

Desenvolvimento de Soluções Utilizando Spark

Capítulo 1. Introdução ao Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 1.1. A Era do Big Data/O que é o Spark?

Prof. Pedro Calais



Bem-vindo ao Módulo 2 do Bootcamp Cientista de Dados!

Agenda

A Era do Big Data

O que é o Apache Spark?

Um pouco sobre mim

- Sou Pedro Calais, mineiro de Belo Horizonte;
- Mestre e doutor em ciência da computação (UFMG);
- Me divido entre ciência e engenharia:
 - ciência de dados, aprendizado de máquina;
 - construir sistemas com código simples, claro e flexível.
- <http://www.dcc.ufmg.br/~pcalais>

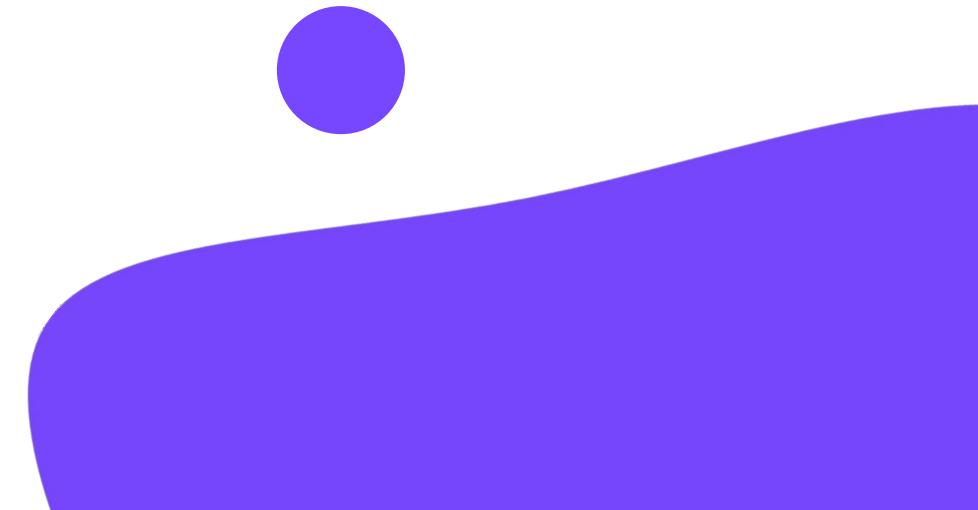
O problema

- Volume de dados está crescendo a uma taxa maior que a capacidade de processamento;
- Capacidade de processamento dos computadores está crescendo a uma velocidade mais lenta.

<https://www.scientificamerican.com/article/computers-are-becoming-fa/>

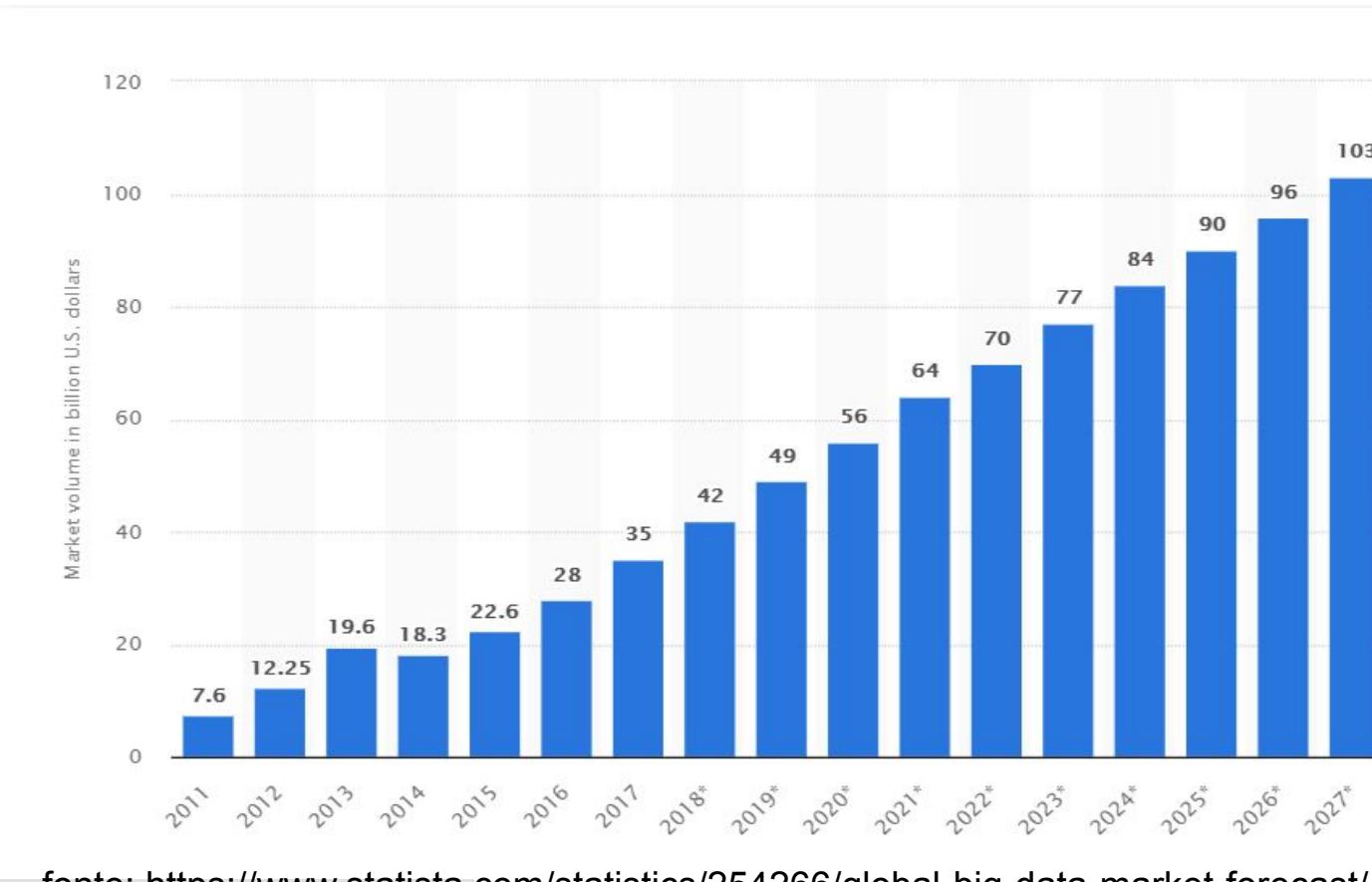
TECHNOLOGY

Computers are becoming faster and faster, but their speed is still limited by the physical restrictions of an electron moving through matter. What technologies are emerging to break through this speed barrier?



Tamanho do mercado de big data em 2027: 100+ bilhões de dólares

IGTI



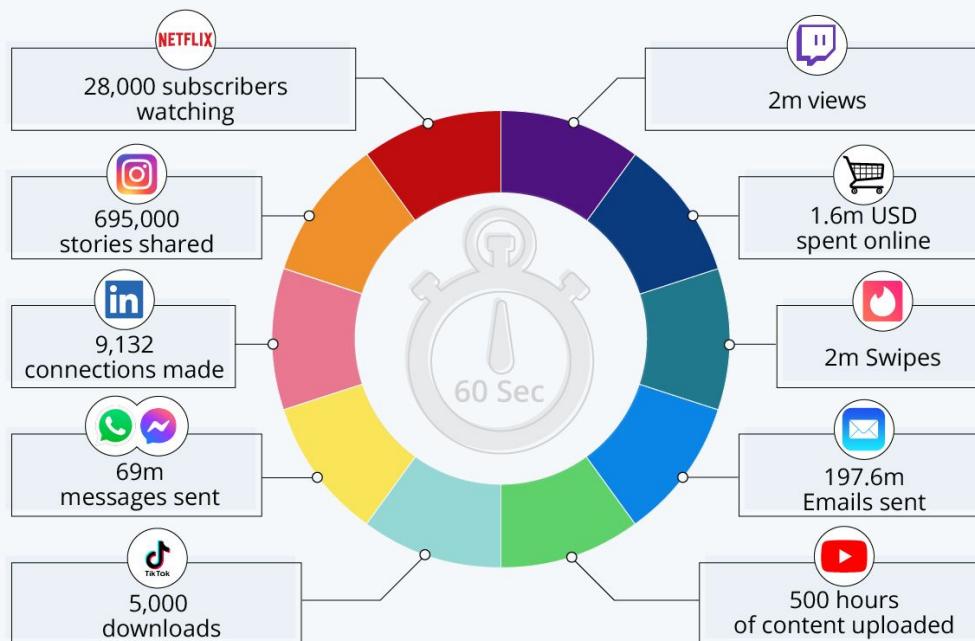
fonte: <https://www.statista.com/statistics/254266/global-big-data-market-forecast/>

Big, very big data!

IGTi

A Minute on the Internet in 2021

Estimated amount of data created
on the internet in one minute



Source: Lori Lewis via AllAccess



statista

~460 exabytes de dados serão gerados por dia em 2025

(1 exabyte = 1 bilhão de GB! (fonte:
Racounter)

- Organizações usam dados para:
 - Tomar melhores decisões;
 - Reduzir custos;
 - Melhorar serviços.
- Necessidade de ferramentas que extraem valor dos dados segue crescendo.

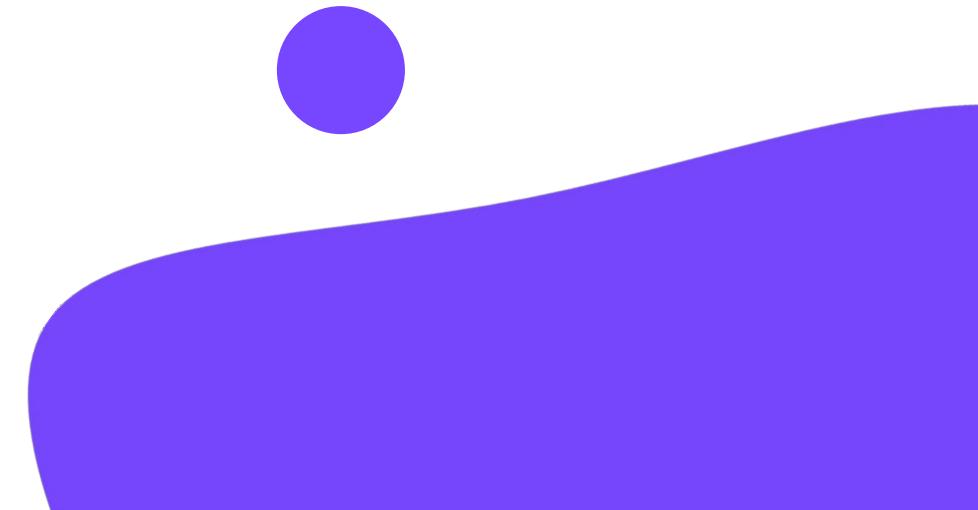
O problema

- Volume de dados está crescendo a uma taxa maior que a capacidade de processamento;
- Capacidade de processamento dos computadores está crescendo a uma velocidade mais lenta.

<https://www.scientificamerican.com/article/computers-are-becoming-fa/>

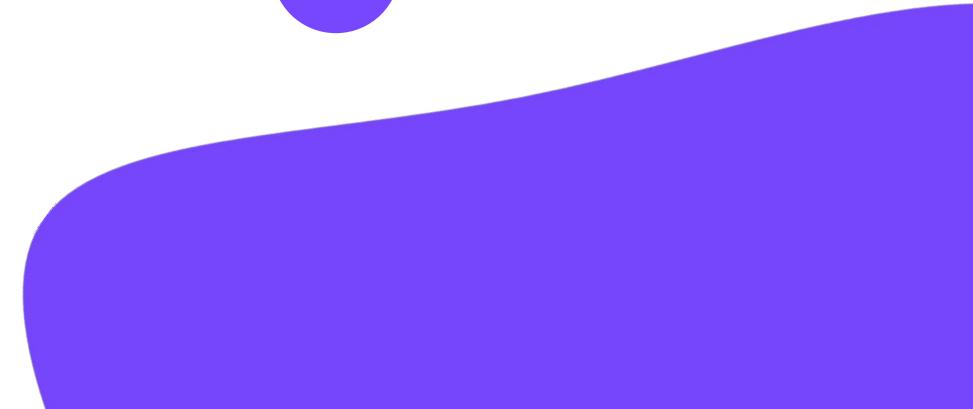
TECHNOLOGY

Computers are becoming faster and faster, but their speed is still limited by the physical restrictions of an electron moving through matter. What technologies are emerging to break through this speed barrier?



O problema

- Dados não cabem na memória de um único computador;
- Dados são gerados em uma taxa muito alta, e queremos interpretá-los e reagir a eles em tempo real;
- Sistemas tradicionais como banco de dados relacionais não escalam para o tamanho “Google”.

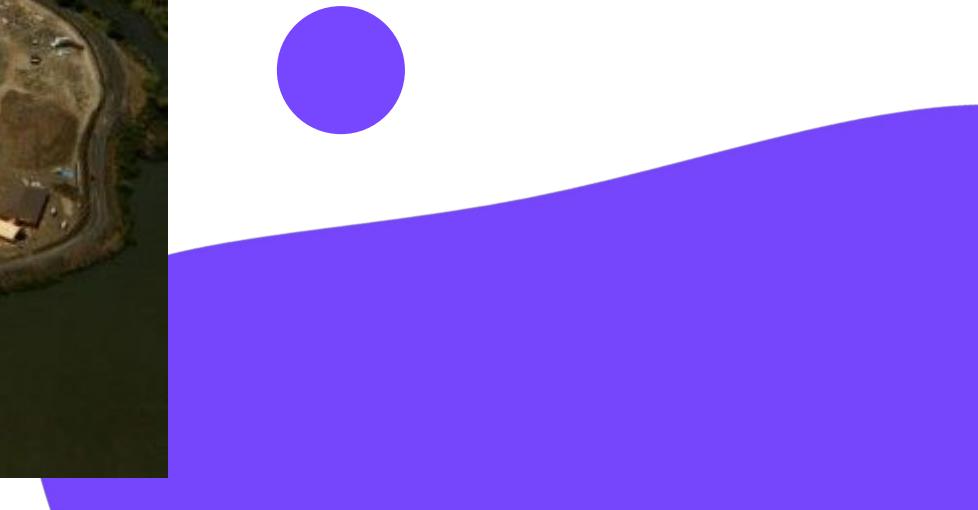


A solução: computação distribuída

IGTI

- Dividimos um conjunto de dados grandes em conjuntos menores;
- Podemos processar os dados de forma distribuída, em múltiplas máquinas;
- Google: 2,5 milhões de servidores.

<https://www.datacenterknowledge.com/archives/2017/03/16/google-data-center-faq>



A solução: computação distribuída



- Grupo de computadores que trabalham em conjunto como se fosse um único computador para o usuário;
- Vasto campo de estudo na ciência da computação;
- Modelos de computação distribuída são complexos de se implementar!
 - Cada máquina pode falhar;
 - Necessidade de coordenar as máquinas entre si;
 - Difíceis de manter, implantar, depurar.

Ferramentas de computação distribuída

IGTI



O que é o Apache Spark?



O que é o Apache Spark?

- Lightning-Fast Unified Analytics for Big Data and Machine Learning
- Lightning-fast cluster computing technology
- Big data platform
- Data processing framework
- Unified engine for large-scale data analytics
- Open-source unified analytics engine for large-scale data processing
- Open-source, distributed processing system used for big data workloads
- General-purpose distributed data processing engine

O que é o Apache Spark?

- Lightning-Fast Unified Analytics for **Big Data** and Machine Learning
- Lightning-fast cluster computing technology
- **Big data** platform
- **Data** processing framework
- Unified engine for **large-scale data** analytics
- Open-source unified analytics engine for **large-scale data** processing
- Open-source, distributed processing system used for **big data workloads**
- General-purpose distributed **data** processing engine

O que é o Apache Spark?

- Lightning-Fast Unified Analytics for Big Data and Machine Learning
- Lightning-fast cluster computing technology
- Big data platform
- Data processing framework
- Unified engine for large-scale data analytics
- Open-source unified analytics engine for large-scale data processing
- Open-source, distributed processing system used for big data workloads
- General-purpose distributed data processing engine

O que é o Apache Spark?

- Lightning-Fast **Unified** Analytics for Big Data and Machine Learning
- Lightning-fast cluster computing technology
- Big data platform
- Data processing framework
- **Unified** engine for large-scale data analytics
- Open-source **unified** analytics engine for large-scale data processing
- Open-source, distributed processing system used for big data workloads
- **General-purpose** distributed data processing engine

História do Apache Spark



Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

Abstract

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. This paper focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. We propose a new framework called Spark that supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time.

História do Apache Spark

IGTI

- Nasceu na Universidade da Califórnia, Berkeley, em 2009.
- Em 2013, foi doado à Fundação Apache;
- Em 2013, os criadores do Spark fundaram a Databricks.



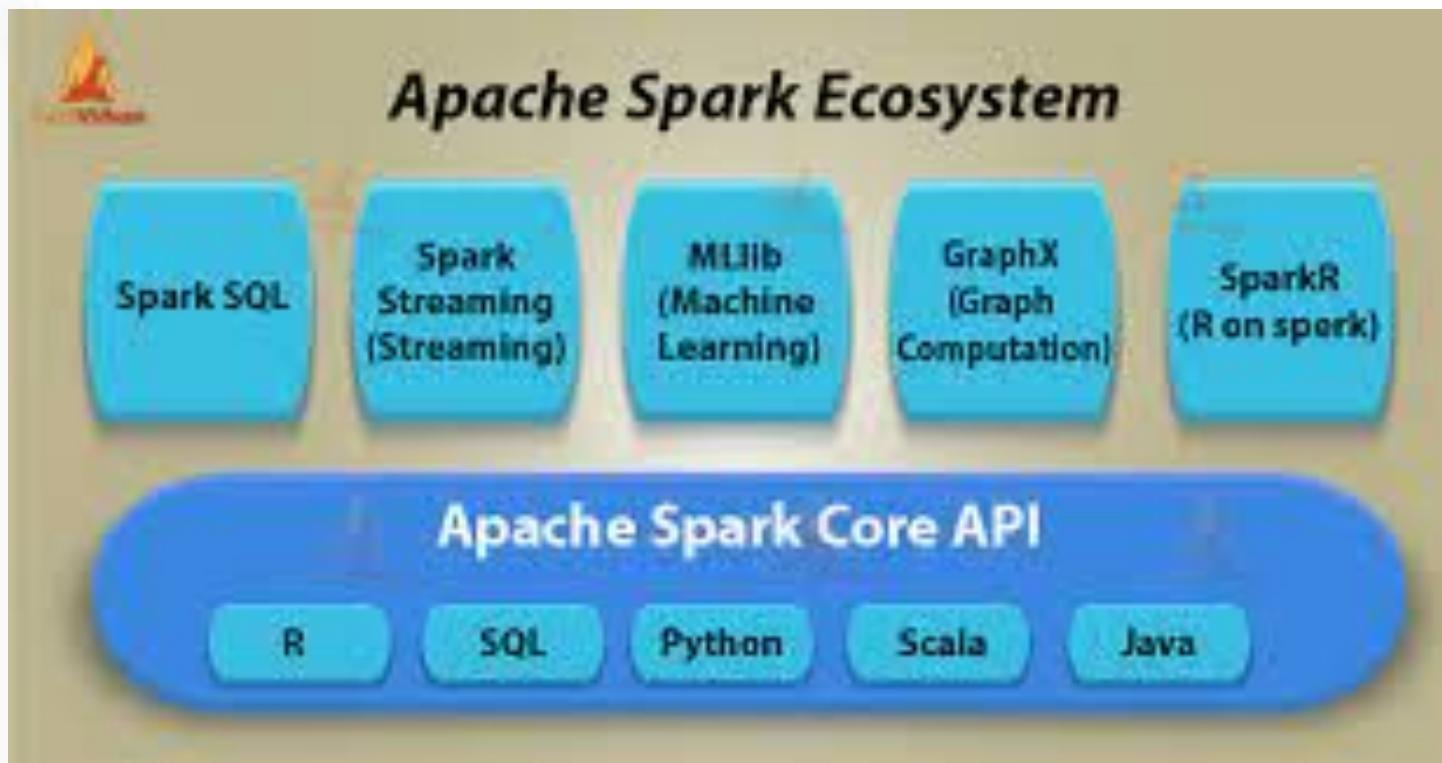
github.com/apache/spark



- É um dos principais projetos da Fundação Apache;
- Construído em Scala;
- Mais de 1.400 contribuidores de 100+ empresas;
- Versão atual: 3.1.1.

A screenshot of the GitHub repository page for "apache / spark". The page has a dark theme. At the top, it shows the repository name "apache / spark" and the status "Public". Below the header, there are navigation links for "Code", "Pull requests 214", "Actions", "Projects", and "Security". A dropdown menu for the "master" branch is open. The stats below the dropdown show "22 branches" and "163 tags". A recent commit by "ueshin" and "HyukjinKwon" is listed, with the commit message "[SPARK-36907][PYTHON] Fix DataFrameGroupBy.a...". The footer of the page includes links for "github" and "SPARK-36883" with the note "[UNRELEASED] Upgrade R version".

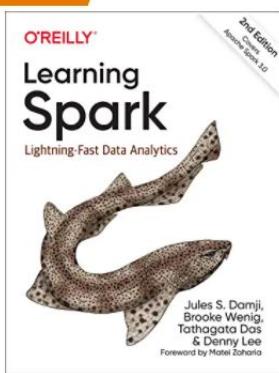
O Spark é um ecossistema



Popularidade

1-48 de 204 resultados para **""apache spark""**

Mais vendido

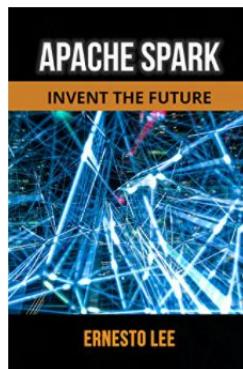


Learning Spark: Lightning-Fast Data Analytics
Edição Inglês
por Jules S Damji, Brooke Weinig, e outros.

