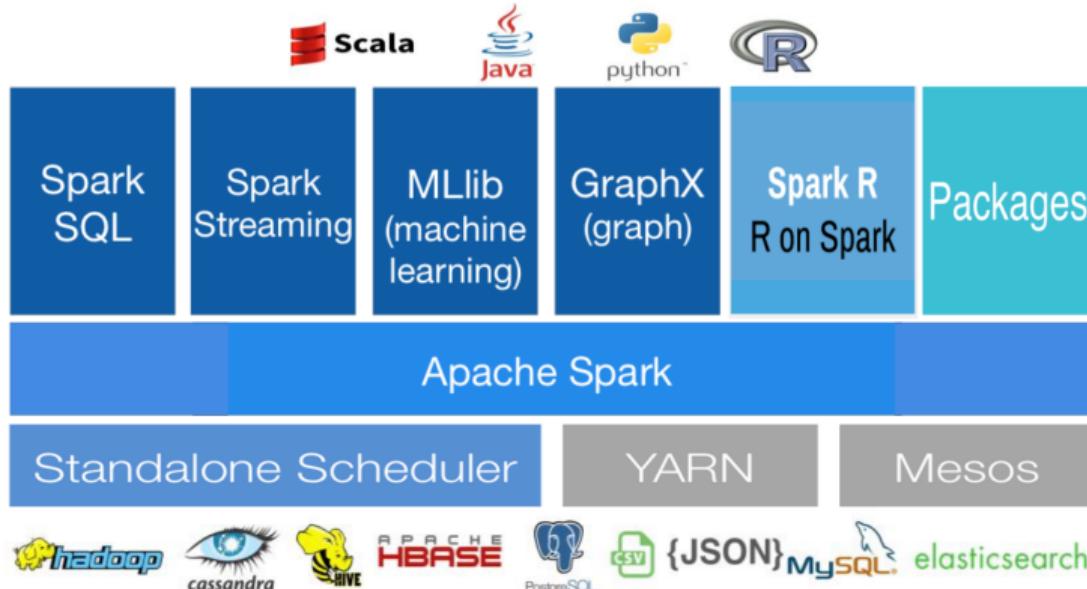


Apache Spark

Mais sobre a arquitetura.



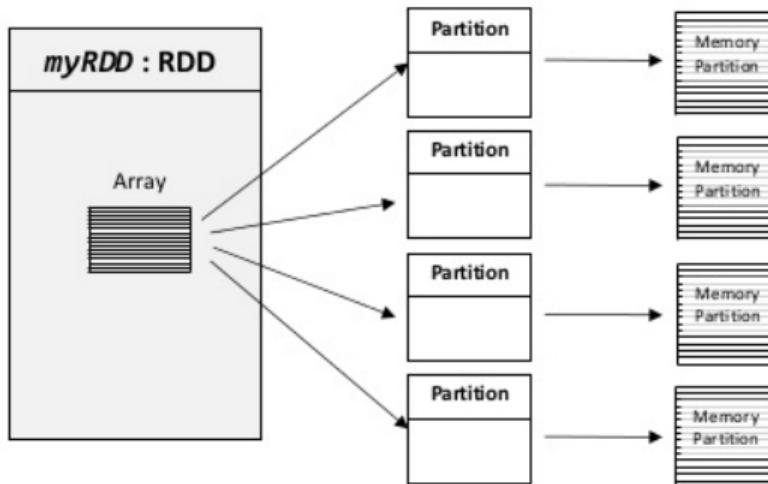
Diferentes módulos integram o *framework* Apache Spark. Cada um estende as funcionalidades e potencialidades do *framework* para um domínio de aplicação.