★★★★★ 60

Capa Comum

R\$ 375⁹⁰



APACHE SPARK: INVENT THE FUTURE (English Edition)
Edição Inglês
por ERNESTO LEE

Kindle

R\$ 0⁰⁰ kindleunlimited

Gratuito com assinatura ilimitada do Kindle [Saiba mais](#)



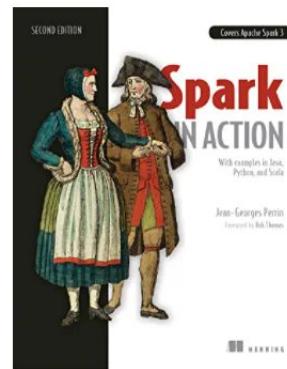
High Performance Spark: Best Practices for Scaling and...
Edição Inglês
por Holden Karau & Rachel Warren

★★★★★ 42

Capa Comum

R\$ 180⁶⁹

em até 6x de R\$ 30,14 sem juros



Spark in Action: Covers Apache Spark 3 with Examples in Java, Python, and Scala
Edição Inglês
por Jean-Georges Perrin

★★★★★ 17

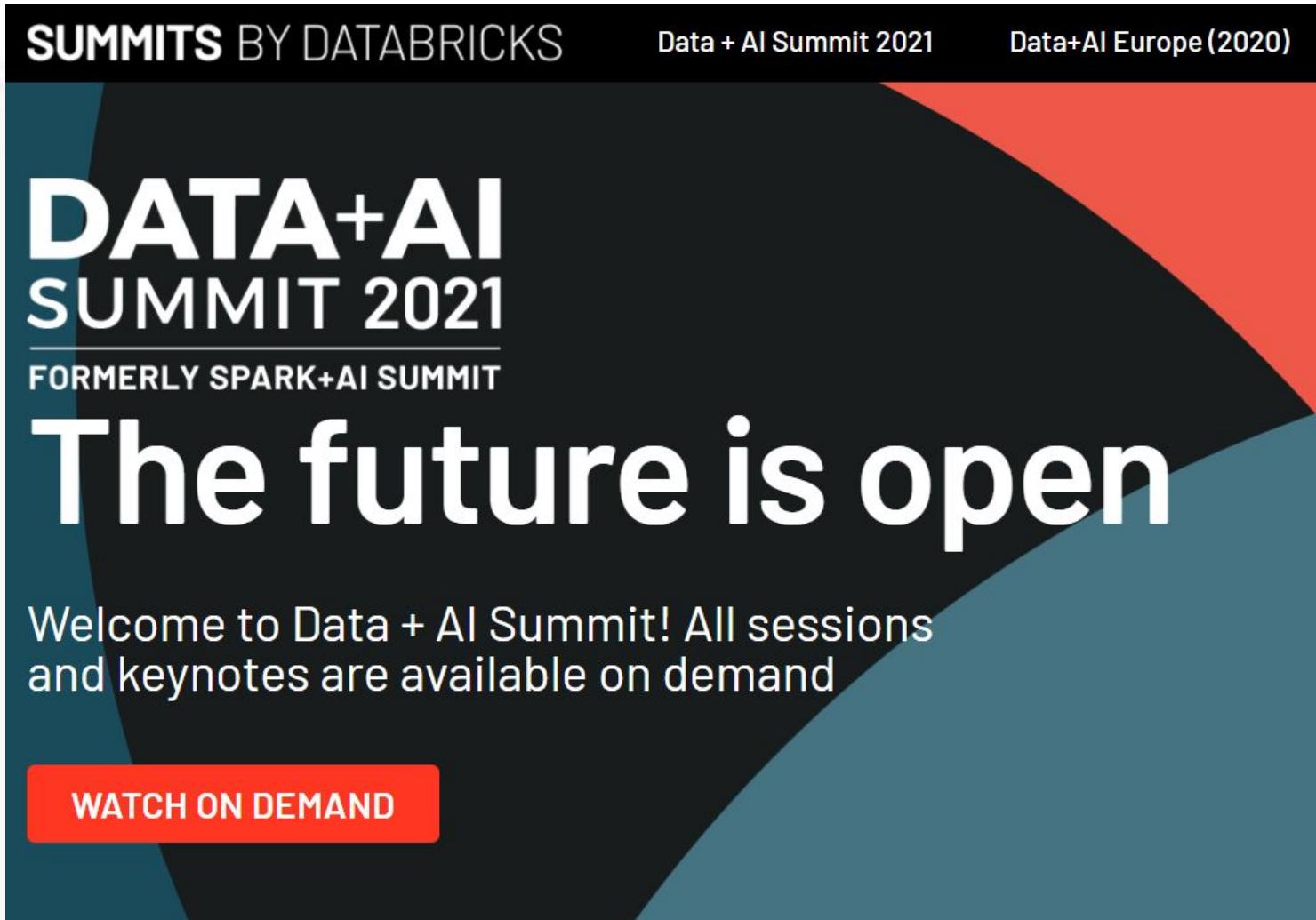
Capa Comum

R\$ 439¹⁸

em até 10x de R\$ 43,99 sem juros

Popularidade

IGTI



The banner for the Data + AI Summit 2021 features a dark background with abstract geometric shapes in teal, black, and orange. At the top left, it says "SUMMITS BY DATABRICKS". In the center, it displays "Data + AI Summit 2021" and "Data+AI Europe (2020)". Below this, the main title "DATA+AI SUMMIT 2021" is prominently shown in large white letters, with "FORMERLY SPARK+AI SUMMIT" underneath. The tagline "The future is open" is written in large white font across the middle. At the bottom left, there is a red button with the text "WATCH ON DEMAND".

SUMMITS BY DATABRICKS

Data + AI Summit 2021 Data+AI Europe (2020)

**DATA+AI
SUMMIT 2021**

FORMERLY SPARK+AI SUMMIT

The future is open

Welcome to Data + AI Summit! All sessions and keynotes are available on demand

WATCH ON DEMAND

Popularidade

Apache Spark

609,225 members | 561 groups

Find out what's happening in Apache Spark Meetup groups around the world and start meeting up with the ones near you.

Related topics: [Big Data](#) · [Hadoop](#) · [Machine Learning](#) · [Apache Kafka](#) · [Big Data Analytics](#) · [Data Science](#) · [Data Analytics](#) · [NoSQL](#) · [Open Source](#) · [MapReduce](#)



- <https://www.meetup.com/'apache-spark/>
- + de 600K membros!

Recapitulando

A Era do Big Data

O que é o Apache Spark?

Na próxima aula

Vantagens e desvantagens do Apache Spark

Por que aprender Apache Spark?



Suporte a várias linguagens



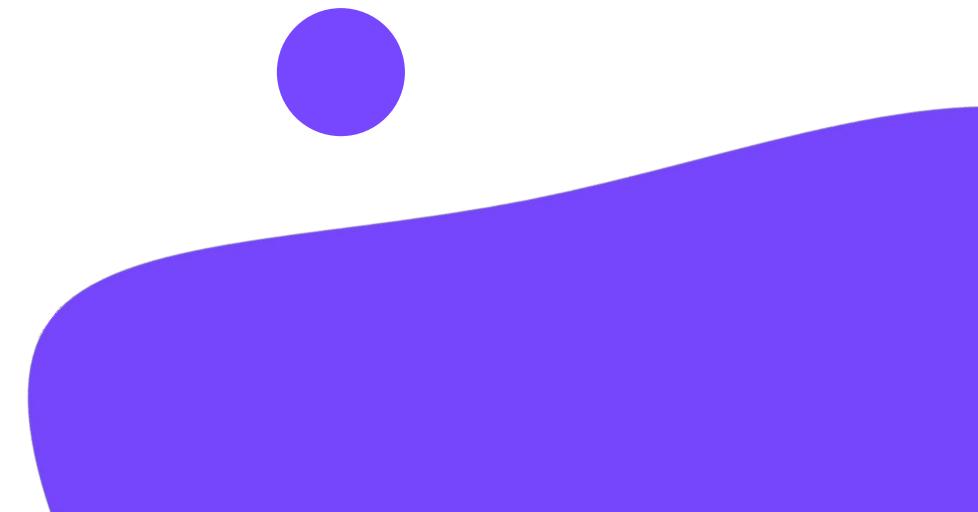
API simples

- Estruturas de dados: RDD, Dataset, Dataframe
- Operações: transformações e ações

```
> lines = sc.textFile("hamlet.txt")
> counts = lines.flatMap(lambda line: line.split(" "))
    .map(lambda word => (word, 1))
    .reduceByKey(lambda x, y: x + y)
```

Desempenho

- Desenhado para processamento em memória.



Ordenando petabytes!



<https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html>

Apache Spark the Fastest Open Source Engine for Sorting a Petabyte



by Reynold Xin

Posted in ENGINEERING BLOG | October 10, 2014



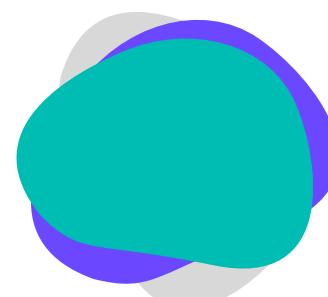
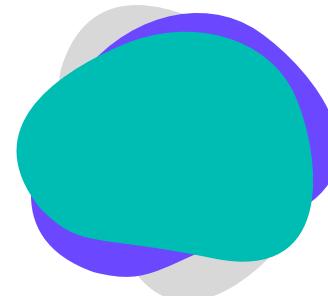
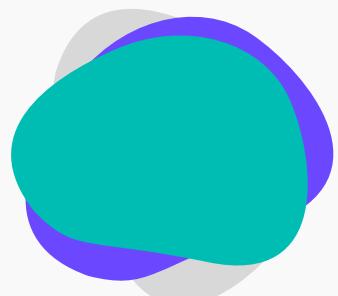
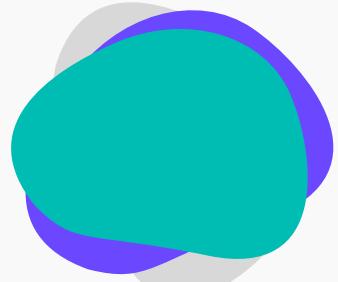
Ordenando petabytes!

<https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html>

		Spark 100 TB *	Spark 1PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

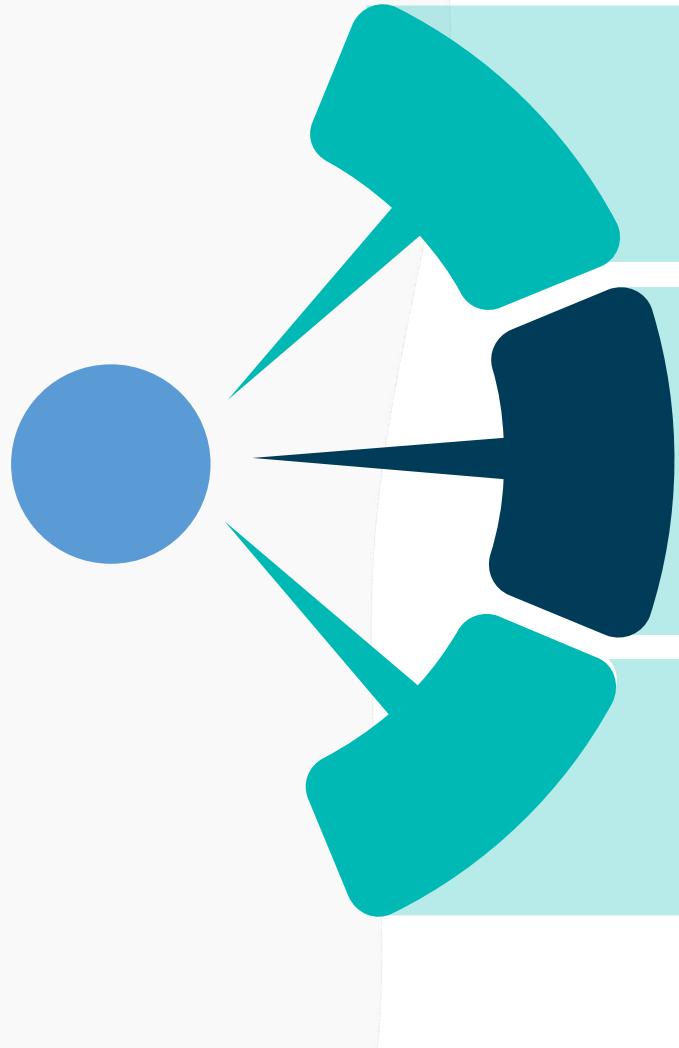


IGTI



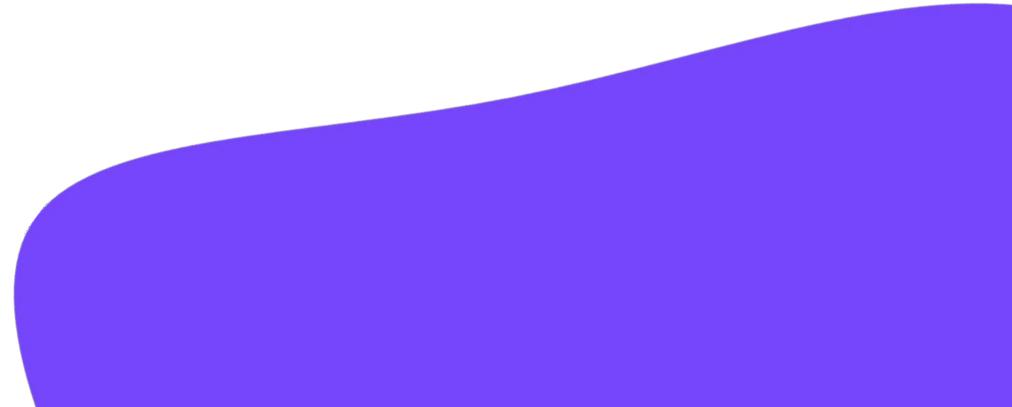


iG Tí





iGTi



01.

03.

02.

04.

Desenvolvimento de Soluções Utilizando Spark

Capítulo 1. Introdução ao Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 1.2. Vantagens e desvantagens do Spark

Prof. Pedro Calais

Recapitulando

A Era do Big Data

O que é o Apache Spark?

Por que aprender Apache Spark?



Desempenho

- Arquitetura alinhada com os avanços da indústria de hardware
 - servidores baratos, com muita memória e várias cores
- Desenhado para processamento em memória.
 - pouco I/O



Ordenando petabytes com Spark



<https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html>

Apache Spark the Fastest Open Source Engine for Sorting a Petabyte



by Reynold Xin

Posted in ENGINEERING BLOG | October 10, 2014



Ordenando petabytes com Spark



<https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html>

		Spark 100 TB *	Spark 1PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2(i2.8xlarge)	EC2(i2.8xlarge)

API simples

- Estruturas de dados: RDD, Dataset, Dataframe
- Operações: transformações e ações

```
val textFile = sparkSession.sparkContext.textFile("hdfs://tmp/words")
val counts = textFile.flatMap(line => line.split(" "))
              .map(word => (word, 1))
              .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://tmp/words_agg")
```

Suporte a várias linguagens



- *bindings* para C#, F#, C#, Julia, Groovy, Clojure, Kotlin
(<https://spark.apache.org/third-party-projects.html>)

Plataforma unificada para *analytics*



componentes para diferentes cargas de trabalho

- dados estruturados: Spark SQL (Capítulo 4)
- dados em grafos: Spark GraphX (Capítulo 5)
- dados em *streaming*: Spark Streaming (Capítulo 6)
- dados para aprendizado de máquina: Spark MLLib (Capítulo 7)

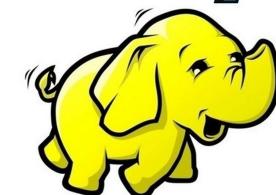
Plataforma unificada para *analytics*

IGTI

- Apache Spark substitui:



hadoop



Quando evitar o Apache Spark?

IGTI

busca em dados textuais:

```
1 sqlContext.sql("select * from myTable where name like '%abc%' ")
```



elasticsearch

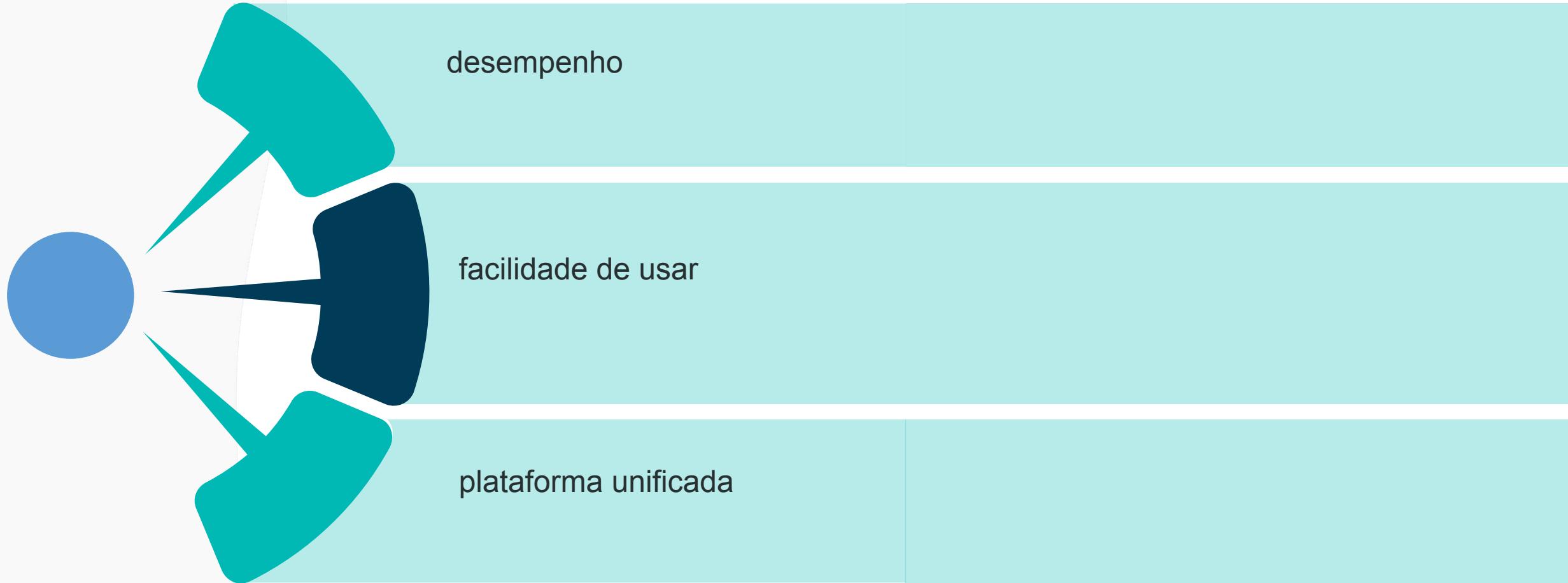
Quando evitar o Apache Spark?

IGTI

- o volume de dados a serem processados não for grande
- poucos recursos de poder computacional e memória



O que faz o Spark especial?



Na próxima aula

Estudos de caso reais com Apache Spark

Desenvolvimento de Soluções Utilizando Spark

Capítulo 1. Introdução ao Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 1.3. Estudos de Caso Reais

Prof. Pedro Calais

Quem usa o Apache Spark?

- cientistas de dados;
- engenheiros de dados;
- engenheiros de aprendizado de máquina.

Quantidade massiva de dados!

Cientistas de dados

- combinação de matemática, estatística, ciência da computação, programação;
- contar estórias por meio dos dados;
- limpar os dados;
- descobrir padrões;
- construir modelos para previsão e recomendação.

Por que o Spark é útil?

- Spark MLLib
- Spark SQL (exploração interativa dos dados)

Engenheiros de dados

- implantar dos modelos em produção;
 - construir e transformar dados reais: sujos e crus;
 - conectar com o pipeline de dados: aplicações web, Apache Kafka;
 - transformação fim-a-fim dos dados: ler de várias fontes e armazenar; os dados na nuvem, em bancos SQL e NoSQL.
-
- Por que o Spark é útil?
 - forma simples de paralelizar a computação;
 - esconde a complexidade de distribuir os dados e tolerar falhas;
 - provê APIs para ler e combinar dados de múltiplas fontes;

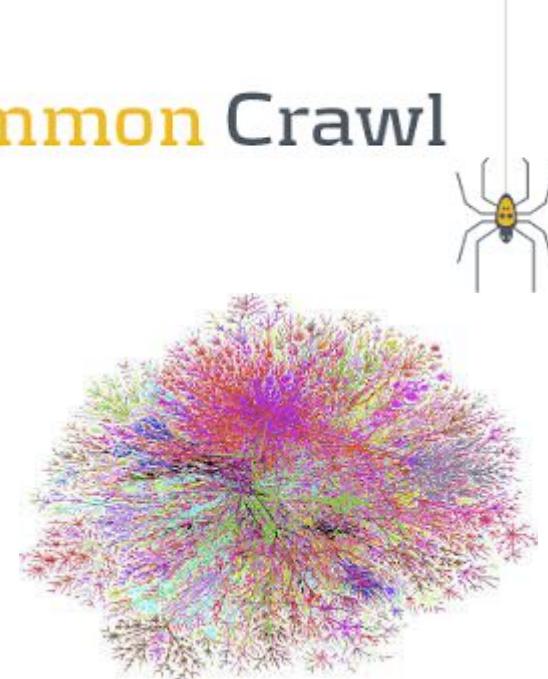
Casos de uso típicos

- Processar em paralelo grandes conjuntos de dados;
- Explorar e visualizar conjuntos de dados de forma interativa;
- Construir, treinar e avaliar modelos de aprendizado de máquina;
- Implementar pipelines de dados fim-a-fim;
- Analisar redes sociais e grafos.

Estudo de caso #1

IGTI

Common Crawl



WORLDSENSE
HYPER YOUR CONTENT

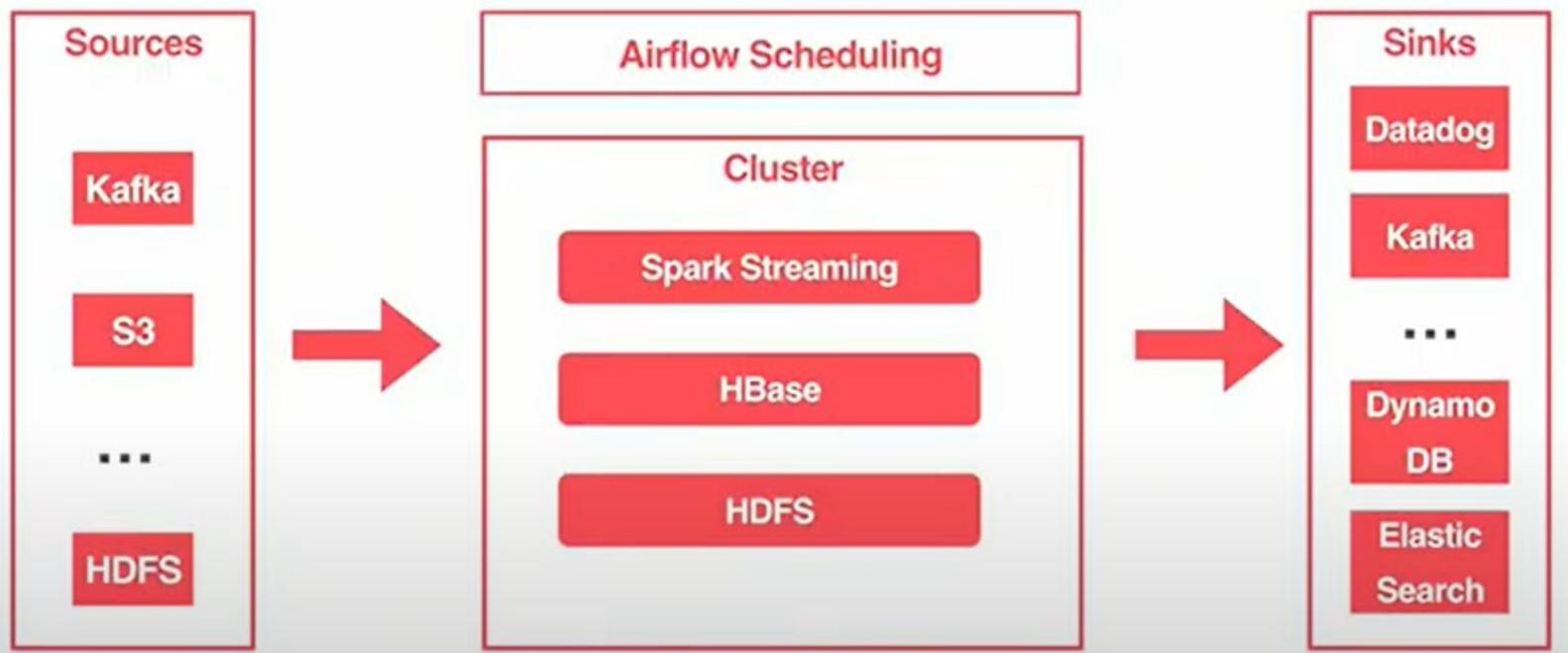
- ~2 bilhões de documentos Web;
- ~ 10 bilhões de links;
- 40 TB de dados.

Estudo de caso #2

- Pipeline de dados complexo;
- Várias fontes e destinos de dados distintas;
- spark em batch e stream;
- <https://databricks.com/session/building-data-product-based-on-apache-spark-at-airbnb>

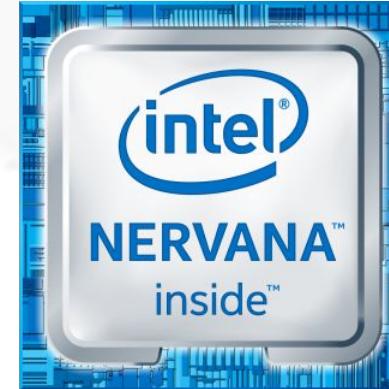


IGTI

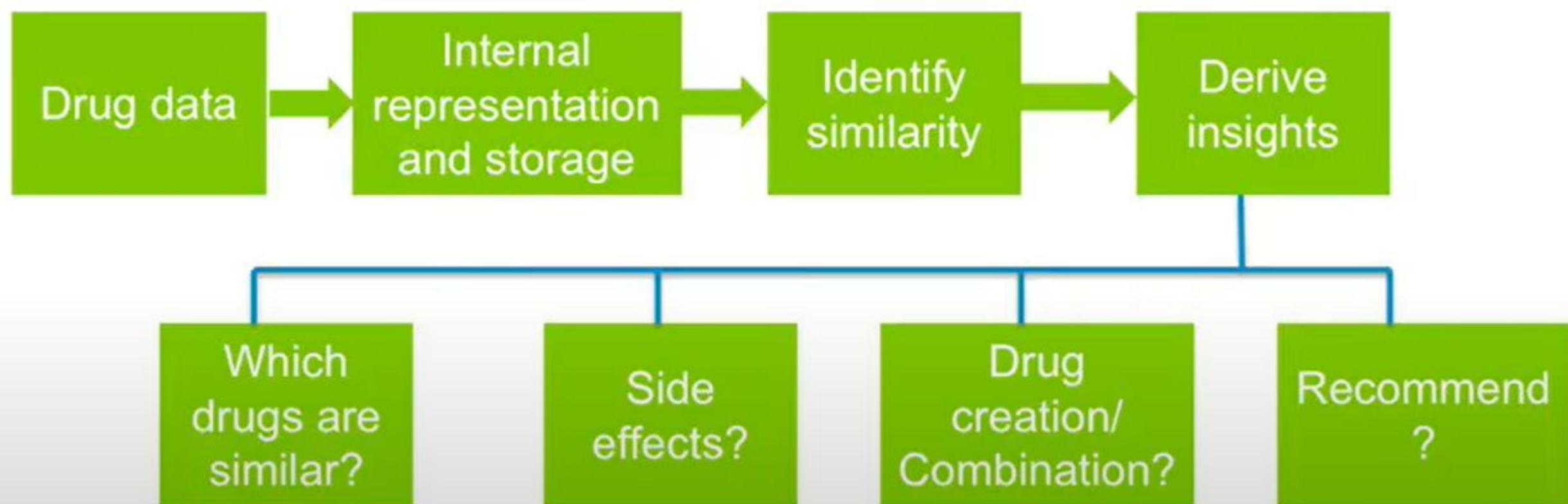


Estudo de caso #3

- Bioinformática;
- <https://databricks.com/session/distributed-end-to-end-drug-similarity-analytics-and-visualization-workflow>

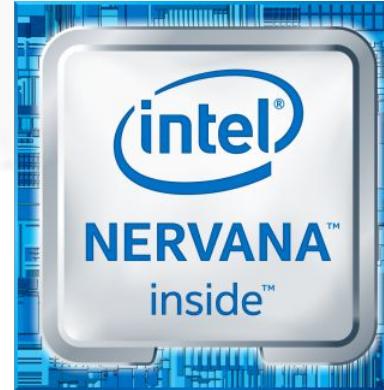


IGTI



Estudo de caso #3

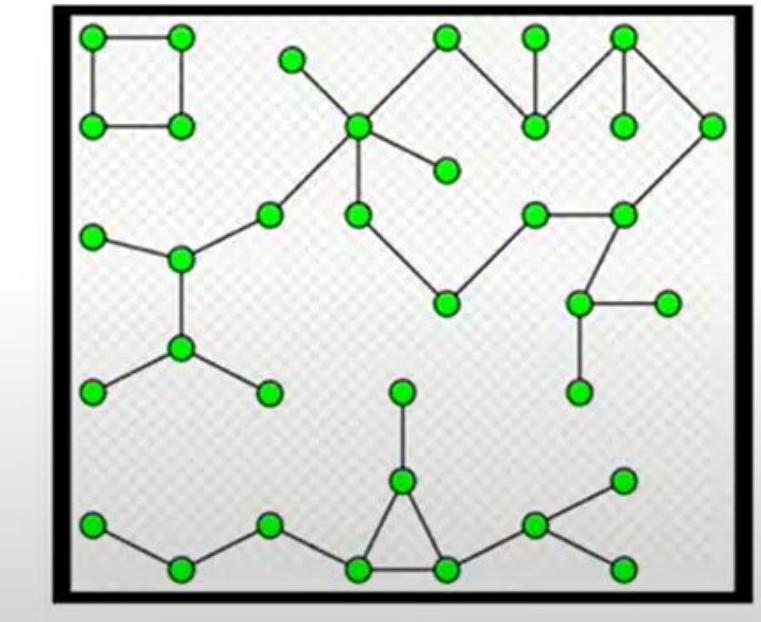
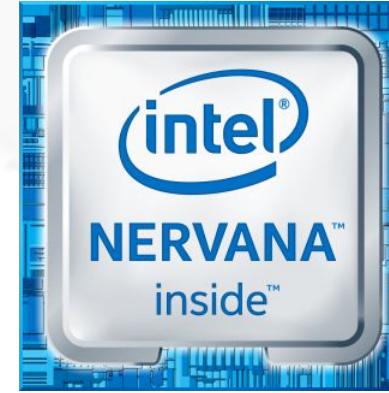
- Dados crus:



id	name	SMILES	chembl_id	chebi_id	DrugBank	PubChem	MeSH_ID	KEGG_drug
0	levodopa	C1=CC(=C(C=C1C[C@H](C(=O)O)N)O)O	CHEMBL1009	15765	DB01235	6047	D007980	D00059
1	nipecotic acid	C1CC(CNC1)C(=O)O	CHEMBL277498	116931	NA	4498	C030278	NA
2	betaxolol	CC(C)NCC(COC1=CC=C(C=C1)CCOCC2CC2)O	CHEMBL423	3082	DB00195	2369	D015784	NA
3	oxybutynin	CCN(CC)CC#CCOC(=O)C(C1CCCCC1)(C2=CC=CC=C2)O	CHEMBL1231	7856	DB01062	4634	C005419	D00465
4	indapamide	CC1CC2=CC=CC=C2N1NC(=O)C3=CC(=C(C=C3)Cl)S(=O)(=O)N	CHEMBL406	100619	DB00808	3702	D007190	D00345
5	anabasine	C1CCNC(C1)C2=CN=CC=C2	CHEMBL280963	28986	NA	2181	D000691	NA
6	tamoxifen	CC/C(=C\CC=CC=C1)/C2=CC=C(C=C2)OCCN(C)C/C3=CC=CC=C3	CHEMBL83	41774	DB00675	2733526	D013629	D00966
7	eucatropine	CC1CC(CC(N1C)(C)C)OC(=O)C(C2=CC=CC=C2)O	CHEMBL1618660	NA	NA	7534	NA	NA
8	flucytosine	C1=NC(=O)NC(=C1F)N	CHEMBL1463	5100	DB01099	3366	D005437	D00323
9	terazosin	COC1=C(C=C2C(=C1)C(=NC(=N2)N3CCN(CC3)C(=O)C4CCCO4)N)OC	CHEMBL611	9445	DB01162	5401	C041226	NA
10	citiolone	CC(=O)NC1CCSC1=O	CHEMBL2104457	NA	NA	14520	C000486	NA
11	nitrofurantoin	C1C(=O)NC(=O)N1/N=C\C2=CC=C(O2)[N+](=O)[O-]	CHEMBL572	116592	DB00698	5353830	D009582	D00439
12	lomustine	C1CCC(CC1)NC(=O)N(CCC1)N=O	CHEMBL514	110898	DB01206	3950	D008130	D00363
13	meptazinol	CCC1(CCCCN(C1)C)C2=CC(=CC=C2)O.Cl	CHEMBL314437	239237	NA	65483	D008621	NA
14	rilmendine	C1CC1C(C2CC2)NC3=NCCO3	CHEMBL289480	160422	NA	68712	C032302	NA
15	bufexamac	CCCCOC1=CC=C(C=C1)CC(=O)NO	CHEMBL94394	253408	NA	2466	D002019	NA
16	mebhydrolin	CN1CCC2=C(C1)C3=CC=CC=C3N2CC4=CC=CC=C4	CHEMBL1625607	NA	NA	22530	C005139	NA
18	piperidolate	CCN1CCCC(C1)OC(=O)C(C2=CC=CC=C2)C3=CC=CC=C3.Cl	CHEMBL1474977	998861	NA	8520	C010293	NA
20	metamizole sodium	CC1=C(C(=O)N(N1C)C2=CC=CC=C2)N(C)CS(=O)(=O)[O-].[Na+]	CHEMBL487894	59033	DB04817	522325	D004177	NA
21	sulfamethoxazole	CC1=CC(=NO1)NS(=O)(=O)C2=CC=C(C=C2)N	CHEMBL443	9332	DB01015	5329	D013420	D00447

Estudo de caso #3

- Benefícios do Spark:
 - Spark Core: sem necessidade de se preocupar em como paralelizar a computação, foco nos algoritmos;
 - GraphX: vários algoritmos que operam sobre grafos implementados.

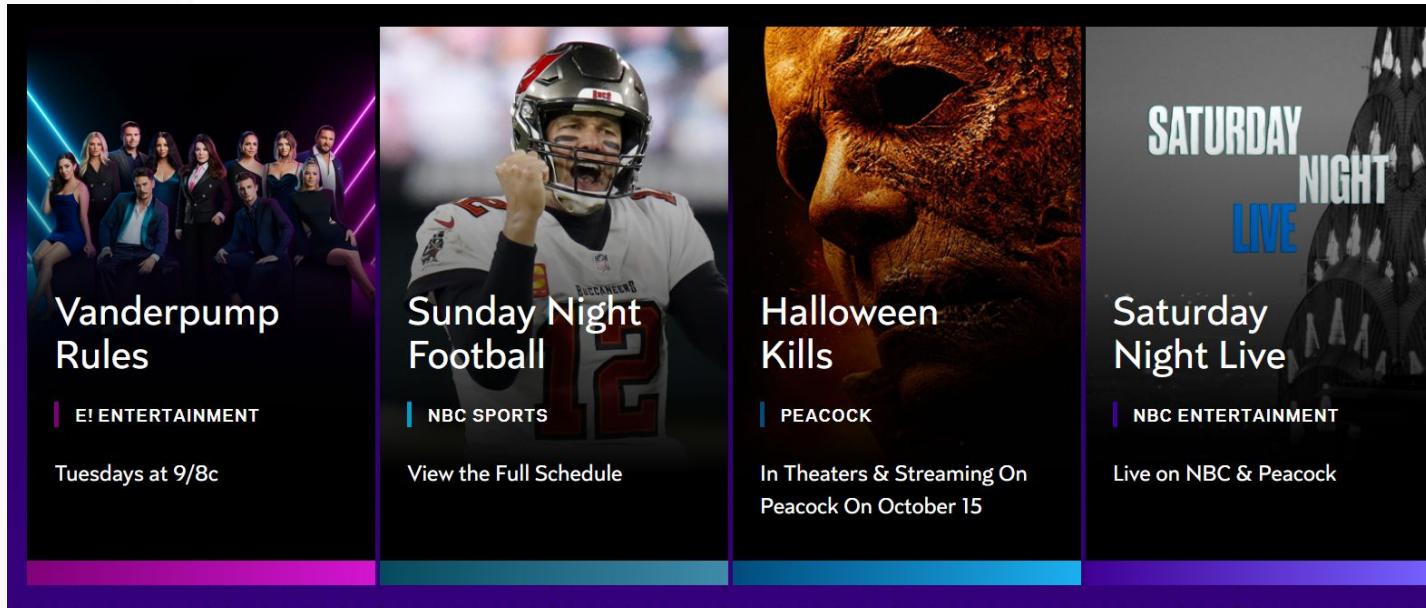


Estudo de caso #4

- NBC: maiores empresas de mídia do mundo;
- time de infraestrutura precisa servir vídeos sob demanda;
- <https://databricks.com/blog/2014/09/24/apache-spark-improves-the-economics-of-video-distribution-at-nbc-universal.html>

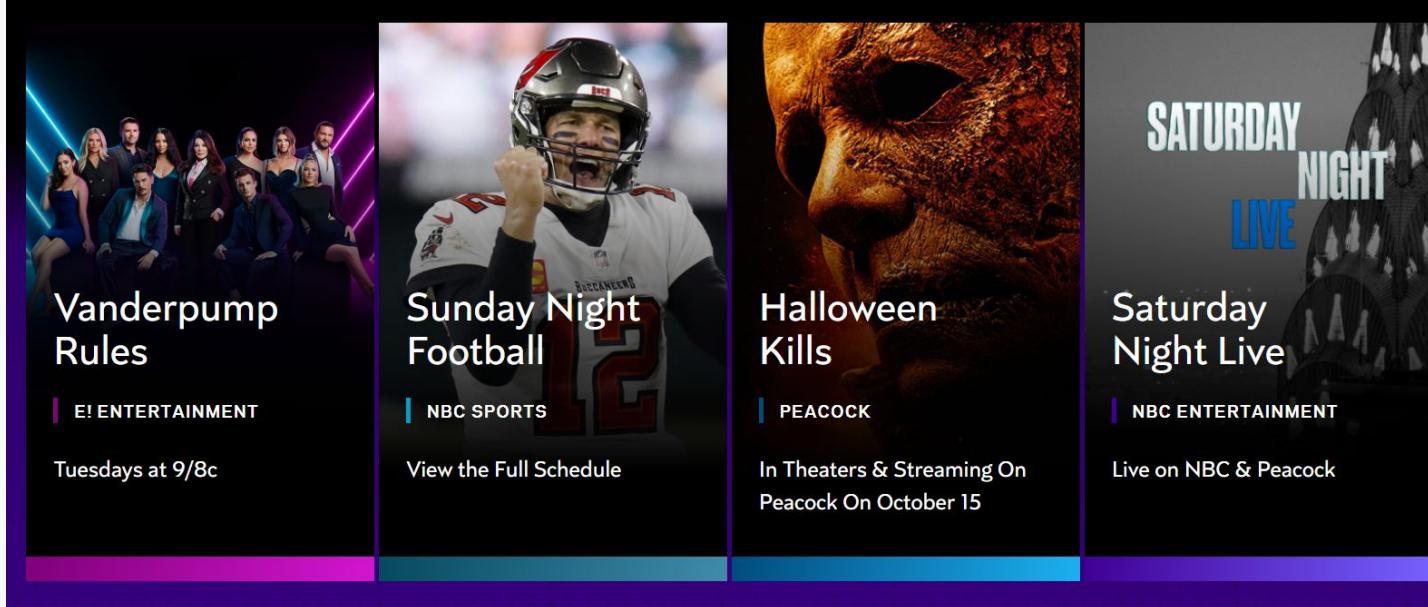


IGTI



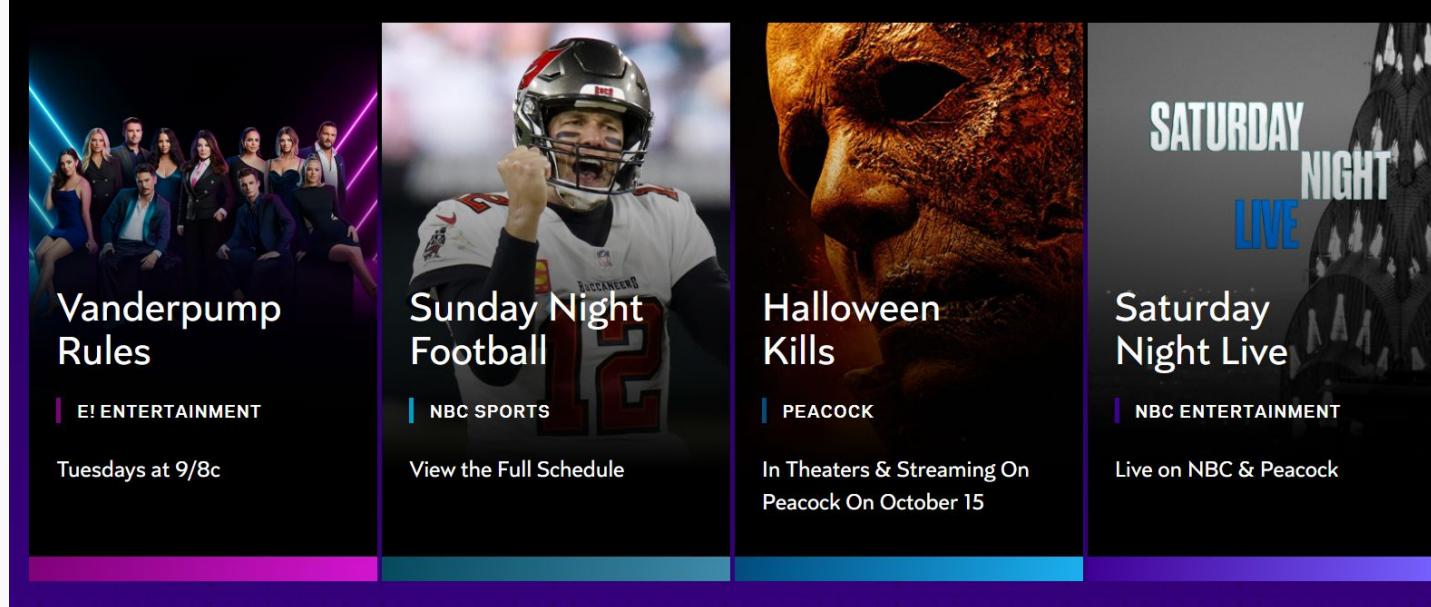
Estudo de caso #4

- cachear todo o conteúdo: baixa latência, alto custo;
- não cachear nada: alta latência para os consumidores;
- problema de negócio: como encontrar o melhor compromisso?



Estudo de caso #4

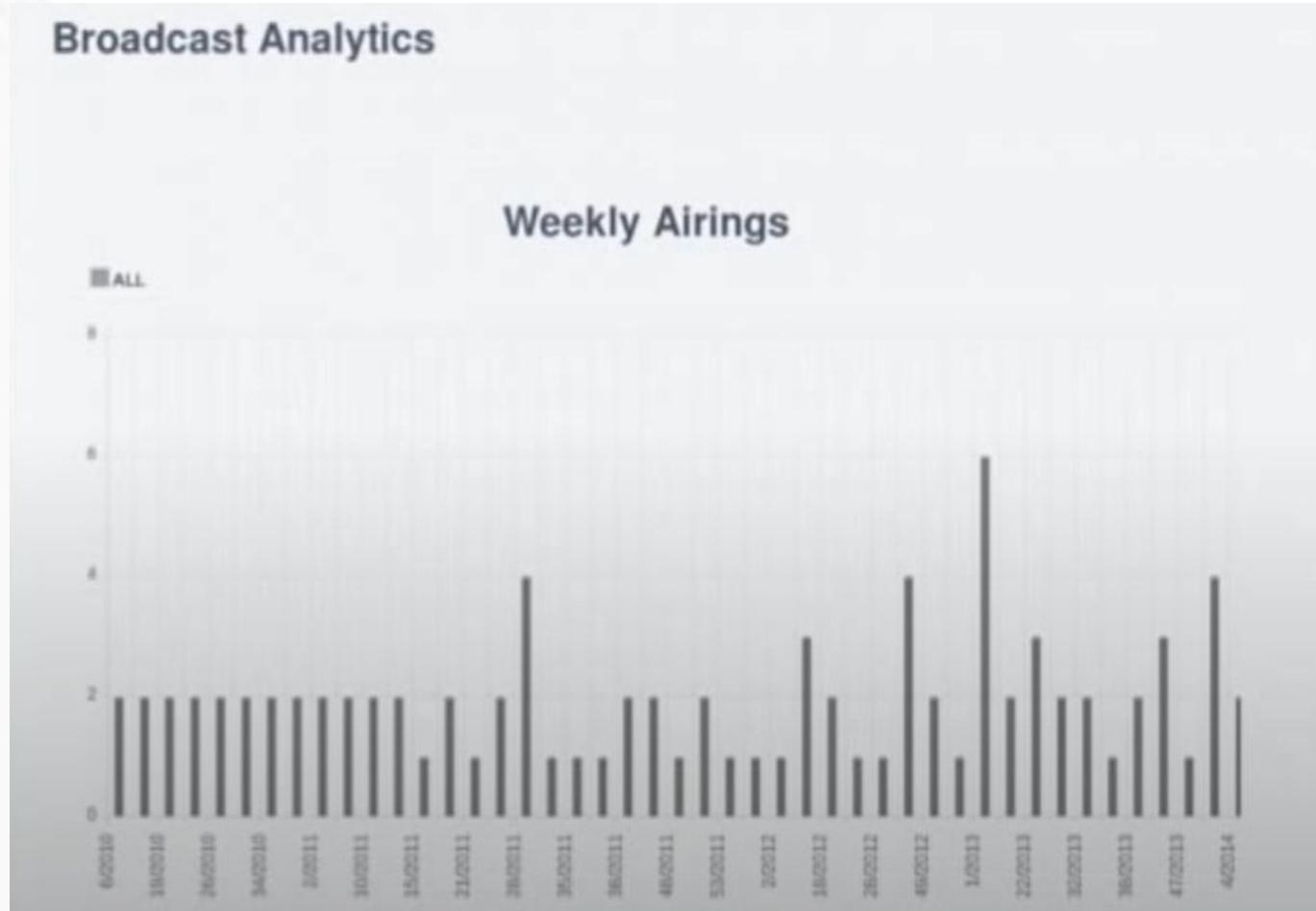
- GB de metadados sobre os vídeos:
 - IDs dos vídeos
 - duração
 - data e hora de transmissão
 - canal



IGTI

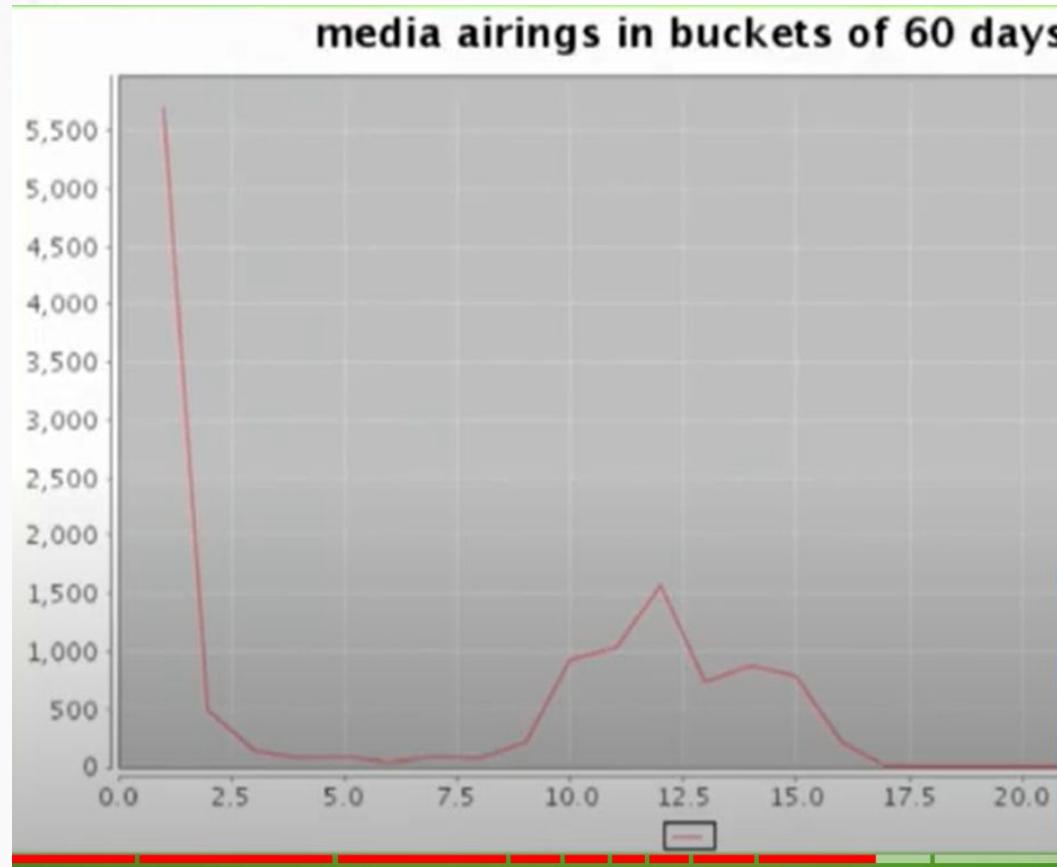
Estudo de caso #4

- fluxo Oracle → Apache Spark → HBase executa todas as noites



Estudo de caso #4

- 10x ganho com 18 cores versus 1 core
- Com muitos dados, era inviável executar sem o cluster



Quanto evitar o Spark?