Apache Spark RDDs



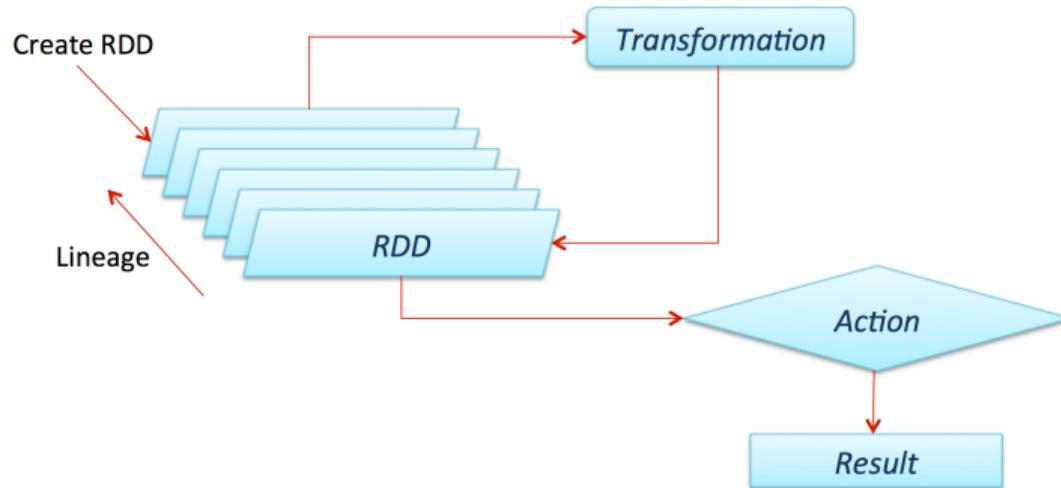
Quando criadas, as RDDs são compartilhadas com os nós que integram o *cluster*. A RDD pode ser criada a partir de coleções de dados ou a partir de fontes de dados como arquivo HDFS, conexões com banco de dados relacionais ou não-relacionais, por exemplo.



Apache Spark RDDs



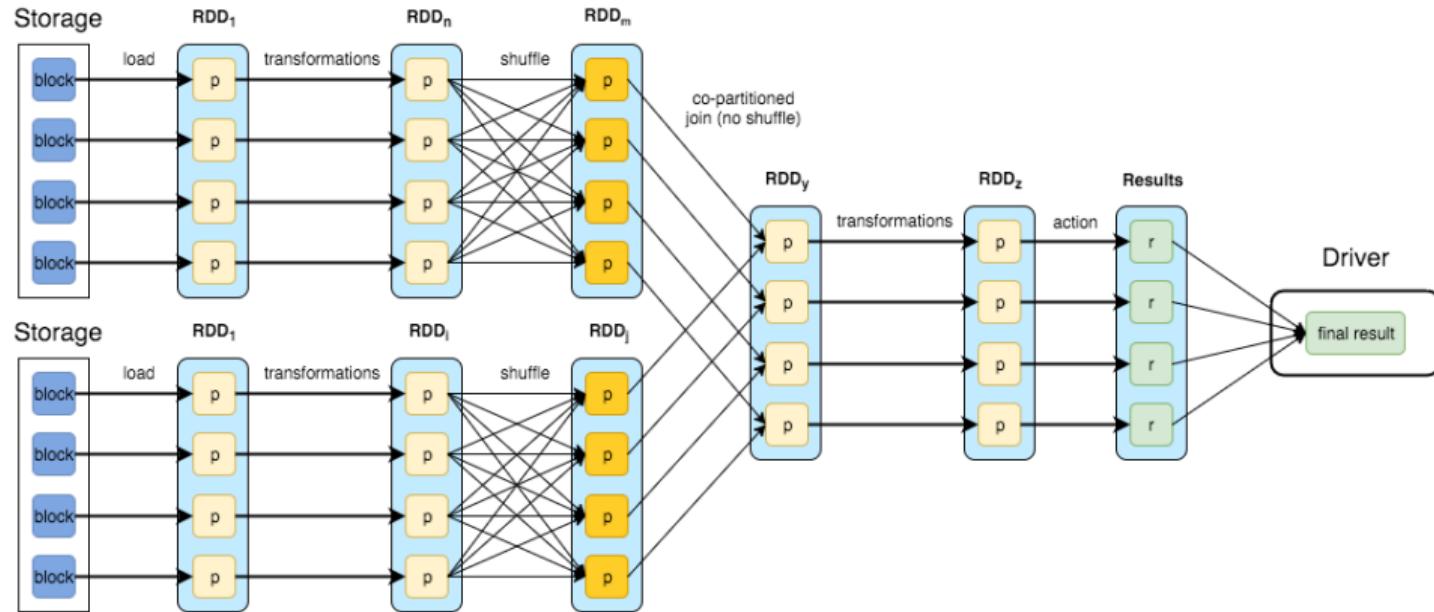
O processamento das RDDs é realizado preguiçosamente (*lazy evaluation*). Sobre as RDDs são aplicadas Transformações e Ações.