- O volume de dados não for grande
 - Spark é útil a partir de dezenas / centenas de GBs
- Poucos recursos computacionais e pouca memória
 - Spark é desenhado para operar com abundância de CPU e memória

Quanto evitar o Spark?

- O volume de dados não for grande
 - Spark é útil a partir de dezenas / centenas de GBs
- Poucos recursos computacionais e pouca memória
 - Spark é desenhado para operar com abundância de CPU e memória

Quanto evitar o Spark?

- Busca textual:

```
1 sqlContext.sql("select * from myTable where name like '%abc%' ")
```



Desenvolvimento de Soluções Utilizando Spark

Capítulo 1. Introdução ao Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 1.4. Spark vs Hadoop MapReduce

Prof. Pedro Calais

O que é o Apache Hadoop?

IGTI

- MapReduce nasceu em 2004, no Google (Jeff Dean e Sanjay Ghemawat);
- Hadoop nasceu 2006;
- Escrito em Java;
- “*A software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner*”.



O que é o Apache Hadoop MapReduce?



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

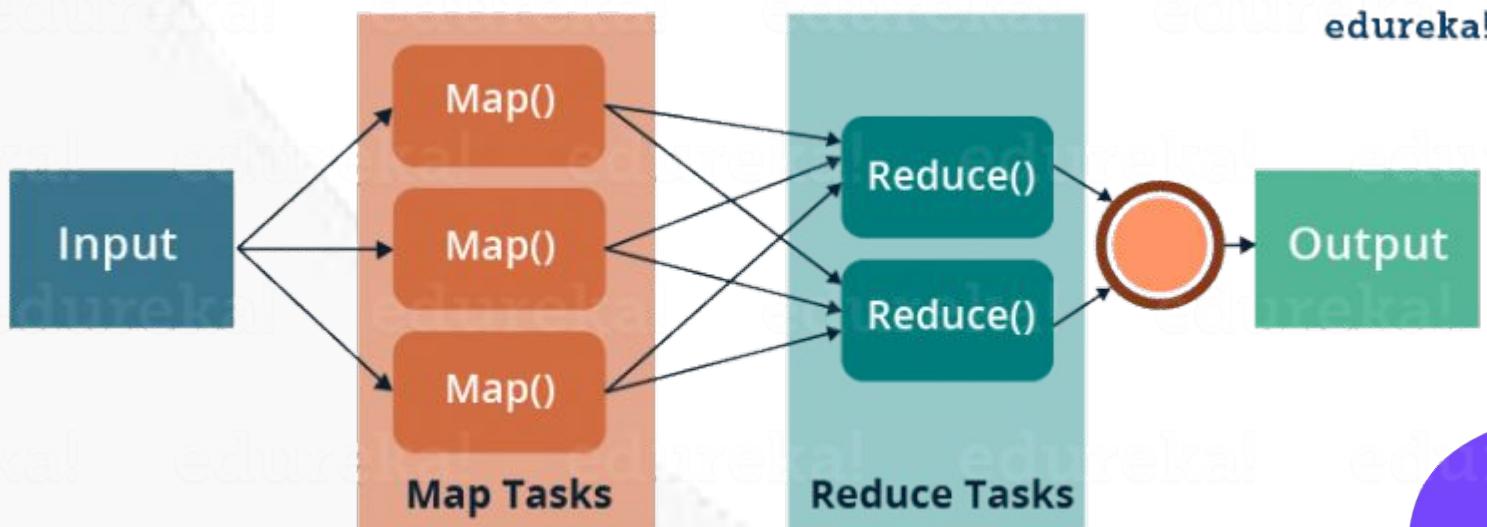
Google, Inc.

3 October 2004

O Modelo de Programação MapReduce

IGTI

- *map*: filtra e ordena dados ao expô-los como pares chave-valor;
- *reduce*: a partir do mapeamento, summariza os dados;
- inspirado em programação funcional.



O Modelo de Programação MapReduce



- *map*: filtra e ordena dados ao expô-los como pares chave-valor;
- *reduce*: a partir do mapeamento, summariza os dados.

```
function map(String name, String document):
    // name: document name
    // document: document contents
    for each word w in document:
        emit (w, 1)

function reduce(String word, Iterator partialCounts):
    // word: a word
    // partialCounts: a list of aggregated partial counts
    sum = 0
    for each pc in partialCounts:
        sum += pc
    emit (word, sum)
```

O Hadoop e seus módulos

- Hadoop Common;
- Hadoop MapReduce;
- Hadoop Distributed File System (HDFS);
- Hadoop YARN;
- Hadoop Ozone.



Spark pode ser executado com o YARN

IGTI

- YARN = *Yet Another Resource Negotiator*
 - gestor de recursos do cluster
 - agendamento e monitoramento de *jobs*



Spark pode ser executado com o HDFS



- *HDFS = Hadoop Distributed File System*

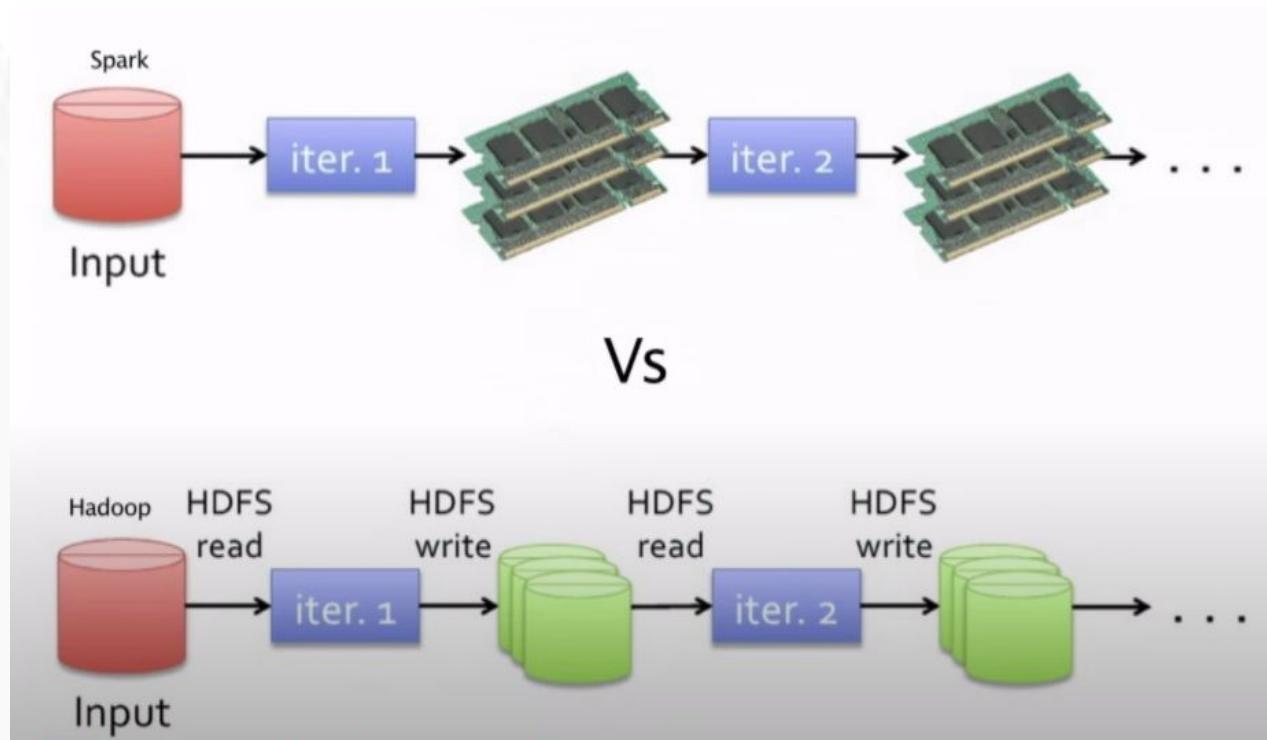
- armazena grandes conjuntos de dados em hardware commodity
- centenas / milhares de nós
- tolerância a falhas via réplicas



Desempenho: Spark vs Hadoop

IGTI

MapReduce



fonte: Music Recommendations at Scale with Spark - Christopher Johnson (Spotify)

Desempenho: Spark vs Hadoop

MapReduce



- Spark pode ser entre 10 e 100 vezes mais veloz que o Hadoop MapReduce
 - Spark faz menos leituras e escritas em disco
 - Spark armazena dados intermediários em memória sempre que possível
- Spark brilha em tarefas iterativas

Tolerância a falhas: Spark vs Hadoop



MapReduce

- Ambos são tolerantes a falhas;
- Como o Hadoop usa mais o disco, ele consegue se recuperar de falhas mais rapidamente.

Spark: plataforma *unificada*!

IGTI

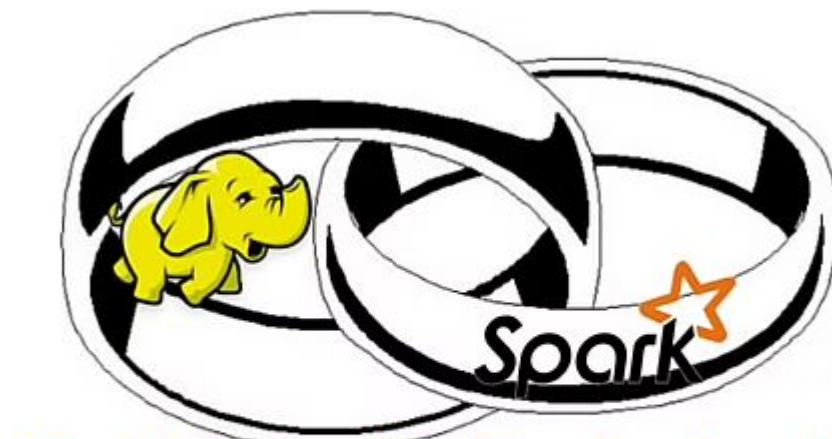
- *Hadoop*: batch
- Spark: batch, tempo real, iterativo, grafos



Spark e Hadoop podem cooperar!

IGTI

- Apache Spark como motor de processamento dos dados
- Apache Hadoop como armazenamento distribuído e gestor de recursos
 - HDFS
 - YARN



Not Mutually Exclusive But
Better Together

DeZyre

fonte: projectpro.io

Fim da Aula 1!

Big Data e o Apache Spark

Vantagens e desvantagens do Apache Spark

Estudos de caso reais

Spark vs Hadoop MapReduce

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 2.1. A Arquitetura do Apache Spark

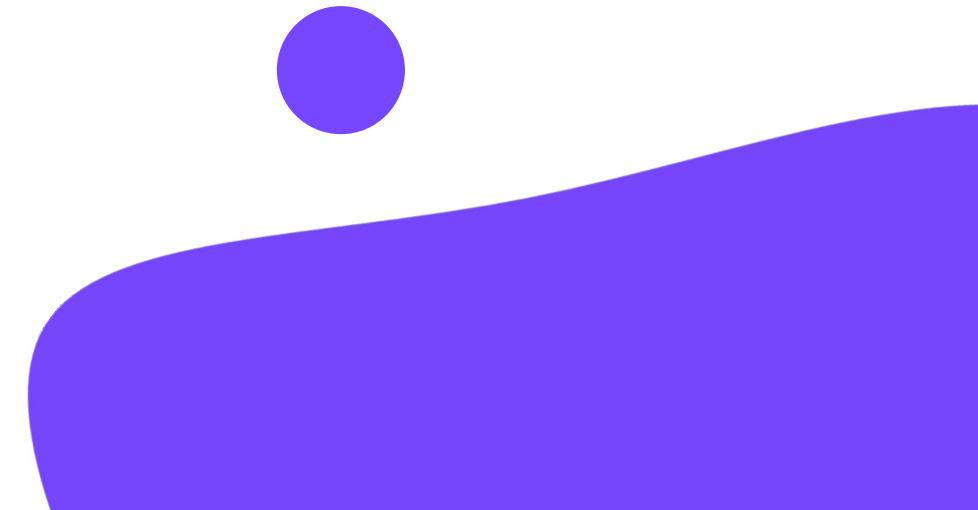
Prof. Pedro Calais

O que você já sabe:

IGTI

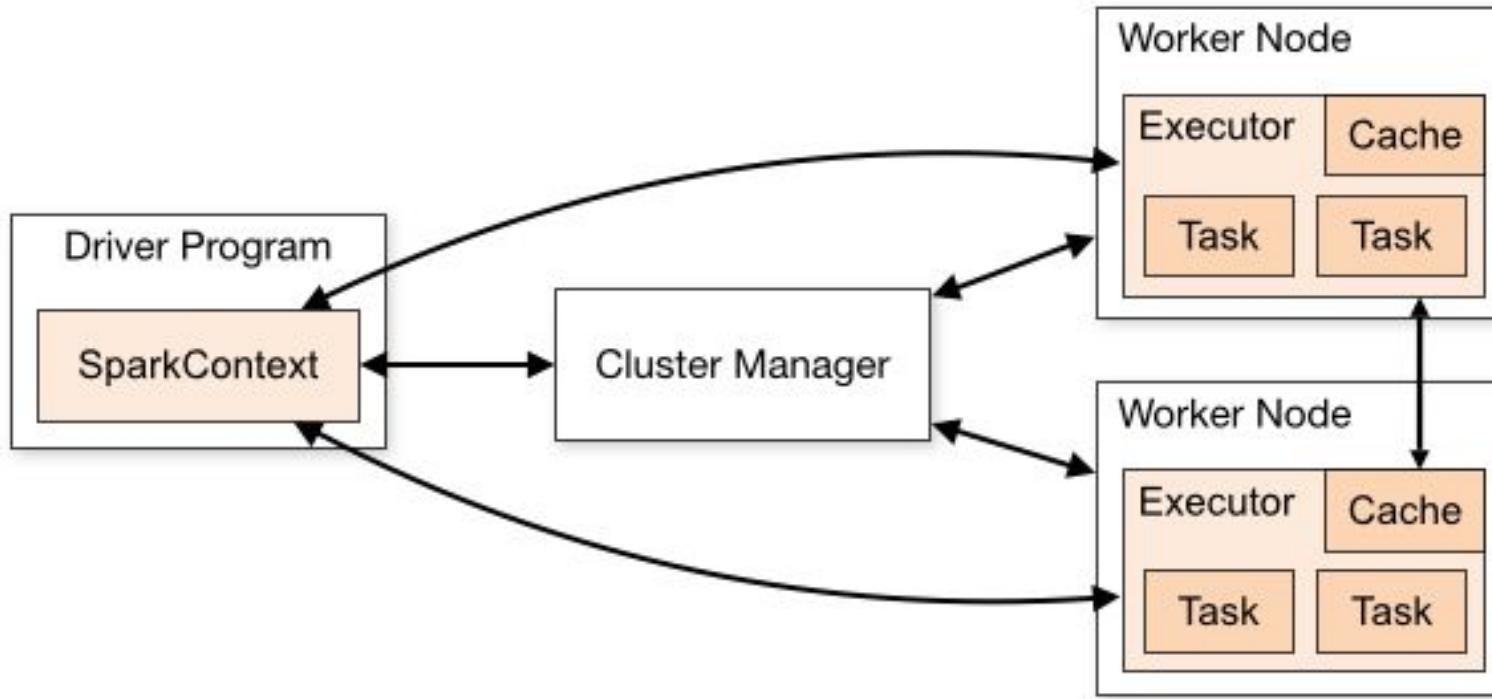
- **O Spark é uma plataforma para processamento de grandes volumes de dados**

Antes de começar a programar em Spark, vamos descer um nível e entender um pouco mais sobre a arquitetura, ou seja, seus principais componentes e como eles interagem.

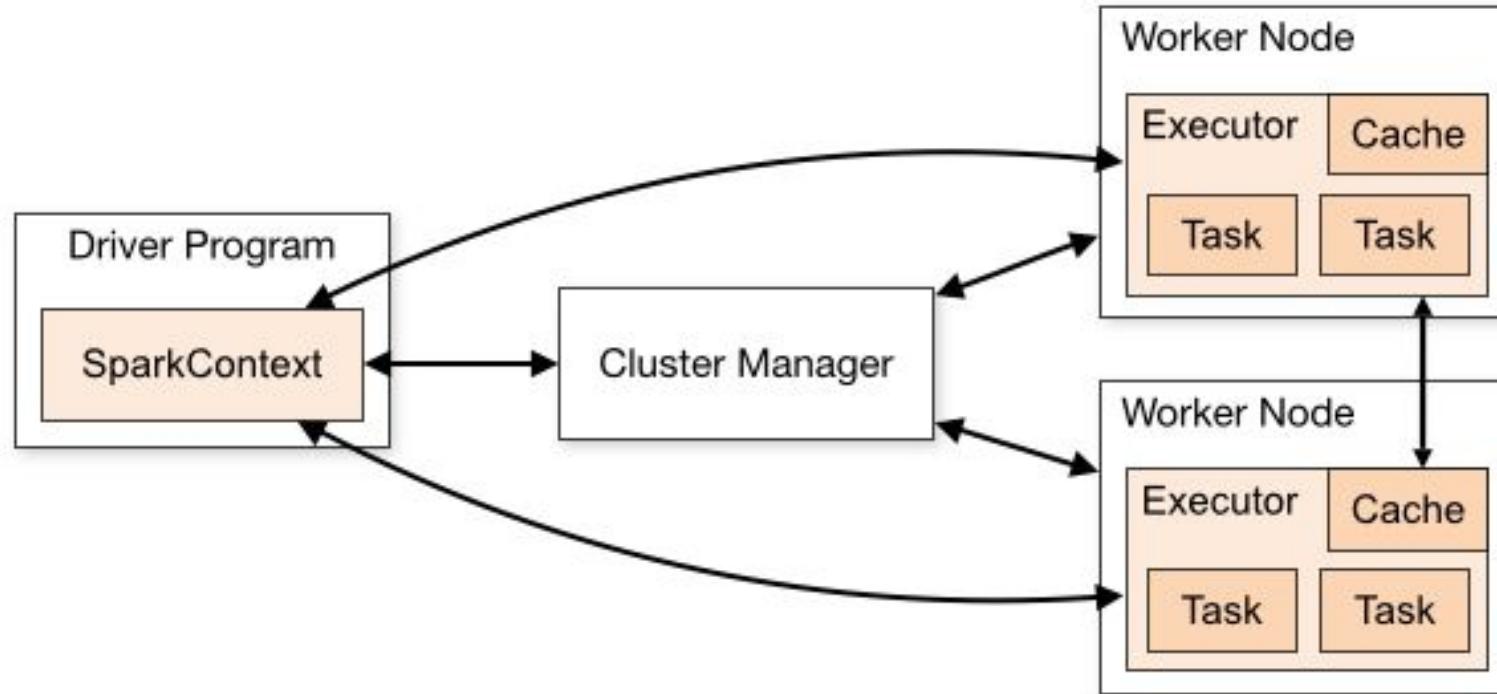


Componentes do Spark

IGTI

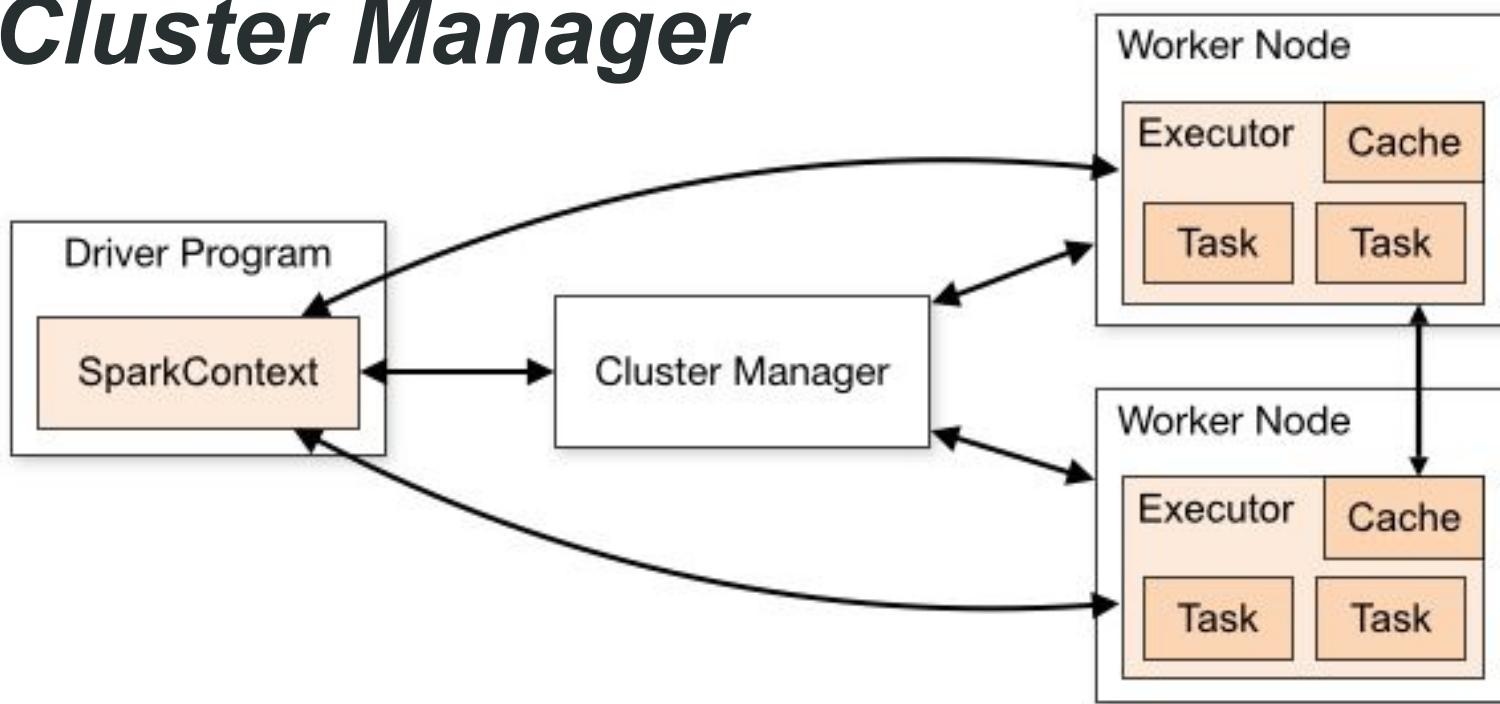


Driver Program



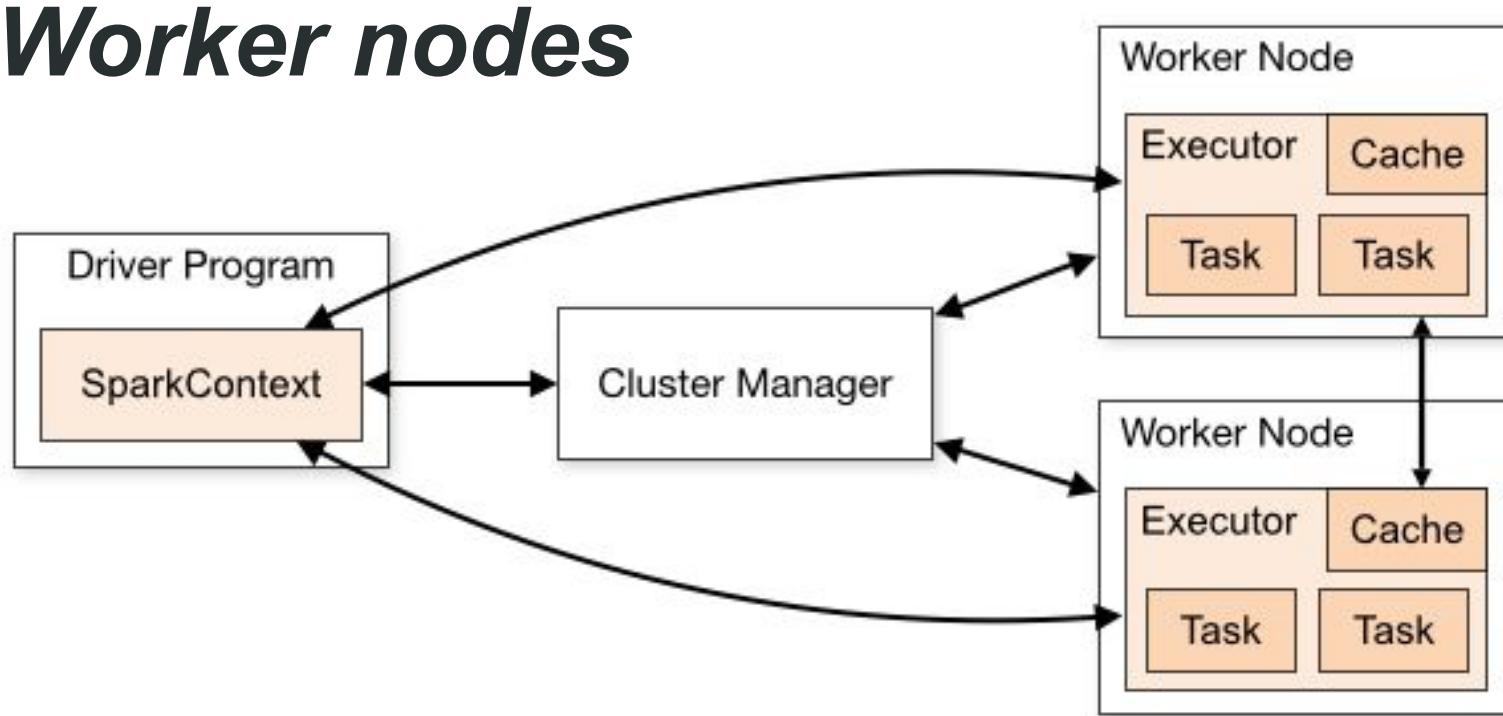
- orquestra as operações paralelas executadas no cluster;
- comunica com o gestor do cluster;
- agenda operações a serem executadas no cluster;
- distribui a alocação nos workers / executors.

Cluster Manager



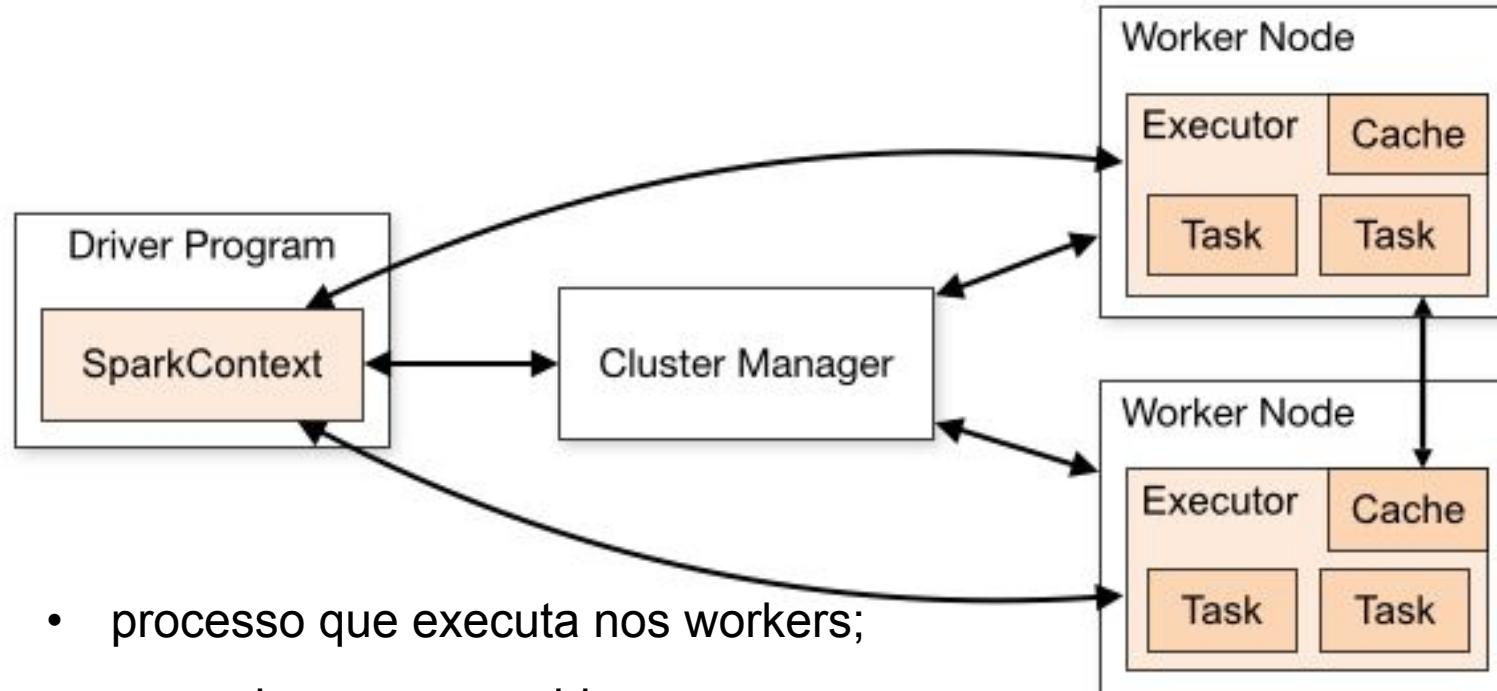
- serviço externo que gerencia e aloca recursos do cluster;
- gerenciadores:
 - built-in;
 - Apache Hadoop YARN;
 - Apache Mesos;
 - Kubernetes.

Worker nodes



- nó do cluster que pode executar a aplicação

Spark executors



- processo que executa nos workers;
- comunicam com os drivers;
- executam tarefas.

RDD: a principal abstração do Spark

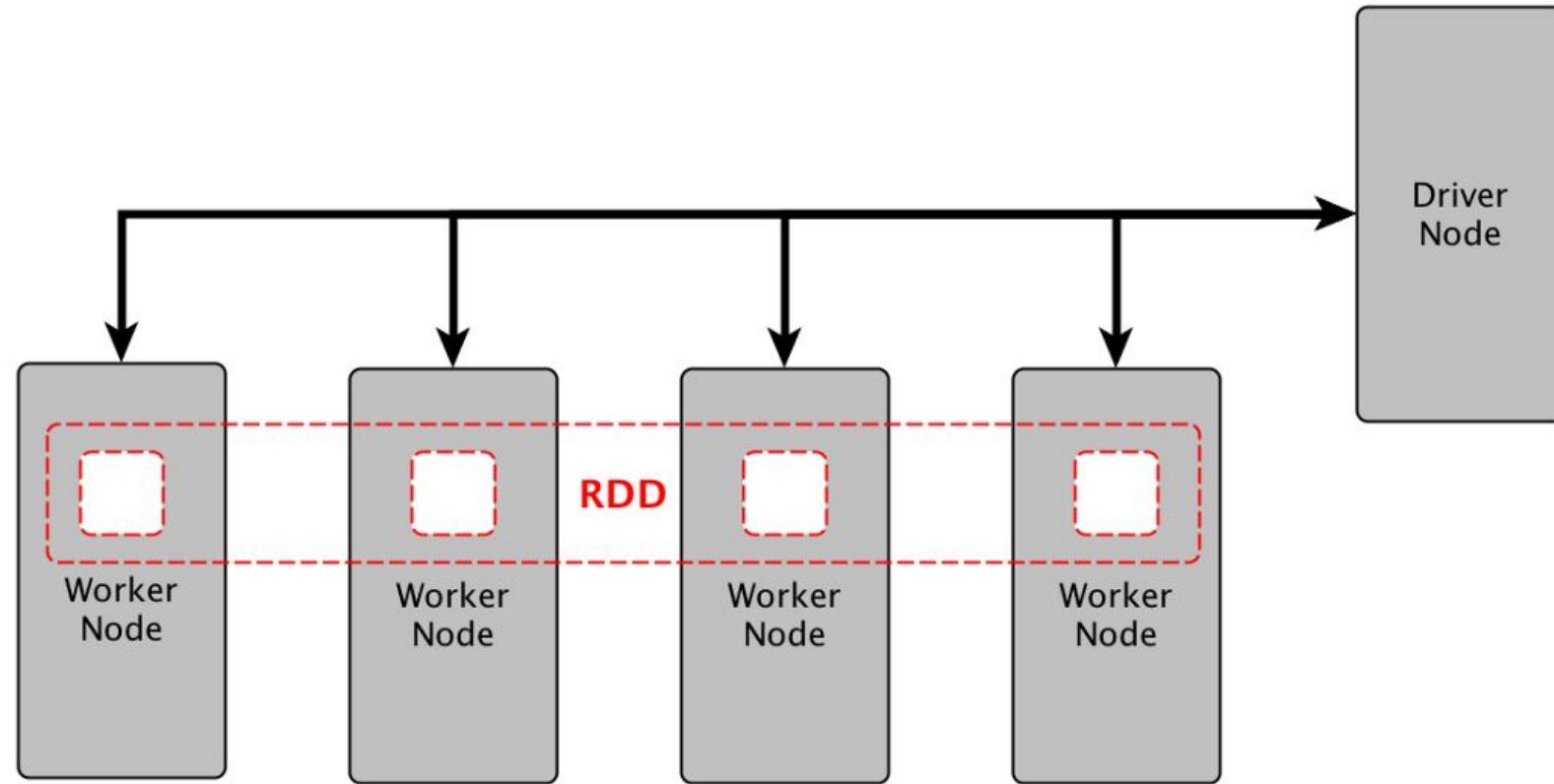


RDD: a principal abstração do Spark



- **Resilient Distributed Dataset;**
- **Dataset:** Representa uma coleção de objetos;
- **Distribuído:** Particionados em um conjunto de máquinas;
- **Resiliente:** Pode ser reconstruído se uma partição for perdida.

RDD e operações paralelas



fonte: <https://medium.com/@lavishj77/spark-fundamentals-part-2-a2d1a78eff73>

Tolerância a falha com RDDs

- Se uma partição de um RDD é perdida, o RDD tem informação suficiente para reconstruir apenas a partição perdida;
- *lineage*: sequência de operações que geraram a partição perdida.:

3 formas de se criar um RDD

- A partir de uma coleção;
- *driver* vai dividir a coleção em pedaços e enviá-los para os *workers*;

```
list=[1,2,3,4,5]
```

```
rdd1=sc.parallelize(list)
```

3 formas de se criar um RDD

- A partir de um sistema de arquivos local ou distribuído como o HDFS (Hadoop File System)

```
rdd2=sc.textFile("path of file")
```



3 formas de se criar um RDD



- transformando um RDD a partir de outro RDD

```
rddf=rdd.flatMap(lambda line:line.split())
```

Variáveis de *broadcast*

- Em alguns cenários, todos os *workers* precisam acessar os mesmos dados.
 - Por exemplo, uma tabela de consulta (lookup table);
 - `sc.broadcast()` copia para todos os *workers* uma cópia da variável apenas uma vez.

Recapitulando

- arquitetura do Spark
 - programa *driver* que, por meio de um *cluster manager*, coordena *workers*
- abstração fundamental do Spark: RDD
 - modelo de memória distribuída compartilhada (DSM)

Na próxima aula

Instalando o Apache Spark

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 2.2. Instalando o Apache Spark

Prof. Pedro Calais

Passo 1: Baixando o Apache Spark



<https://spark.apache.org/downloads.html>

Download Apache Spark™

1. Choose a Spark release: [3.1.2 \(Jun 01 2021\) ▾](#)
2. Choose a package type: [Pre-built for Apache Hadoop 3.2 and later ▾](#)
3. Download Spark: [spark-3.1.2-bin-hadoop3.2.tgz](#)
4. Verify this release using the [3.1.2 signatures, checksums](#) and [project release KEYS](#).

Passo 2: Instalando o Spark



- Confirme que você tem o Java instalado:

```
pcalais:~/Downloads$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~20.04-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
```

Passo 2: Instalando o Spark

<https://phoenixnap.com/kb/install-spark-on-ubuntu>

<https://phoenixnap.com/kb/install-spark-on-windows-10>

```
$ tar xvf spark-3.1.2-bin-hadoop3.2.tgz
```

```
sudo mv spark-3.1.2-bin-hadoop3.2 /opt/spark
```

```
$ echo "export SPARK_HOME=/opt/spark" >> ~/.profile  
$ echo "export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin" >> ~/.profile  
$ echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.profile  
$ source ~/.profile
```

Passo 3: Instalando o PySpark

IGTI

https://spark.apache.org/docs/latest/api/python/getting_started/install.html

- Spark é escrito em Scala
- PySpark é a API Python para o Spark



Passo 3: Instalando o PySpark

https://spark.apache.org/docs/latest/api/python/getting_started/install.html

- Confirme que você tem o python instalado:

```
pcalais:~/Downloads$ python3 --version  
Python 3.8.10
```

- Instale o PySpark:

```
$ pip install pyspark  
$ pip3 install pyspark
```

Passo 3: Instalando o PySpark

- PySpark shell é um modo interativo:

```
pcalais:~/Downloads$ pyspark
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/10/17 18:54:36 WARN Utils: Your hostname, pcalais-Inspiron-15-7000-Gaming resolves to a loopback
interface wlp3s0)
21/10/17 18:54:36 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/17 18:54:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

    / \ / \
    \ / - \ - \ / \
    / \ / \ , / / / \ \ \
    / \ / \ / \ / \ / \ / \
    / \ / \ / \ / \ / \ / \ / \
version 3.1.2

Using Python version 3.8.10 (default, Sep 28 2021 16:10:42)
Spark context Web UI available at http://192.168.0.11:4040
Spark context available as 'sc' (master = local[*], app id = local-1634507678142).
SparkSession available as 'spark'.
>>> █
```

Passo 3: Instalando o PySpark



```
>>> spark.version  
'3.1.2'  
>>> █
```

Objeto *spark* é um *SparkSession*

```
Python 3.8.10 | Spark 3.1.2  
Using Python version 3.8.10 (default, Sep 28 2021 16:10:42)  
Spark context Web UI available at http://192.168.0.11:4040  
Spark context available as 'sc' (master = local[*], app id = local-1634507678142).  
SparkSession available as 'spark'.  
>>> █
```

- *SparkSession* é um objeto que serve como ponto de entrada para interagir com o Spark.
 - no modo interativo, o *SparkSession* é criado automaticamente;
 - em uma aplicação Spark, você é responsável por criar o *SparkSession*.

Passo 3: Instalando o PySpark

- spark-shell é uma interface interativa em Scala

```
pcalais:~/Downloads$ spark-shell
21/10/17 19:12:20 WARN Utils: Your hostname, pcalais-Inspiron-15-7000-Gaming resolves to a loopback address: 127.0.1.1; using 192.168.0.11 instead (on interface wlp3s0)
21/10/17 19:12:20 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/10/17 19:12:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.0.11:4040
Spark context available as 'sc' (master = local[*], app id = local-1634508745785).
Spark session available as 'spark'.
Welcome to

      \____/-
     /_\  \_`/_/ \_/\_ \
    /  \  .\_/\_,/_/_/ \_/\_ \
   /_/
version 3.1.2

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_292)
Type in expressions to have them evaluated.
Type :help for more information.

scala> spark.version
res0: String = 3.1.2
```

Primeiro exemplo com spark-shell



```
scala> val strings = spark.read.text("/opt/spark/README.md")
strings: org.apache.spark.sql.DataFrame = [value: string]

scala> strings.show(10, false)
+-----+
|value
+-----+
| # Apache Spark
|
| Spark is a unified analytics engine for large-scale data processing. It provides
| high-level APIs in Scala, Java, Python, and R, and an optimized engine that
| supports general computation graphs for data analysis. It also supports a
| rich set of higher-level tools including Spark SQL for SQL and DataFrames,
| MLlib for machine learning, GraphX for graph processing,
| and Structured Streaming for stream processing.
|
| <https://spark.apache.org/>
+-----+
only showing top 10 rows

scala> strings.count()
res1: Long = 108

scala> ■
```

Primeiro exemplo com PySpark



```
Welcome to  
  \  /\ / \  \  \  / / /    version 3.1.2  
  / / / . - \  - , / / / / \ \ /  
  
Using Python version 3.8.10 (default, Sep 28 2021 16:10:42)  
Spark context Web UI available at http://192.168.0.11:4040  
Spark context available as 'sc' (master = local[*], app id = local-1634553669366).  
SparkSession available as 'spark'.  
>>> strings = spark.read.text("/opt/spark/README.md")  
>>> strings.show(10, truncate=False)  
+-----+  
|value|  
+-----+  
|# Apache Spark  
|  
|Spark is a unified analytics engine for large-scale data processing. It provides  
|high-level APIs in Scala, Java, Python, and R, and an optimized engine that  
|supports general computation graphs for data analysis. It also supports a  
|rich set of higher-level tools including Spark SQL for SQL and DataFrames,  
|MLlib for machine learning, GraphX for graph processing,  
|and Structured Streaming for stream processing.  
|  
|<https://spark.apache.org/>  
+-----+  
only showing top 10 rows  
  
>>> strings.count()  
108  
>>> █
```

Passo 4: Instalando o JupyterLab



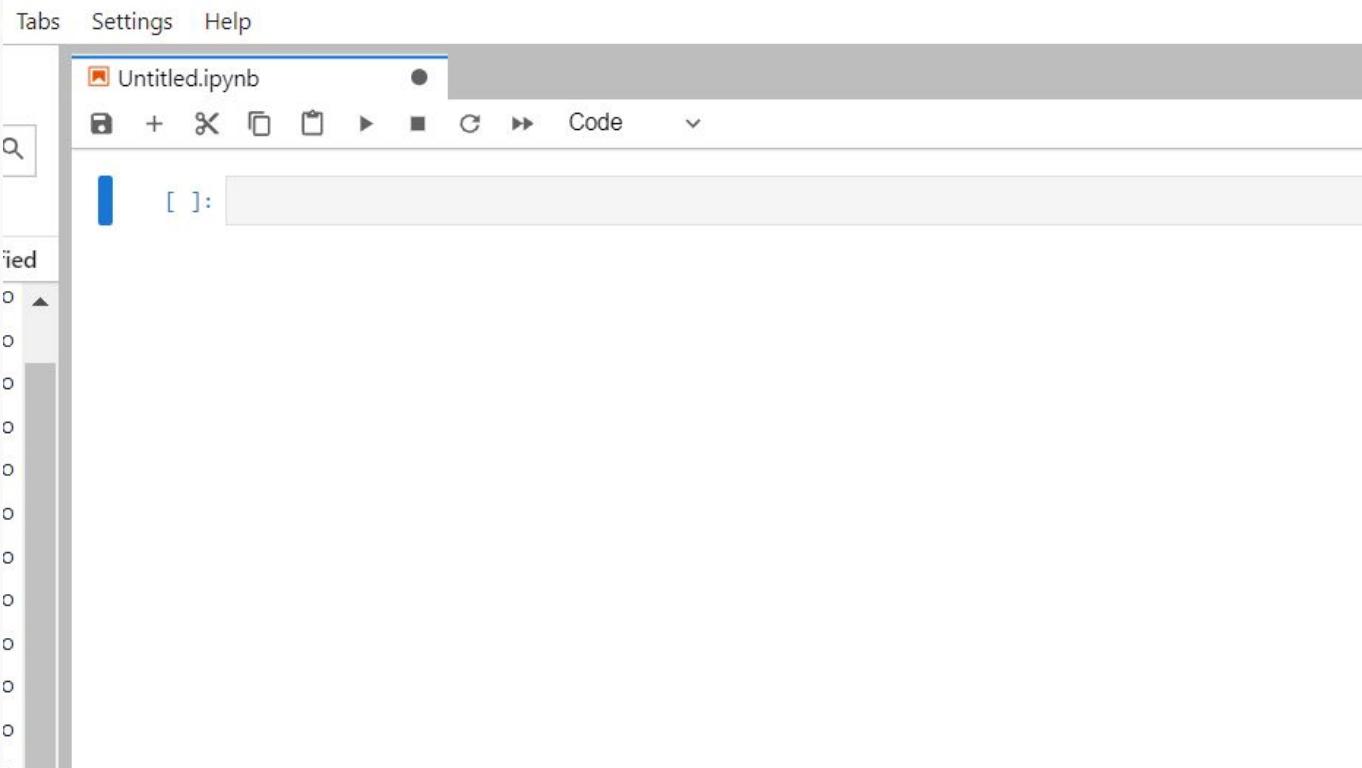
- ambiente Web interativo
- <https://jupyter.org/install>
- <https://stackoverflow.com/questions/28831854/how-do-i-add-python3-kernel-to-jupyter-ipython>

The screenshot shows the official Jupyter website at <https://jupyter.org/>. The top navigation bar includes links for Install, About Us, Community, Events, Documentation, NBViewer, JupyterHub, Widgets, Blog, and Security. Below the navigation is a large image of the JupyterLab interface, which is a web-based development environment. It features multiple panes: a left sidebar with a file tree and notebook list; a central notebook editor with code and output cells; and a bottom pane with a terminal and other tools. A text overlay on the right side of the image reads: "JupyterLab: Jupyter's Next-Generation Notebook Interface". Below this, a detailed description of JupyterLab is provided: "JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones."