Apache Spark RDDs



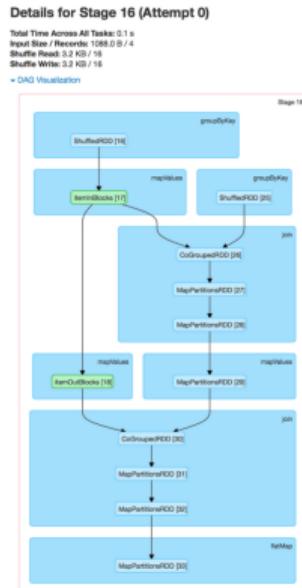
Na Figura abaixo, vemos as RDDs sofrendo uma série de Transformações e Ações. RDDs são imutáveis. Observe que após transformação da RDD_y é criada uma nova RDD_z.



Apache Spark RDDs



As Transformações criam um DAG *Scheduler* que será percorrido quando executada uma Ação.



Transformações

Apache Spark

Algumas Transformações



Transformação	Descrição
<code>map(func)</code>	Retorna uma nova RDD formada pelo processamento de cada elemento pela função <i>func</i>
<code>filter(func)</code>	Retorna uma nova RDD formada pela seleção dos elementos para os quais o valor da função <i>func</i> retorna TRUE
<code>flatMap(func)</code>	Similar à transformação map, retornando uma sequência de itens
<code>mapPartitions(func)</code>	Similar ao map, mas processa separadamente em cada partição da RDD
<code>mapPartitionsWithIndex(func)</code>	Similar à transformação mapPartitions, mas também fornece uma função com um valor inteiro cujo valor identifica a partição
<code>sample(withReplacement, fraction, seed)</code>	Adquire uma fração dos dados, com ou sem reposição, utilizando um número gerado aleatoriamente a partir de uma semente

Apache Spark

Algumas Transformações



Transformação	Descrição
<code>union(otherDataset)</code>	Retorna uma nova RDD que contém a união dos elementos
<code>intersection(otherDataset)</code>	Retorna uma nova RDD que contém a interseção dos elementos
<code>distinct([numPartitions])</code>	Retorna uma nova RDD que não possui valores duplicados
<code>groupByKey([numPartitions])</code>	É uma transformação que realiza a operação de <i>shuffle</i> , retornando uma RDD que possui um par, identificando a chave com seu valor
<code>reduceByKey(func, [numPartitions])</code>	É uma transformação que realiza a operação de redução no valor do par
<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	É uma transformação que retorna uma RDD contendo pares, na qual o valor para cada chave é agregado utilizando as funções de combinação
<code>join(otherDataset, [numPartitions])</code>	Realiza a operação de junção, cujo tipo pode ser especificado

Apache Spark

Algumas Ações



Transformação	Descrição
reduce(<i>func</i>)	Agrega os elementos da RDD utilizando uma função <i>func</i>
collect()	Retorna todos os elementos da RDD
take(n)	Retorna o elemento n da RDD
takeSample(withReplacement, num, [seed])	Retorna um arranjo obtido aleatoriamente com <i>num</i> elementos, a partir de uma semente
foreach(<i>func</i>)	Aplica uma função <i>func</i> sobre cada elemento da RDD
countByKey()	Disponível em RDD com elementos em pares, conta a quantidades de pares existente para cada chave
first()	Retorna o primeiro elemento da RDD

APIs do Apache Spark

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados
- ▶ Data-Frames são abstrações de alto nível sobre as RDDs, o que lhe confere suas características centrais, como: imutabilidade, avaliação preguiçosa (*lazy evaluation*), fácil transmutação Data-Frame/RDD e distribuição

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados
- ▶ Data-Frames são abstrações de alto nível sobre as RDDs, o que lhe confere suas características centrais, como: imutabilidade, avaliação preguiçosa (*lazy evaluation*), fácil transmutação Data-Frame/RDD e distribuição
- ▶ **Data-frames organizam os dados segundo um *schema*, o que facilita a otimização de um plano de execução de queries**