Passo 4: Instalando o JupyterLab

IGTI

- ambiente Web interativo



Passo 4: Instalando o JupyterLab



- ambiente Web interativo

A screenshot of the JupyterLab interface. At the top, there's a menu bar with 'bs', 'Settings', and 'Help'. Below the menu is a toolbar with icons for file operations like new, open, save, and run. The main area shows two code cells. The first cell, labeled '[*]:', contains the Python code 'from pyspark.sql import SparkSession'. The second cell, labeled '[]:', is empty and has a blue vertical indicator bar to its left.

```
[*]: from pyspark.sql import SparkSession
```

Recapitulando

- Baixamos o Spark
- Instalamos o Spark e entramos no spark-shell
- Instalamos o PySpark
- Executamos código de forma interativa
- Instalamos o JupyterLab

Execute os passos desta aula!! :-)

Na próxima aula

Contando palavras e números: o “Hello World” do Apache Spark

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 2.3. Contando palavras: o “Hello World” do Apache Spark

Prof. Pedro Calais

Já deu uma olhada em /examples?



```
pcalais:/opt/spark$ ls -lh examples/src/main/python/
total 60K
-rwxr-xr-x 1 pcalais pcalais 3,2K mai 24 01:45 als.py
-rw-r--r-- 1 pcalais pcalais 3,1K mai 24 01:45 avro_inputformat.py
-rwxr-xr-x 1 pcalais pcalais 2,7K mai 24 01:45 kmeans.py
-rwxr-xr-x 1 pcalais pcalais 3,1K mai 24 01:45 logistic_regression.py
drwxr-xr-x 2 pcalais pcalais 4,0K mai 24 01:45 ml
drwxr-xr-x 2 pcalais pcalais 4,0K mai 24 01:45 mllib
-rwxr-xr-x 1 pcalais pcalais 3,1K mai 24 01:45 pagerank.py
-rw-r--r-- 1 pcalais pcalais 2,3K mai 24 01:45 parquet_inputformat.py
-rwxr-xr-x 1 pcalais pcalais 1,4K mai 24 01:45 pi.py
-rwxr-xr-x 1 pcalais pcalais 1,5K mai 24 01:45 sort.py
drwxr-xr-x 3 pcalais pcalais 4,0K mai 24 01:45 sql
-rw-r--r-- 1 pcalais pcalais 2,2K mai 24 01:45 status_api_demo.py
drwxr-xr-x 2 pcalais pcalais 4,0K mai 24 01:45 streaming
-rwxr-xr-x 1 pcalais pcalais 2,4K mai 24 01:45 transitive_closure.py
-rwxr-xr-x 1 pcalais pcalais 1,4K mai 24 01:45 wordcount.py
```

examples/src/main/python/wordcount.py



```
1 import sys
2 from operator import add
3
4 from pyspark.sql import SparkSession
5
6
7 if __name__ == "__main__":
8     if len(sys.argv) != 2:
9         print("Usage: wordcount <file>", file=sys.stderr)
10        sys.exit(-1)
11
12     spark = SparkSession\
13         .builder\
14         .appName("PythonWordCount")\
15         .getOrCreate()
16
17     lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
18     counts = lines.flatMap(lambda x: x.split(' ')) \
19                 .map(lambda x: (x, 1)) \
20                 .reduceByKey(add)
21     output = counts.collect()
22     for (word, count) in output:
23         print("%s: %i" % (word, count))
24
25     spark.stop()
26
```

Gerando números aleatórios



```
1 from random import randint
2
3 # Generates random numbers.
4
5 # 2 billions of numbers
6 count = 2 * 1000 * 1000 * 1000
7 for _ in range(count):
8     # Random number between 0 and 10
9     value = randint(0, 10)
10    # IF value is 5, print a new line.
11    if value != 5:
12        print(value, end=" ")
13    else:
14        print(value)
15
```

Gerando números aleatórios



```
2 6 4 6 8 8 4 2 6 0 1 0 4 1 0 7 0 2 9 0 6 7 3 5  
2 1 7 2 7 2 3 0 7 3 7 10 6 3 7 7 4 3 10 10 9 7 10 6 0 7 6 6 2 1 10  
1 9 4 8 10 1 2 9 1 2 3 6 6 8 8 1 5  
1 7 0 9 8 4 3 9 0 1 9 2 5  
8 8 8 0 0 6 5  
7 7 6 2 2 2 9 4 9 2 3 9 7 6 2 6 3 0 10 10 3 3 3 10 10 7 1 3 9 4 4  
0 0 10 1 1 4 7 8 2 3 5  
8 3 5  
6 0 2 4 2 8 8 6 7 7 2 2 5  
3 8 9 5  
2 10 0 8 8 9 5  
5  
6 8 10 9 5  
9 7 4 5  
4 7 1 10 7 9 2 8 4 9 7 0 5  
6 7 2 2 1 1 1 7 6 0 3 8 5  
0 7 8 5  
5  
9 0 1 2 0 3 8 4 9 5  
7 2 3 10 1 8 8 9 4 0 4 5  
3 4 3 6 3 2 10 0 7 8 2 0 1 6 1 10 0 9 8 6 0 1 3 4 9 9 7 5  
7 6 5  
0 1 10 6 3 7 6 0 2 5  
8 10 0 6 1 1 10 0 0 2 3 9 0 5  
5  
1 4 5  
5  
4 7 4 7 1 4 8 0 8 2 9 2 2 3 2 5  
7 7 5
```

- **2 bilhões de inteiros entre 0 e 10**
- **~ 4GB**
- **~180M de linhas**

Contando números



```
1 import sys
2 from operator import add
3
4 from pyspark.sql import SparkSession
5
6 filename = "/home/pcalais/IGTI/bootcamp-cientista-dados/numbers.txt"
7
8 spark = SparkSession\
9     .builder\
10    .appName("PythonNumberCount")\
11    .master("local[*]")\
12    .getOrCreate()
13
14 linesRdd = spark.read.text(filename).rdd.map(lambda r: r[0])
15 print("Number of partitions: {}".format(linesRdd.getNumPartitions()))
16
17 counts = linesRdd.flatMap(lambda x: x.split(' ')) \
18     .map(lambda x: (x, 1)) \
19     .reduceByKey(add)
20 output = counts.collect()
21 for (number, count) in output:
22     print("%s: %i" % (number, count))
```

Spark UI: localhost:4040/jobs



A screenshot of the Apache Spark 3.2.0 User Interface. The top navigation bar includes the Apache Spark logo (3.2.0), followed by tabs for Jobs, Stages, Storage, Environment, Executors, and SQL. The 'Jobs' tab is currently selected, indicated by a grey background.

Spark Jobs (?)

User: pcalais

Total Uptime: 6,4 h

Scheduling Mode: FIFO

Completed Jobs: 19

Failed Jobs: 3

▶ Event Timeline

▼ Completed Jobs (19)

Spark UI: localhost:4040/jobs

Spark Jobs (?)

User: pcalais

Total Uptime: 7,2 h

Scheduling Mode: FIFO

Completed Jobs: 20

Failed Jobs: 3

▶ Event Timeline

▼ Completed Jobs (20)

Page:

1 Pages. Jump to . Show items in a page.

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
22	collect at /tmp/ipykernel_133766/2325597050.py:20 collect at /tmp/ipykernel_133766/2325597050.py:20	2021/10/24 18:33:49	6,7 min	2/2	64/64

lscpu mostra quantos *cores* você tem



```
pcalais:~/IGTI/bootcamp-clienta-dados$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         39 bits physical, 48 bits virtual
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):              1
```

Spark usando 8/8 cores

- comando *top*.
- lembra do paralelismo implícito?

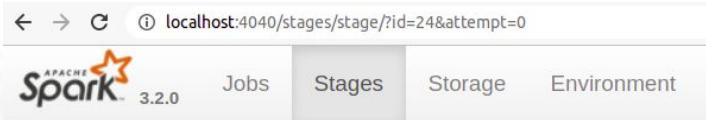
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
54928	pcalais	20	0	43352	32440	5216	R	94,4	0,2	2:11.06	python3
54939	pcalais	20	0	43352	32444	5216	R	92,7	0,2	2:15.10	python3
54941	pcalais	20	0	43344	32432	5216	R	92,7	0,2	2:14.75	python3
54940	pcalais	20	0	43352	32444	5216	R	89,0	0,2	2:13.25	python3
54929	pcalais	20	0	35672	24776	5216	R	87,7	0,2	2:14.95	python3
54938	pcalais	20	0	43604	32696	5216	R	87,0	0,2	2:14.30	python3
54934	pcalais	20	0	43344	32436	5216	R	84,1	0,2	2:13.57	python3
54936	pcalais	20	0	43356	32444	5216	R	83,7	0,2	2:13.37	python3

Spark UI: localhost:4040/stages



Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
24	collect at /tmp/ipykernel_133766/2325597050.py:20	+details	2021/10/24 18:40:31	0,2 s	32/32		20.9 KiB	
23	reduceByKey at /tmp/ipykernel_133766/2325597050.py:17	+details	2021/10/24 18:33:49	6,7 min	32/32	3.9 GiB		20.9 KiB

Spark DAG



Details for Stage 24 (Attempt 0)

Resource Profile Id: 0

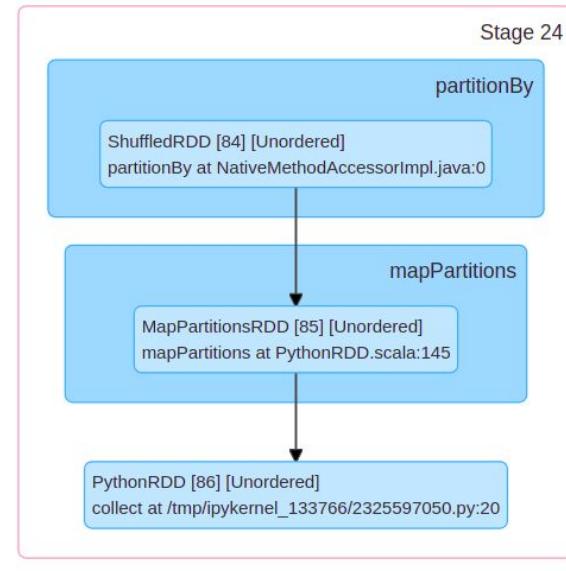
Total Time Across All Tasks: 1 s

Locality Level Summary: Node local: 9; Process local: 23

Shuffle Read Size / Records: 20.9 KiB / 288

Associated Job Ids: 22

DAG Visualization



Spark UI: localhost:4040/environment



APACHE Spark 3.2.0 Jobs Stages Storage Environment Executors SQL

Environment

Runtime Information

Name	Value
Java Home	/usr/lib/jvm/java-8-openjdk-amd64/jre
Java Version	1.8.0_292 (Private Build)
Scala Version	version 2.12.15

Spark Properties

Name	Value
spark.app.id	local-1635085714002
spark.app.name	SquareFunction
spark.app.startTime	1635085713117
spark.driver.host	192.168.0.11
spark.driver.port	42599
spark.executor.id	driver
spark.master	local[*]
spark.rdd.compress	True
spark.scheduler.mode	FIFO
spark.serializer.objectStreamReset	100
spark.submit.deployMode	client

Recapitulando

- Baixamos o Spark.
- Instalamos o Spark no modo “standalone”.
- Entramos no spark-shell.
- Executamos código de forma interativa.

Na próxima aula

Operações no Spark: transformações e ações

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

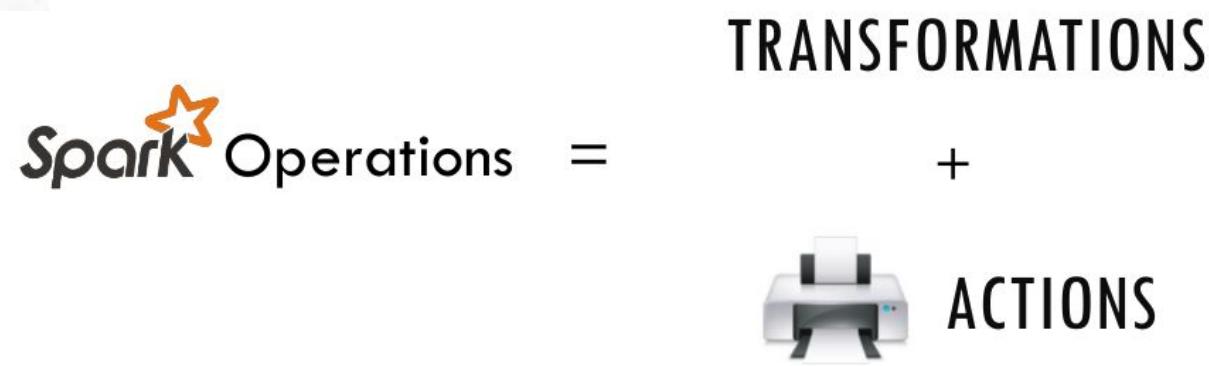
Utilizando Spark

Aula 2.4. Transformações e ações

Prof. Pedro Calais

Operações sobre RDDs

- Transformações.
- Ações.



Transformações

- RDDs são imutáveis!
- Uma transformação é uma função aplicada sobre um RDD que produz um ou mais RDDs.
- Transformações são preguiçosas (*lazy*).

map(func)

- Cria uma novo RDD passando cada elemento do RDD de origem pela função *func*.
- Cada elemento de entrada no RDD de origem é mapeado em único item no novo RDD.

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import avg
3
4 spark = (SparkSession.builder
5         .appName("Map")
6         .getOrCreate())
7
8 data = spark.sparkContext.textFile("/opt/spark/README.md")
9 mapFile = data.map(lambda line : (line, len(line)))
10 mapFile.foreach(print)
11
```

```
('# Apache Spark', 14)
('', 0)
('Spark is a unified analytics engine for large-scale data processing. It provides', 80)
('high-level APIs in Scala, Java, Python, and R, and an optimized engine that', 75)
('supports general computation graphs for data analysis. It also supports a', 73)
('rich set of higher-level tools including Spark SQL for SQL and DataFrames,', 74)
('MLlib for machine learning, GraphX for graph processing,', 56)
('and Structured Streaming for stream processing.', 47)
('', 0)
('<https://spark.apache.org/>', 27)
('', 0)
```

flatMap(func)

- Similar ao *map*, mas cada item do RDD original pode ser mapeado em zero ou mais itens no novo RDD.

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import avg
3
4 spark = (SparkSession.builder
5         .appName("Map")
6         .getOrCreate())
7
8 data = spark.sparkContext.textFile("/opt/spark/README.md")
9 flatFile = data.flatMap(lambda line: line.split())
10 flatFile.foreach(print)
```

```
#  
Apache  
Spark  
Spark  
is  
a  
unified  
analytics  
engine  
for  
large-scale  
data  
processing.  
It  
provides
```

filter(func)

- Cria um novo RDD que contém apenas os itens que foram retornados verdadeiros, de acordo com a função *func*.

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import avg
3
4 spark = (SparkSession.builder
5         .appName("Map")
6         .getOrCreate())
7
8 data = spark.sparkContext.textFile("/opt/spark/README.md")
9 flatFile = (data
10     .flatMap(lambda line: line.split())
11     .filter(lambda word: word.startswith("a")))
12 flatFile.foreach(print)
```

```
a
analytics
and
and
an
analysis.
also
a
and
and
a
and
```

reduceByKey(func)

- Os pares (K, V) que têm a mesma chave são agregados e combinados.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2
3 rdd = spark.sparkContext.parallelize(list)
4 rdd2 = (rdd.map(lambda w : (w, 1))
5         .reduceByKey(lambda a,b: a+b))
6
7 rdd2.foreach(print)
```

```
('um', 2)
('dois', 2)
('tres', 1)
```

sortByKey(func)

- Ordena as tuplas (K, V) de acordo com a chave.

```
: 1 list = ["um", "um", "dois", "dois", "tres"]
  2
  3 rdd = spark.sparkContext.parallelize(list)
  4 rdd2 = (rdd.map(lambda w : (w, 1))
  5         .reduceByKey(lambda a,b: a+b)
  6         .sortByKey("asc"))
  7
  8 rdd2.foreach(print)
```

```
('dois', 2)
('tres', 1)
('um', 2)
```

union(rdd)

- Cria um RDD que contém todos os elementos de ambos os RDDs.
- Os objetos dos RDDs devem ser do mesmo tipo.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["quatro", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddUnion = rdd.union(rdd2)
8 rddUnion.foreach(print)
9
```

```
um
tres
dois
dois
um
quatro
cinco
```

intersection(rdd)

- Cria um RDD com elementos comuns entre os dois RDDs.

```
: 1 list = ["um", "um", "dois", "dois", "tres"]
 2 list2 = ["um", "cinco"]
 3
 4 rdd = spark.sparkContext.parallelize(list)
 5 rdd2 = spark.sparkContext.parallelize(list2)
 6
 7 rddIntersection = rdd.intersection(rdd2)
 8 rddIntersection.foreach(print)
```

um

distinct(rdd)

- Retira duplicações.

```
: 1 list = ["um", "um", "dois", "dois", "tres"]
  2
  3 rdd = spark.sparkContext.parallelize(list)
  4
  5 rddDistinct = rdd.distinct()
  6 rddDistinct.foreach(print)
```

tres
dois
um

join(rdd)

- Análogo à operação típica em bancos de dados.

```
: 1 list = [("Pedro", 38), ("Maria", 21)]
  2 list2 = [("Pedro", "BH"), ("Maria", "SP")]
  3
  4 rdd = spark.sparkContext.parallelize(list)
  5 rdd2 = spark.sparkContext.parallelize(list2)
  6
  7 rddJoin = rdd.join(rdd2)
  8 rddJoin.foreach(print)
```

```
('Pedro', (38, 'BH'))
('Maria', (21, 'SP'))
```

Ações

- Ações são operações que produzem valores que não são RDDs.
- Os valores produzidos são copiados para o *driver* ou para o sistema de armazenamento.
- As ações tiram as transformações da “preguiça”!

foreach(func)

- Executa uma operação sobre cada elemento do RDD, porém sem retornar nada para o *driver*.
- Imprimir um item na tela ou escrevê-lo em um banco de dados

```
1 list = [("Pedro", 38), ("Maria", 21)]
2 list2 = [("Pedro", "BH"), ("Maria", "SP")]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddJoin = rdd.join(rdd2)
8 # rddJoin.foreach(print)
```

collect()

- Retorna todo o conteúdo do RDD para o *driver*, em memória.
- É bom que você tenha espaço em memória :-)

```
: 1 list = [("Pedro", 38), ("Maria", 21)]
 2 list2 = [("Pedro", "BH"), ("Maria", "SP")]
 3
 4 rdd = spark.sparkContext.parallelize(list)
 5 rdd2 = spark.sparkContext.parallelize(list2)
 6
 7 rddJoin = rdd.join(rdd2)
 8 print(rddJoin.collect())
[(('Maria', (21, 'SP')), ('Pedro', (38, 'BH')))]
```

count()

- Conta o número de itens do RDD.

```
: 1 list = [("Pedro", 38), ("Maria", 21)]
 2 list2 = [("Pedro", "BH"), ("Maria", "SP")]
 3
 4 rdd = spark.sparkContext.parallelize(list)
 5 rdd2 = spark.sparkContext.parallelize(list2)
 6
 7 rddJoin = rdd.join(rdd2)
 8 print(rddJoin.count())
```

take(n)

- Retorna N elementos aleatórios do RDD.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["um", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddIntersection = rdd.intersection(rdd2)
8 rddIntersection.take(1)
```

```
['um']
```

top(k)

- Ações são operações que produzem valores que não são RDDs.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["um", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddIntersection = rdd.union(rdd2)
8 rddIntersection.top(3)
```

```
['um', 'um', 'um']
```

countByValue()

- Ações são operações que produzem valores que não são RDDs.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["um", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddIntersection = rdd.union(rdd2)
8 rddIntersection.countByValue()
```

```
: defaultdict(int, {'um': 3, 'dois': 2, 'tres': 1, 'cinco': 1})
```

reduce(func)

- Combina dois elementos de um RDD e produz um único elemento.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["um", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddIntersection = rdd.union(rdd2)
8 rddIntersection.reduce(lambda a,b: a + ' ' + b)
```

'um um dois dois tres um cinco'

saveAsTextFile(path)

- Grava o conteúdo do RDD em uma pasta.

```
1 list = ["um", "um", "dois", "dois", "tres"]
2 list2 = ["um", "cinco"]
3
4 rdd = spark.sparkContext.parallelize(list)
5 rdd2 = spark.sparkContext.parallelize(list2)
6
7 rddIntersection = rdd.union(rdd2)
8 rddIntersection.saveAsTextFile('/var/tmp/spark-bootcamp')
```

saveAsTextFile(path)

- Grava o conteúdo do RDD em uma pasta.

```
pcalais:/var/tmp$ ls -lhrt spark-bootcamp/
total 28K
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00005
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00002
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00000
-rw-r--r-- 1 pcalais pcalais 5 out 24 10:14 part-00007
-rw-r--r-- 1 pcalais pcalais 5 out 24 10:14 part-00006
-rw-r--r-- 1 pcalais pcalais 5 out 24 10:14 part-00004
-rw-r--r-- 1 pcalais pcalais 3 out 24 10:14 part-00003
-rw-r--r-- 1 pcalais pcalais 3 out 24 10:14 part-00001
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00010
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00014
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00013
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00012
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00009
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 part-00008
-rw-r--r-- 1 pcalais pcalais 6 out 24 10:14 part-00015
-rw-r--r-- 1 pcalais pcalais 3 out 24 10:14 part-00011
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:14 _SUCCESS
```

transformação *repartition*

- Altera o número de partições do RDD.

```
: 1 list = ["um", "um", "dois", "dois", "tres"]
 2 list2 = ["um", "cinco"]
 3
 4 rdd = spark.sparkContext.parallelize(list)
 5 rdd2 = spark.sparkContext.parallelize(list2)
 6
 7 rddIntersection = rdd.union(rdd2)
 8 rddIntersection.repartition(1).saveAsTextFile('/var/tmp/spark-bootcamp-one-partition')
```

transformação *repartition*

- Altera o número de partições do RDD.

```
pcalais:/var/tmp$ ls -lhrt spark-bootcamp-one-partition
total 4,0K
-rw-r--r-- 1 pcalais pcalais 0 out 24 10:17 _SUCCESS
-rw-r--r-- 1 pcalais pcalais 30 out 24 10:17 part-00000
pcalais:/var/tmp$ █
```

Recapitulando

- Uma aplicação Spark é um conjunto de operações:
transformações e ações

Na próxima aula

Operações do Spark e desempenho

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

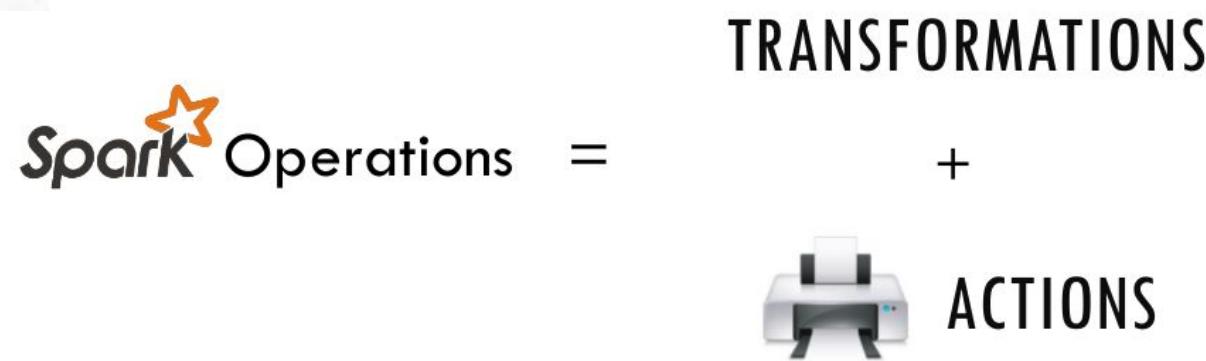
Utilizando Spark

Aula 2.5. Desempenho de operações no Spark

Prof. Pedro Calais

Operações sobre RDDs

- Transformações.
- Ações.



Operações sobre RDDs

- Transformações.
- Ações.