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados
- ▶ Data-Frames são abstrações de alto nível sobre as RDDs, o que lhe confere suas características centrais, como: imutabilidade, avaliação preguiçosa (*lazy evaluation*), fácil transmutação Data-Frame/RDD e distribuição
- ▶ Data-frames organizam os dados segundo um *schema*, o que facilita a otimização de um plano de execução de *queries*
- ▶ **Habilidade de manipular petabytes de dados**

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados
- ▶ Data-Frames são abstrações de alto nível sobre as RDDs, o que lhe confere suas características centrais, como: imutabilidade, avaliação preguiçosa (*lazy evaluation*), fácil transmutação Data-Frame/RDD e distribuição
- ▶ Data-frames organizam os dados segundo um *schema*, o que facilita a otimização de um plano de execução de *queries*
- ▶ Habilidade de manipular petabytes de dados
- ▶ **Tem suporte para uma ampla gama de formatos e fontes**

Data-Frames



- ▶ Data-frames foram desenvolvidos para processar extensa coleção de dados estruturado ou semi-estruturados
- ▶ Data-Frames são abstrações de alto nível sobre as RDDs, o que lhe confere suas características centrais, como: imutabilidade, avaliação preguiçosa (*lazy evaluation*), fácil transmutação Data-Frame/RDD e distribuição
- ▶ Data-frames organizam os dados segundo um *schema*, o que facilita a otimização de um plano de execução de *queries*
- ▶ Habilidade de manipular petabytes de dados
- ▶ Tem suporte para uma ampla gama de formatos e fontes
- ▶ **Está disponível para diferentes linguagens como Python (assemelha-se ao Pandas), R, Scala e Java.**

Apache Spark

APIs do Apache Spark



- ▶ O Apache Spark foi desenvolvido utilizando a linguagem de programação Scala, que opera sobre a JVM (Máquina Virtual Java)
- ▶ Contudo, é possível desenvolver aplicações no Spark utilizando Python, R e Java
- ▶ Neste curso, utilizaremos exemplos em Python (PySpark)

Data-Frames



```
from pyspark.sql import SparkSession

spark = (SparkSession
         .builder
         .appName("Nome_APP")
         .getOrCreate())

df = spark.read.format("csv").
      option("header", "true").
      option("inferSchema", "false").
      option("delimiter",';').
      schema(schema).
      load(path)
```

Data-Frames



```
df.printSchema()
```

```
df.columns
```

```
df.describe().show()
```

```
df.summary().show()
```

```
df.take(n) # n=1,2,...
```

```
df.collect()
```

Data-Frames



```
df.head(n) # n=1, 2 ,...
```

```
dict1=df.head(n)[ i ].asDict() # n=1, 2; i = 0, 1, (n-1)
```

```
df.count()
```

```
df.select('Column').distinct().show()
```

```
df.select('Column').distinct().count()
```

```
df.withColumn('Range', df['High'] - df['Low']).limit(5).  
select(['High', 'Low', 'Range']).show()
```

Data-Frames



```
df.filter("Close < 500").show(5)

# SQL-like
df.filter("Close < 500 AND Open >500").
select(['Date','Open','Close']).show()

# pythonic
df.filter((df['Close']<500) & (df['Open']>500)).
select(['Date','Open','Close']).show(5)
```

Data-Frames



```
df4 = df.select('High', 'Low').limit(10)

df4.sort("High").show()

df4.sort("High", ascending=False).show()

from pyspark.sql.functions import desc

df4.sort(desc("High")).show()

df2.orderBy(["Company", "Sales"], ascending=[0,1]).show()
```

Data-Frames



```
# arquivo disponivel para download  
df.groupby('Company').mean().show()  
  
df.groupby('Company').max().show()  
  
df.groupby('Company').min().show()  
  
df.groupby('Company').count().show()  
  
df.agg({'Sales':'sum'}).show()  
  
df.agg({'Sales':'max'}).show()
```

Data-Frames



```
# arquivo dispon vel para download  
  
group_data = df.groupBy('Company')  
  
group_data.agg({'Sales': 'max'}).show()  
  
from pyspark.sql.functions import stddev, countDistinct, count, avg  
  
df.agg({'Sales': 'stddev'}).show()  
  
df.select(stddev('Sales')).show()  
  
df.select(count('Sales')).show()  
  
df.select(countDistinct('Sales')).show()
```

Conclusão

Conclusão



OBRIGADO!