```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder
4         .appName("CountOnes")
5         .master("local[*]")\
6         .getOrCreate())
7
8 data = spark.sparkContext.textFile("/home/pcalais/IGTI/bootcamp-clientista-dados/numbers.txt")
9 ones = (data
10        .flatMap(lambda line: line.split())
11        .filter(lambda number: int(number) == 1))
12
13 print("Counting numbers...")
14 print(ones.count())
15 print("Counting numbers again...")
16 print(ones.count())
17
```

Operações sobre RDDs

localhost:4040/jobs

Job Id ▼	Description	Submitted	Duration
5	count at /tmp/ipykernel_133766/2355173990.py:14 count at /tmp/ipykernel_133766/2355173990.py:14	2021/10/24 11:33:21	2,9 min
4	count at /tmp/ipykernel_133766/2355173990.py:13 count at /tmp/ipykernel_133766/2355173990.py:13	2021/10/24 11:30:28	2,9 min

Operações sobre RDDs

- Quando uma ação é completada, os RDDs utilizados na computação do resultado são descartados.
- Podemos guardar resultados intermediários usando:
 - .cache().
 - .persist().

Usando .cache()

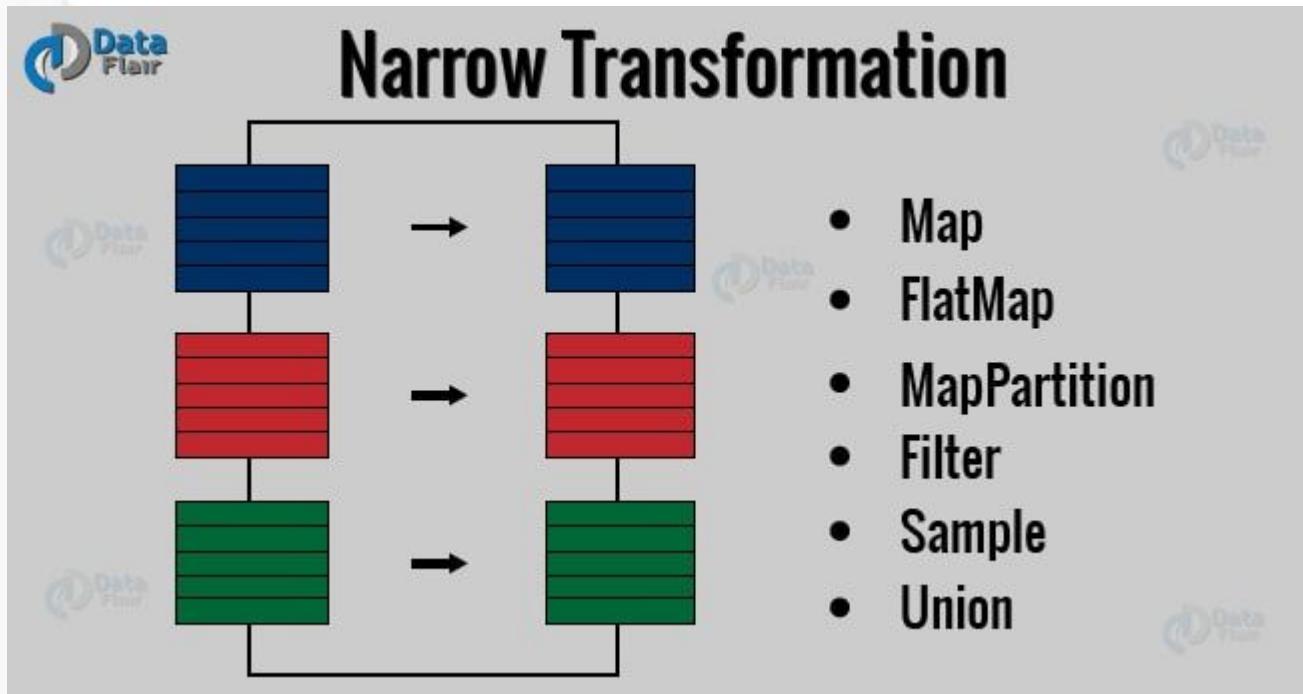
```
1 from pyspark.sql import SparkSession
2
3 spark = (SparkSession.builder
4     .appName("CountOnes")
5     .master("local[*]")\
6     .getOrCreate())
7
8 data = spark.sparkContext.textFile("/home/pcalais/IGTI/bootcamp-cientista-dados/numbers.txt")
9 ones = (data
10    .flatMap(lambda line: line.split())
11    .filter(lambda number: int(number) == 1))
12
13 print("Counting numbers...")
14 ones.cache()
15 print(ones.count())
16 print(ones.is_cached)
17 print("Counting numbers again...")
18 print(ones.count())
19
```

Usando .cache()

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total
21	count at /tmp/ipykernel_133766/1158868817.py:18 count at /tmp/ipykernel_133766/1158868817.py:18	2021/10/24 17:52:21	3 s	1/1
20	count at /tmp/ipykernel_133766/1158868817.py:15 count at /tmp/ipykernel_133766/1158868817.py:15	2021/10/24 17:49:09	3,2 min	1/1

Transformações *narrow* e *wide*

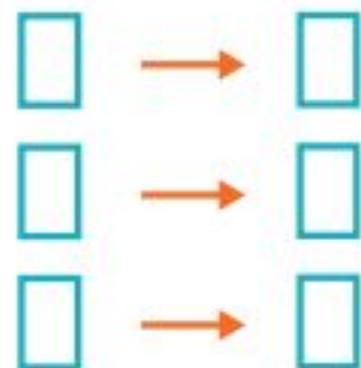
- Transformações *narrow*: todos os itens do RDD necessários para computar os novos itens da partição, estão na mesma partição.



Transformações *narrow* e *wide*

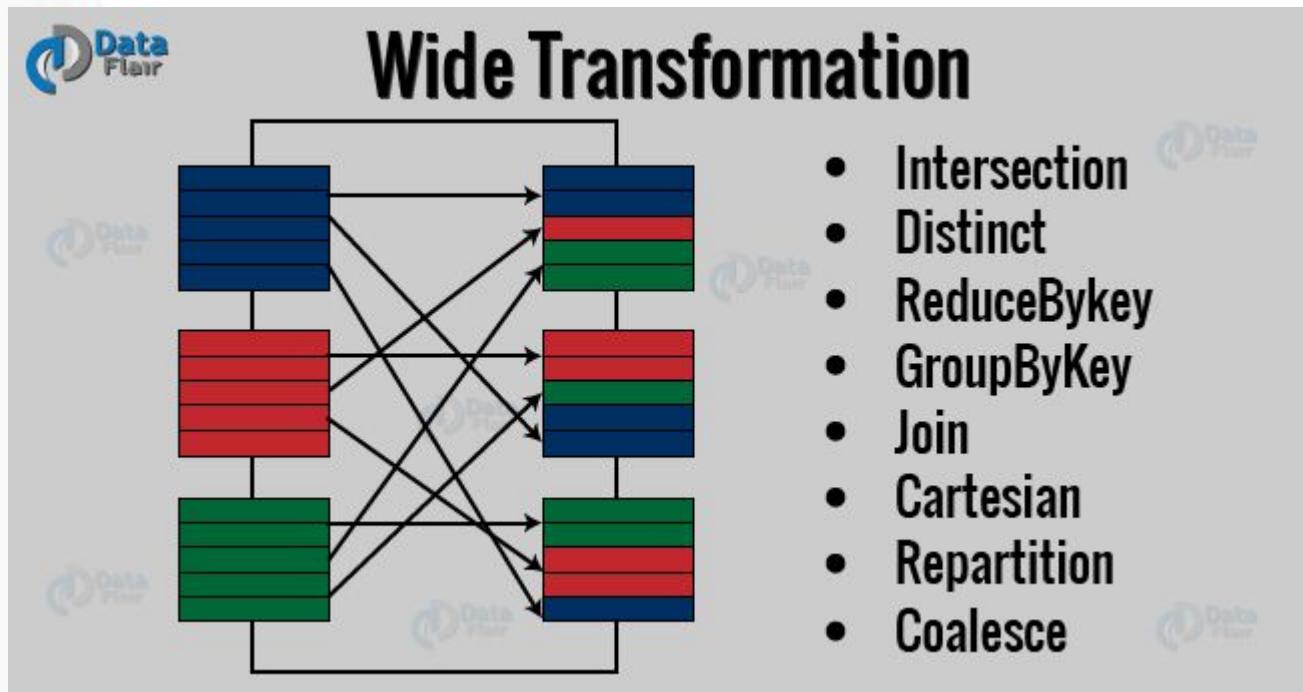
- Transformações *narrow*: todos os itens do RDD necessários para computar os novos itens da partição, estão na mesma partição.

Narrow Transformations
1 to 1



Transformações *wide*

- Transformações *wide*: os elementos necessários para computar os itens de uma partição podem viver em outras partições do RDD pai.



Transformações wide

- *shuffle*: spark precisa deslocar partições entre os executores.

Wide Transformations (shuffles)
1 to N



Recapitulando

- `.cache()` e `.persist()` evita recomputações necessárias.
- Algumas transformações exigem movimento de dados entre os nós do cluster e são mais caras.

Na próxima aula

O que é estatística descritiva?

Desenvolvimento de Soluções Utilizando Spark

Capítulo 2. Conceitos Fundamentais do Apache Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 2.6. A API Dataframe

Prof. Pedro Calais

RDDs: abstração poderosa, mas...

- O RDD é uma abstração simples e poderosa;
- Uma coleção de objetos que é particionada em executores e pode executar computações sobre seus objetos de forma distribuída e paralela;
- API bastante genérica. que trata os objetos de forma “opaca”;
- Em um RDD, o Spark não sabe que o programador está acessando uma coluna de um objeto, por exemplo:
 - Impede otimizações;
 - Torna o código mais difícil de entender.

Código complexo com RDDs



In [16]:

```
1 # Aggregate all ages by name and get the average name by age.  
2  
3 # Create an RDD of tuples (name, age)  
4 dataRDD = spark.sparkContext.parallelize([("Pedro", 38), ("Maria", 20), ("Pedro", 40), ("Rafael", 10)])  
5  
6 agesRDD = (dataRDD  
7         .map(lambda x: (x[0], (x[1], 1)))  
8         .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))  
9         .map(lambda x: (x[0], x[1][0]/x[1][1])))  
10  
11 agesRDD.collect()
```

Out[16]: [('Pedro', 39.0), ('Maria', 20.0), ('Rafael', 10.0)]

A API DataFrame

- Introduzida no Spark 2.x;
- Assim como o RDD, é uma coleção distribuída de dados;
- No entanto, permite que o programa lide com colunas nomeadas.
- Permite que o programador execute transformações típicas de análise de dados:
 - filtro
 - seleção
 - contagem
 - agregação

Por colunas!

A API DataFrame



```
: 1 from pyspark.sql import SparkSession
: 2 from pyspark.sql.functions import avg
: 3
: 4 spark = (SparkSession.builder
:           .appName("Ages")
:           .getOrCreate())
: 5
: 6 # Create a DataFrame
: 7 data_df = spark.createDataFrame([('Pedro', 38), ('Maria', 20), ('Pedro', 40), ('Rafael', 10)], ["nome", "idade"])
: 8
: 9 avg_df = data_df.groupBy("nome").agg(avg("idade"))
:10
:11 avg_df.show()
:12
:13
```

```
+-----+
| nome|avg(idade)|
+-----+
| Pedro|      39.0|
| Maria|      20.0|
| Rafael|     10.0|
+-----+
```

A API Dataframe

- Parecido com o DataFrame do Python (Pandas) e R;
- Funciona no mundo de Big Data!

DataFrame ~ tabela relacional

Direction	Year	Date	Weekday	Country	Commodity	Transport_Mode	Measure	Value	Cumulative
Exports	2015	01/01/2015	Thursday	All	All	All	\$	104000000	104000000
Exports	2015	02/01/2015	Friday	All	All	All	\$	96000000	200000000
Exports	2015	03/01/2015	Saturday	All	All	All	\$	61000000	262000000
Exports	2015	04/01/2015	Sunday	All	All	All	\$	74000000	336000000
Exports	2015	05/01/2015	Monday	All	All	All	\$	105000000	442000000

only showing top 5 rows

DataFrame tem esquema!

```
root
|--- Direction: string (nullable = true)
|--- Year: string (nullable = true)
|--- Date: string (nullable = true)
|--- Weekday: string (nullable = true)
|--- Country: string (nullable = true)
|--- Commodity: string (nullable = true)
|--- Transport_Mode: string (nullable = true)
|--- Measure: string (nullable = true)
|--- Value: string (nullable = true)
|--- Cumulative: string (nullable = true)
```

A API DataFrame

```
# Create a new DataFrame that contains "young users" only
young = users.filter(users.age < 21)

# Alternatively, using Pandas-like syntax
young = users[users.age < 21]

# Increment everybody's age by 1
young.select(young.name, young.age + 1)

# Count the number of young users by gender
young.groupBy("gender").count()

# Join young users with another DataFrame called logs
young.join(logs, logs.userId == users.userId, "left_outer")
```

DataFrames suportam SQL!



```
young.registerTempTable("young")
context.sql("SELECT count(*) FROM young")
```

Conversão para DataFrame do Pandas

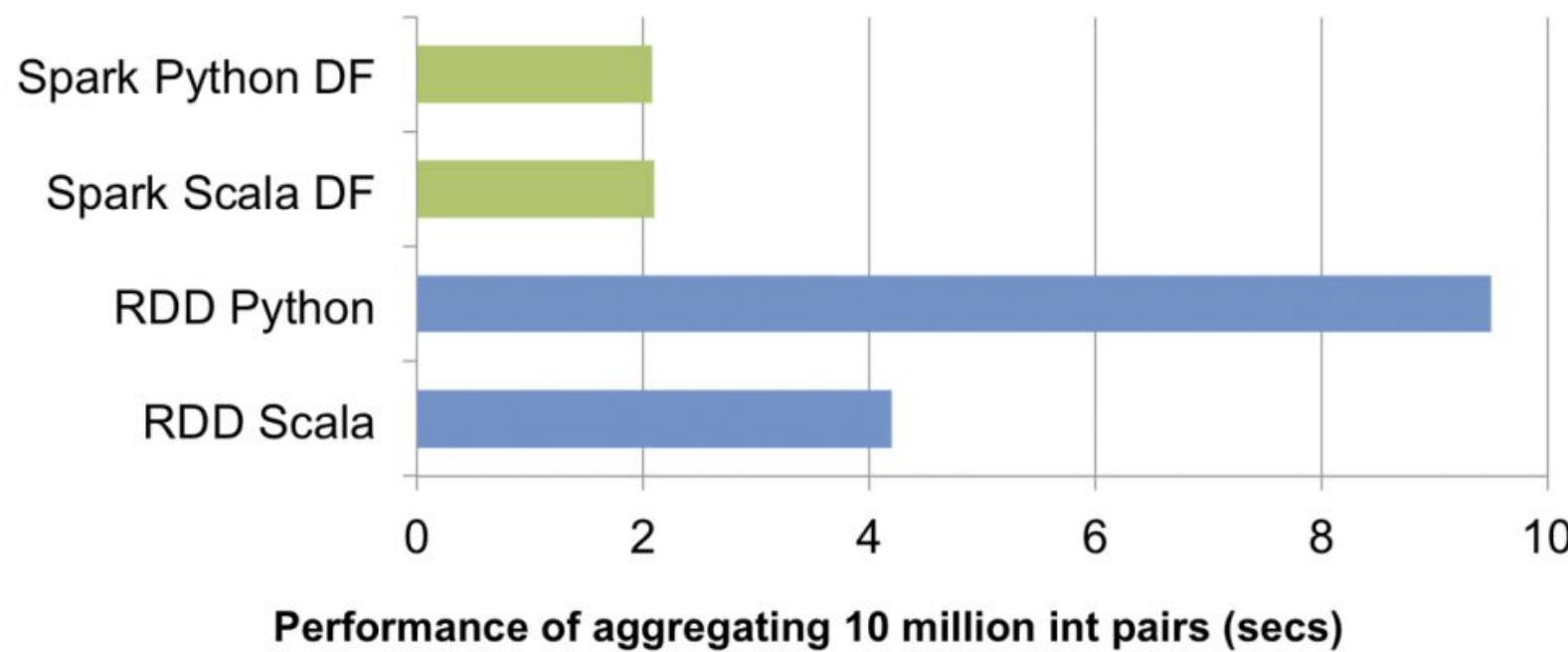


```
# Convert Spark DataFrame to Pandas
pandas_df = young.toPandas()

# Create a Spark DataFrame from Pandas
spark_df = context.createDataFrame(pandas_df)
```

Desempenho

- Dataframe conhece mais sobre o que o programador quer fazer que o RDD
 - conhece as colunas
 - conhece as operações sobre as colunas



DataFrame lê dados de várias fontes



{ JSON }



and more ...

Formats and Sources supported by DataFrames

Combinando dados de várias fontes



```
users = context.jdbc("jdbc:postgresql:production", "users")
logs = context.load("/path/to/traffic.log")
logs.join(users, logs.userId == users.userId, "left_outer") \
    .groupBy("userId").agg({"*": "count"})
```

Recapitulando

- A API DataFrame é uma alternativa à API de RDDs
- Coleção distribuída de objetos que têm colunas nomeadas

Desenvolvimento de Soluções Utilizando Spark

Capítulo 3. Estatística Descritiva com Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 3.1. O que é estatística descritiva?

Prof. Pedro Calais

O que é estatística descritiva?

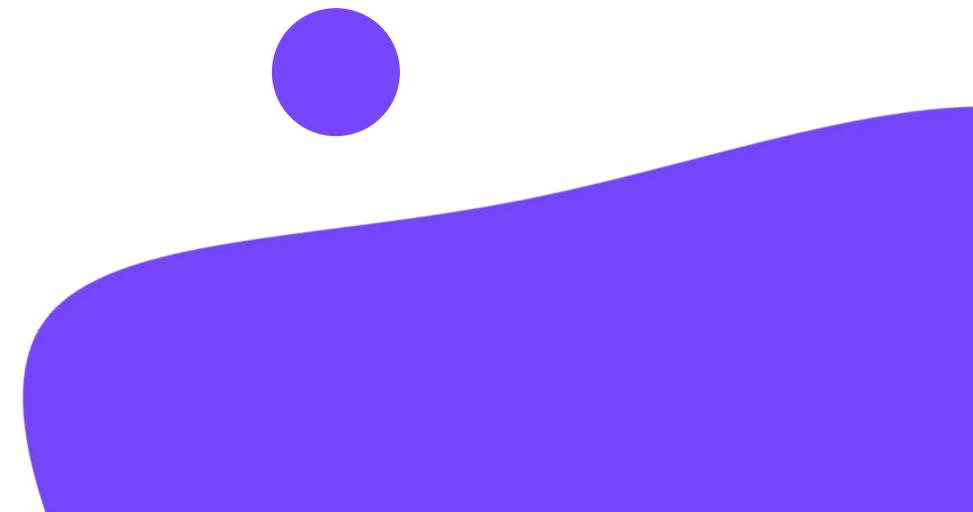
- Sumariza as características e atributos de um conjunto de dados;
- Descreve uma população ou amostra;
 - Medidas de tendência central;
 - Medidas de variabilidade;
 - Distribuição dos dados.



Estatística descritiva vs

Estatística inferencial

- Estatística inferencial busca tirar conclusões que estão além dos dados;
- Estatística generaliza para população a partir da amostra;
- Realiza previsões sobre o futuro;
- Teste de hipótese; análise de regressão, intervalo de confiança.



Medidas de tendência central

- Valores típicos centrais:
 - Média;
 - Mediana;
 - Moda.

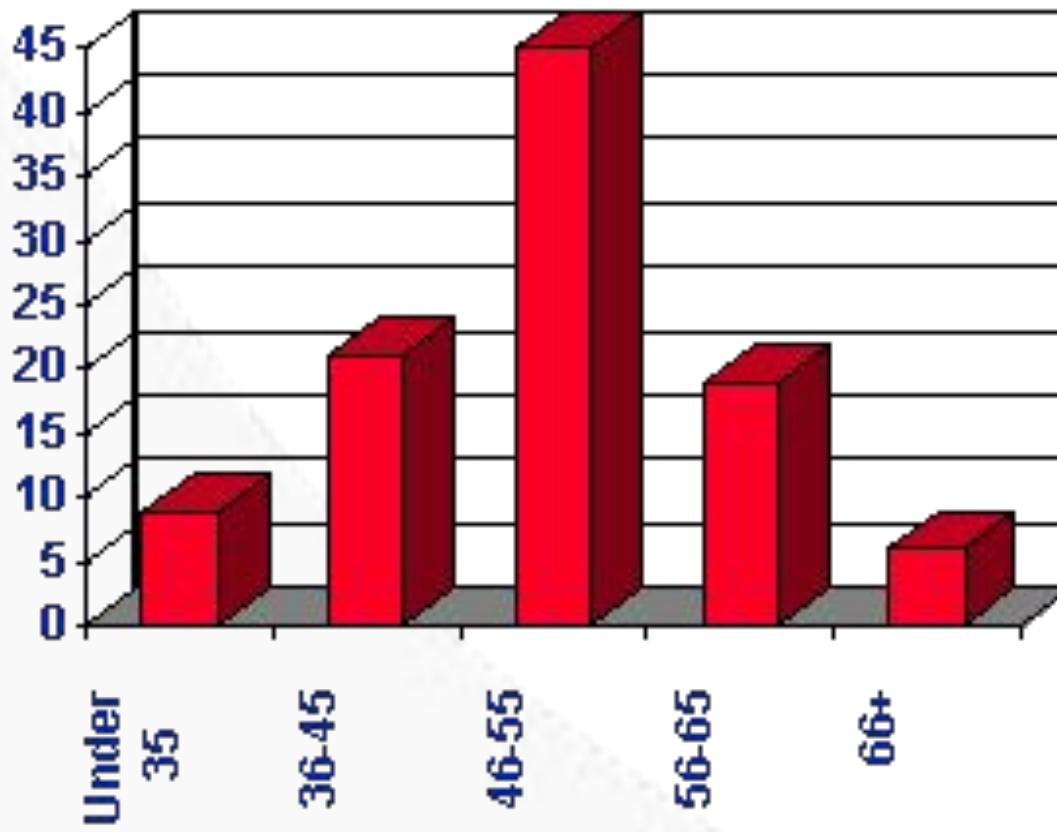


Medidas de variabilidade

- Valores típicos centrais:
 - Variância;
 - Desvio-padrão;
 - Variância;
 - Intervalo [min - max]
 - Percentil 25, 50, 75.



Distribuição de frequência



Na próxima aula

Computando Estatística Descritiva com Spark

Desenvolvimento de Soluções Utilizando Spark

Capítulo 3. Estatística Descritiva com Spark

Prof. Pedro Calais



Desenvolvimento de Soluções

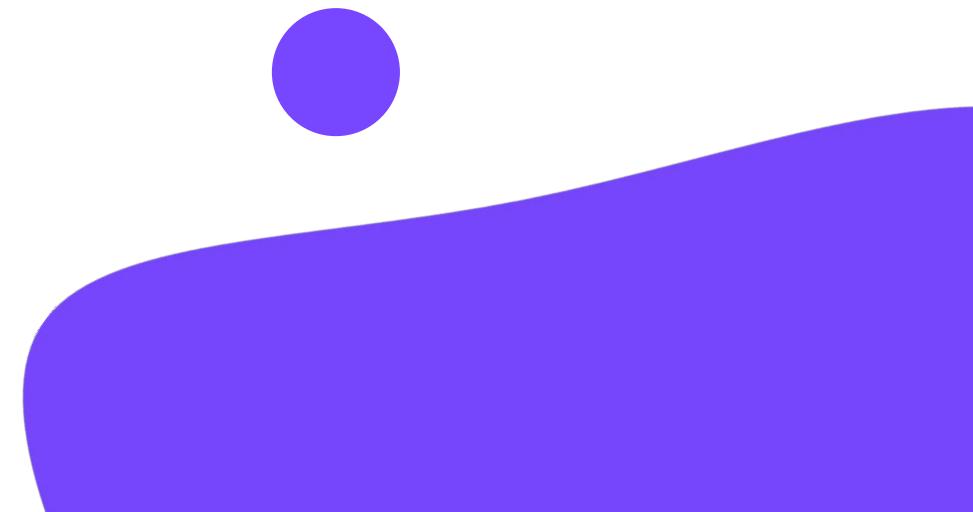
Utilizando Spark

Aula 3.2. Estatística Descritiva com Spark e Dataframes

Prof. Pedro Calais

O que é estatística descritiva?

- Sumariza as características e atributos de um conjunto de dados;
- Descreve uma população ou amostra;
 - Medidas de tendência central;
 - Medidas de variabilidade;
 - Distribuição dos dados.
- Estatística faz parte do dia a dia do cientista de dados!



Dataframes

- Inspirado no Dataframe do Python e R
- Oferece:
 - Geração de dados aleatórios
 - Diversas funções matemáticas
 - Estatística descritiva
 - Cálculo de covariância e correlação
 - Tabelas de contingência
 - Itens frequentes



Geração de dados aleatórios



```
1 from pyspark.sql.functions import rand, randn
2 from pyspark import SparkContext
3 from pyspark.sql import SQLContext
4
5 spark = SparkSession\
6     .builder\
7     .appName("PythonWordCount")\
8     .getOrCreate()
9
10 sqlc = SQLContext(spark.sparkContext)
11
12 df = (sqlc.range(0, 1000 * 1000)
13     .withColumn('uniform', rand(seed=10))
14     .withColumn('normal', randn(seed=27)))
15
16 print('# rows: ', df.count())
17 df.show()
18
```

Geração de dados aleatórios



```
# rows: 1000000
+-----+-----+
| id |      uniform |      normal |
+-----+-----+
| 0 | 0.1709497137955568 | -0.8664700627108758 |
| 1 | 0.8051143958005459 | -0.5970491018333267 |
| 2 | 0.5775925576589018 | 0.18267161219540898 |
| 3 | 0.9476047869880925 | -1.8497305679917546 |
| 4 | 0.2093704977577 | 0.9410417279045351 |
| 5 | 0.36664222617947817 | -0.6516475674670159 |
| 6 | 0.8078688178371882 | 0.5901002135239671 |
| 7 | 0.7135143433452461 | -1.850241871360443 |
| 8 | 0.7195325566306053 | 0.09176896733073023 |
| 9 | 0.31335292311175456 | -0.38605118617831075 |
| 10 | 0.8062503712025726 | 1.2134544166783332 |
| 11 | 0.10814914646176654 | -1.0757702531630617 |
| 12 | 0.3362232980701172 | 0.04961226872064977 |
| 13 | 0.8133304803837667 | -0.768259602441542 |
| 14 | 0.47649428738170896 | 0.2911293146907403 |
| 15 | 0.524728096293865 | -0.33406080411047484 |
| 16 | 0.9701253460019921 | 1.3607097640771781 |
| 17 | 0.6232167713919952 | 0.5986772981082732 |
| 18 | 0.5089687568245219 | -0.35158579838711623 |
| 19 | 0.5467504094508642 | -0.9115825072509143 |
+-----+
only showing top 20 rows
```

Sumarizando os dados

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

summary	id	uniform	normal
count	1000000	1000000	1000000
mean	499999.5	0.4997785318606761	6.545992003465573E-4
stddev	288675.27893234405	0.2887560412263698	1.0003498848232582
min	0	2.710561290975022E-7	-4.949492960499273
max	999999	0.9999998822463074	4.474351963425938

Sumarizando os dados

```
1 df.describe('uniform', 'normal').show()
```

summary	uniform	normal
count	1000000	1000000
mean	0.4997785318606761	6.545992003465573E-4
stddev	0.2887560412263698	1.0003498848232582
min	2.710561290975022E-7	-4.949492960499273
max	0.9999998822463074	4.474351963425938

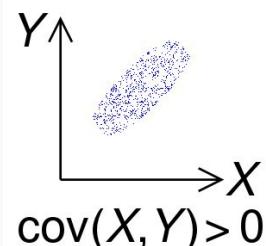
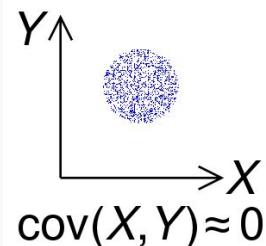
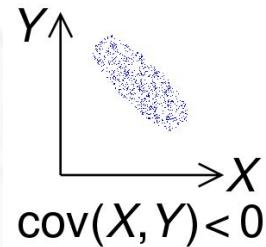
Sumarizando os dados

```
1 from pyspark.sql.functions import mean, min, max  
2  
3 df.select([mean('uniform'), min('uniform'), max('uniform')]).show()
```

avg(uniform)	min(uniform)	max(uniform)
0.4997785318606761	2.710561290975022E-7	0.999999822463074

Covariância

- medida da intensidade com a qual duas variáveis variam juntas



Covariância

- medida da intensidade com a qual duas variáveis variam juntas

```
1 from pyspark.sql.functions import rand, randn
2 from pyspark import SparkContext
3 from pyspark.sql import SQLContext
4
5
6 spark = SparkSession\
7     .builder\
8     .appName("PythonWordCount")\
9     .getOrCreate()
10
11 sqlContext = SQLContext(spark.sparkContext)
12
13 df = sqlContext.range(0, 1000 * 1000).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))
14
15 df.stat.cov('rand1', 'rand2')
```

0.00011441526053927191

Covariância

- $\text{cov}(X, X) = \text{variância}(X)$

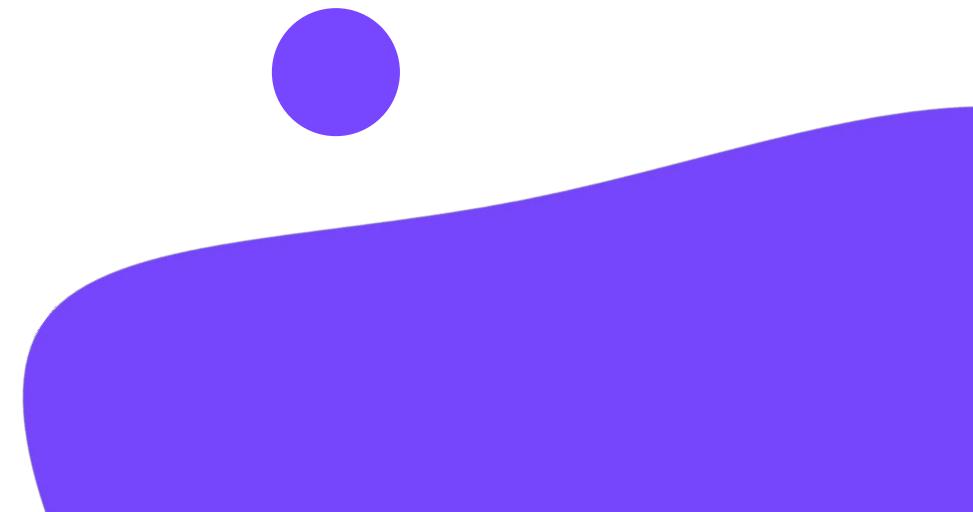
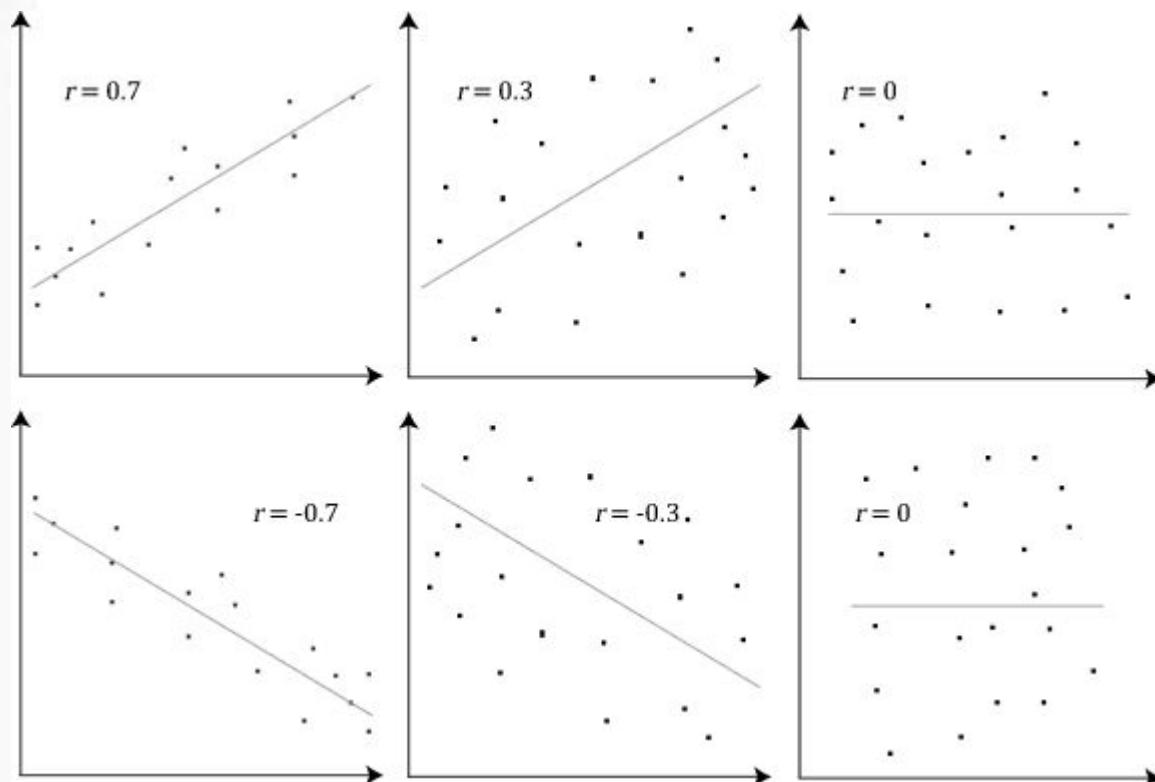
```
1 from pyspark.sql.functions import rand, randn
2 from pyspark import SparkContext
3 from pyspark.sql import SQLContext
4
5
6 spark = SparkSession\
7     .builder\
8     .appName("PythonWordCount")\
9     .getOrCreate()
10
11 sqlContext = SQLContext(spark.sparkContext)
12
13 df = sqlContext.range(0, 1000 * 1000).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))
14
15 print('cov(rand2, rand2): ', df.stat.cov('rand2', 'rand2'))
16
17 df.agg({'rand2': 'variance'}).show()
18
```

cov(rand2, rand2): 0.08339799711775965

variance(rand2)
0.08339799711775966

Correlação

- Relação estatística entre duas variáveis aleatórias
- Correlação = covariância normalizada pela variância



Correlação

- Relação estatística entre duas variáveis aleatórias

```
1 from pyspark.sql.functions import rand, randn
2 from pyspark import SparkContext
3 from pyspark.sql import SQLContext
4
5
6 spark = SparkSession\
7     .builder\
8     .appName("PythonWordCount")\
9     .getOrCreate()
10
11 sqlContext = SQLContext(spark.sparkContext)
12
13 df = sqlContext.range(0, 1000 * 1000).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))
14
15 print('cor(rand2, rand2): ', df.stat.corr('rand2', 'rand2'))
16
17 print('cor(rand1, rand2): ', df.stat.corr('rand1', 'rand2'))
```

cor(rand2, rand2): 1.0
cor(rand1, rand2): 0.00137206619523886

Tabela de Contingência

- Contagem da frequênciade pares de valores de duas colunas

```
1 names = ["Pedro", "Maria", "João"]
2 colors = ["verde", "amarelo", "rosa", "vermelho", "preto"]
3 df = sqlContext.createDataFrame([(names[i % 3], colors[i % 5]) for i in range(100)], ["name", "color"])
4
5 df.show(10)
6
7
```

```
+----+-----+
| name|  color|
+----+-----+
| Pedro|    verde|
| Maria| amarelo|
| João|     rosa|
| Pedro|vermelho|
| Maria|    preto|
| João|    verde|
| Pedro| amarelo|
| Maria|     rosa|
| João|vermelho|
| Pedro|    preto|
+----+-----+
only showing top 10 rows
```

Tabela de Contingência

- Contagem da frequênciade pares de valores de duas colunas

```
1 names = ["Pedro", "Maria", "João"]
2 colors = ["verde", "amarelo", "rosa", "vermelho", "preto"]
3 df = sqlContext.createDataFrame([(names[i % 3], colors[i % 5]) for i in range(100)], ["name", "color"])
4
5 df.show(10)
6
7
```

```
+----+-----+
| name|  color|
+----+-----+
| Pedro|    verde|
| Maria| amarelo|
| João|     rosa|
| Pedro|vermelho|
| Maria|    preto|
| João|    verde|
| Pedro| amarelo|
| Maria|     rosa|
| João|vermelho|
| Pedro|    preto|
+----+-----+
only showing top 10 rows
```

Tabela de Contingência

- Contagem da frequência de pares de valores de duas colunas

```
1 df.stat.crosstab("name", "color").show()
```

name_color	amarelo	preto	rosa	verde	vermelho
João	6	6	7	7	7
Maria	7	7	7	6	6
Pedro	7	7	6	7	7

Itens frequentes

- Quais são os valores de colunas que co-ocorrem mais frequentemente?

```
: 1 df = sqlContext.createDataFrame([(1, 2, 3) if i % 2 == 0 else (i, 2 * i, i % 4) for i in range(100)], ["a", "b", "c"])
2
3 df.show(10)

+---+---+---+
| a| b| c|
+---+---+---+
| 1| 2| 3|
| 1| 2| 1|
| 1| 2| 3|
| 3| 6| 3|
| 1| 2| 3|
| 5| 10| 1|
| 1| 2| 3|
| 7| 14| 3|
| 1| 2| 3|
| 9| 18| 1|
+---+---+---+
only showing top 10 rows
```

Itens frequentes

- Quais são os valores de colunas que co-ocorrem mais frequentemente?

```
: 1 df = sqlContext.createDataFrame([(1, 2, 3) if i % 2 == 0 else (i, 2 * i, i % 4) for i in range(100)], ["a", "b", "c"])
2
3 df.show(10)

+---+---+---+
| a| b| c|
+---+---+---+
| 1| 2| 3|
| 1| 2| 1|
| 1| 2| 3|
| 3| 6| 3|
| 1| 2| 3|
| 5| 10| 1|
| 1| 2| 3|
| 7| 14| 3|
| 1| 2| 3|
| 9| 18| 1|
+---+---+---+
only showing top 10 rows
```

Itens frequentes

- Quais são os valores de colunas que co-ocorrem mais frequentemente?

```
1 freq = df.stat.freqItems(["a", "b", "c"], 0.4)
2
3 freq.collect()[0]
```

```
Row(a_freqItems=[11, 1], b_freqItems=[2, 22], c_freqItems=[1, 3])
```

```
1 from pyspark.sql.functions import struct
2
3 freq = df.withColumn('ab', struct('a', 'b')).stat.freqItems(['ab'], 0.4)
4 freq.collect()[0]
5
```

```
Row(ab_freqItems=[Row(a=11, b=22), Row(a=1, b=2)])
```

Funções matemáticas

```
In [1]: from pyspark.sql.functions import *
```

- *sin, cos*
- *exp*
- *toDegrees, toRadians*
- *log*
- *ceil, floor*

Funções matemáticas

```
1 from pyspark.sql.functions import *
2
3 df = sqlContext.range(0, 10).withColumn('uniform', rand(seed=10) * 3.14)
4
5 df.select(
6     'uniform',
7     toDegrees('uniform'),
8     (pow(cos(df['uniform']), 2) + pow(sin(df.uniform), 2)). \
9     alias("cos^2 + sin^2")).show()
10
```

	uniform	DEGREES(uniform)	cos^2 + sin^2
1	0.5367821013180484	30.75534892368792	1.0
2	0.10747087445354876	6.157627526768682	1.0
3	1.1475525508626785	65.74991793390322	1.0
4	1.310955978808693	75.1122447131799	1.0
5	3.1083266315458262	178.09399733569154	0.9999999999999999
6	0.5165986402305565	29.598921787408106	1.0
7	0.5696528438969835	32.63870374292187	0.9999999999999999
8	1.5573024855692674	89.22685984835182	0.9999999999999999
9	3.0450071328478523	174.46605729941354	1.0
10	0.23646103537894467	13.548219346507173	1.0

Na próxima aula

Spark SQL

Desenvolvimento de Soluções Utilizando Spark

Capítulo 4. Spark SQL

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 4.1. O que é o Spark SQL?

Prof. Pedro Calais

Spark SQL



- Módulo do Apache Spark que integra o processamento de dados estruturados e relacionais com a API do Spark.
- Traz para o Spark a possibilidade dos programadores usarem programação relacional (escrevendo consultas SQL), ao mesmo tempo em que podem usar bibliotecas complexas de aprendizado de máquina.



Spark SQL usa DataFrames

IGTI

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

Data grouped into
named columns

Spark SQL

Spark SQL usa DataFrames



RDD API

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \  
    .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \  
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \  
    .collect()
```



DataFrame API

```
data.groupBy("dept").avg("age")
```

Spark SQL usa DataFrames



- Coleção distribuída de registros com o mesmo esquema.
- É como um banco relacional.
- Pode ser construído a partir de fontes externas ou de RDDs contendo objetos Java / Python.

Dataframe tem uma DSL

```
ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())
```

Operações com Dataframe

```
employees
  .join(dept, employees("deptId") === dept("id"))
  .where(employees("gender") === "female")
  .groupBy(dept("id"), dept("name"))
  .agg(count("name"))
```

RDD → Dataframe

```
case class User(name: String, age: Int)

// Create an RDD of User objects
usersRDD = spark.parallelize(
    List(User("Alice", 22), User("Bob", 19)))

// View the RDD as a DataFrame
usersDF = usersRDD.toDF
```

Suporta linguagem SQL



- Em algumas situações, o SQL “puro” é mais conveniente.

```
users.where(users("age") < 21)
    .registerTempTable("young")
ctx.sql("SELECT count(*), avg(age) FROM young")
```

Spark SQL usa DataFrames



- Transformações sobre Dataframes são otimizadas.
- Código mais simples e legível para tarefas típicas de análise de dados que trabalham com dados estruturados.

Benefícios do Spark SQL

- Integração:
 - Spark SQL “mistura” consultas SQL em programas Spark.
 - Integrar consultas SQL com *analytics* complexos.

Benefícios do Spark SQL

- Acesso unificado aos dados:
 - Lê e escreve de várias fontes e formatos de dados.
 - Arquivos Parquet, JSON, tabelas HIVE.

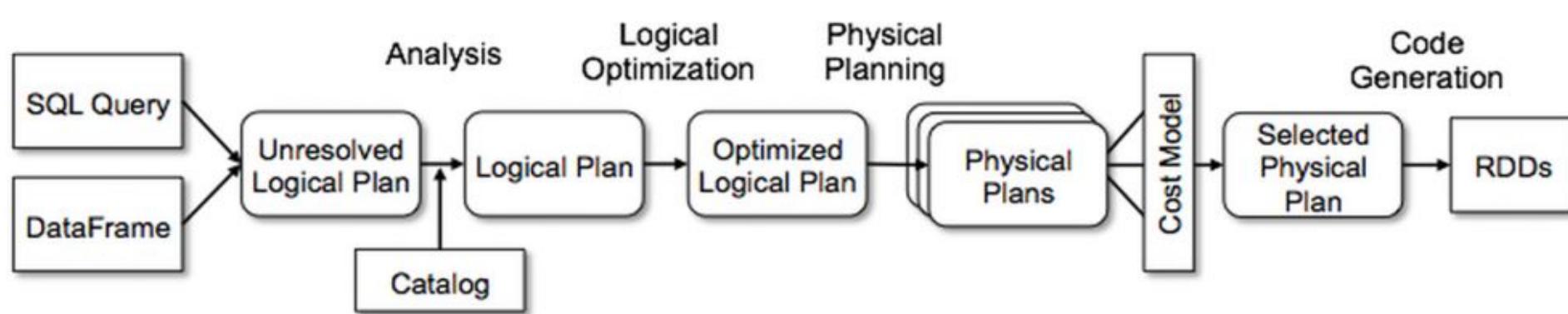
Benefícios do Spark SQL

- Alta compatibilidade com o Hive
 - Hive: armazém de dados que executa sobre o Hadoop.
 - Suporta consultas SQL.



Benefícios do Spark SQL

- Otimização de consultas:
 - Plano lógico e físico para cada consulta.
 - Catalyst Optimizer:



Artigo “oficial”



Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†],
Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin^{††}, Ali Ghodsi[†], Matei Zaharia^{†*}

[†]Databricks Inc. ^{*}MIT CSAIL [‡]AMPLab, UC Berkeley

Artigo “oficial”

ABSTRACT

Spark SQL is a new module in Apache Spark that integrates relational processing with Spark’s functional programming API. Built on our experience with Shark, Spark SQL lets Spark programmers leverage the benefits of relational processing (*e.g.*, declarative queries and optimized storage), and lets SQL users call complex analytics libraries in Spark (*e.g.*, machine learning). Compared to previous systems, Spark SQL makes two main additions. First, it offers much tighter integration between relational and procedural processing, through a declarative DataFrame API that integrates with procedural Spark code. Second, it includes a highly extensible optimizer, Catalyst, built using features of the Scala programming language, that makes it easy to add composable rules, control code generation, and define extension points. Using Catalyst, we have built a variety of features (*e.g.*, schema inference for JSON, machine learning types, and query federation to external databases) tailored for the complex needs of modern data analysis. We see Spark SQL as an evolution of both SQL-on-Spark and of Spark itself, offering richer APIs and optimizations while keeping the benefits of the Spark programming model.

Artigo “oficial”

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform advanced analytics, such as machine learning and graph processing, that are challenging to express in relational systems. In practice, we have observed that most data pipelines would ideally be expressed with a combination of both relational queries and complex procedural algorithms. Unfortunately, these two classes of systems—relational and procedural—have until now remained largely disjoint, forcing users to choose one paradigm or the other.

Artigo “oficial”



Spark SQL bridges the gap between the two models through two contributions. First, Spark SQL provides a *DataFrame API* that can perform relational operations on both external data sources and Spark’s built-in distributed collections. This API is similar to the widely used data frame concept in R [32], but evaluates operations lazily so that it can perform relational optimizations. Second, to support the wide range of data sources and algorithms in big data, Spark SQL introduces a novel extensible optimizer called *Catalyst*. Catalyst makes it easy to add data sources, optimization rules, and data types for domains such as machine learning.

Artigo “oficial”

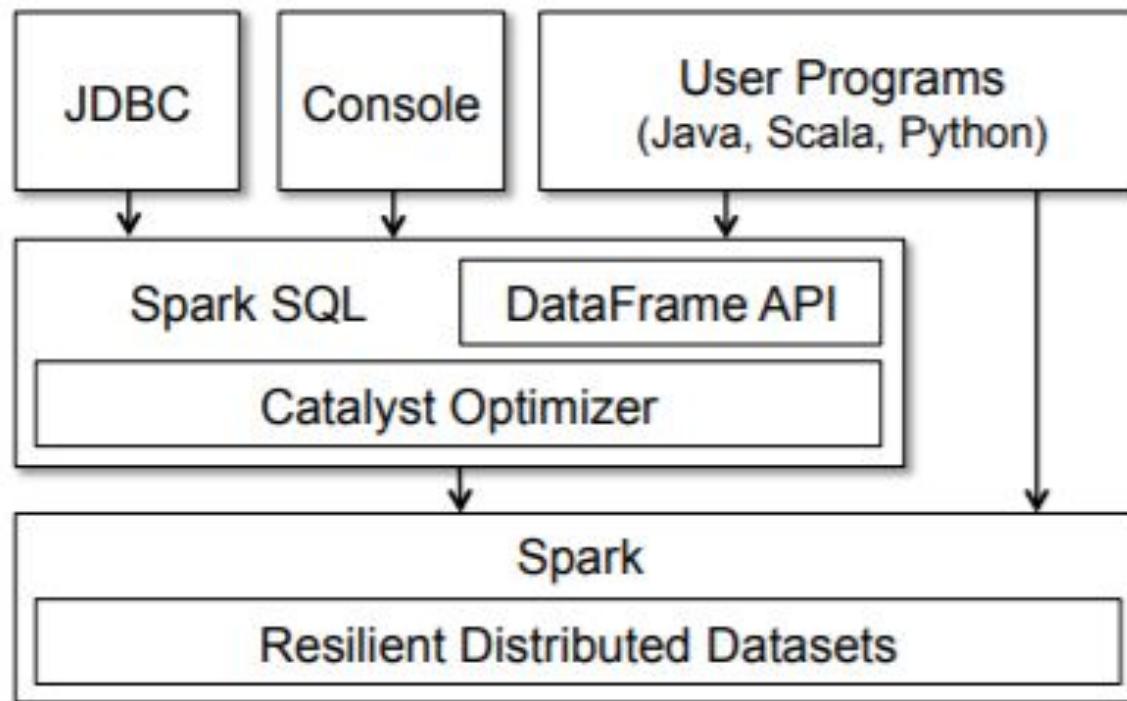
The DataFrame API offers rich relational/procedural integration within Spark programs. DataFrames are collections of structured records that can be manipulated using Spark’s procedural API, or using new relational APIs that allow richer optimizations. They can be created directly from Spark’s built-in distributed collections of Java/Python objects, enabling relational processing in existing Spark programs. Other Spark components, such as the machine learning library, take and produce DataFrames as well. DataFrames are more convenient and more efficient than Spark’s procedural API in many common situations. For example, they make it easy to compute multiple aggregates in one pass using a SQL statement, something that is difficult to express in traditional functional APIs. They also automatically store data in a columnar format that is significantly more compact than Java/Python objects. Finally, unlike existing data frame APIs in R and Python, DataFrame operations in Spark SQL go through a relational optimizer, Catalyst.

Objetivos do Spark SQL



- Suportar processamento relacional em programas Spark e lendo de fontes externas.
- Prover desempenho usando técnicas estabelecidas de banco de dados.
- Suportar novas fontes de dados com facilidade.
- Habilitar integração com aplicações de aprendizado de máquina e processamento de grafos.

Spark SQL + Spark



Na próxima aula

User Defined Functions: UDFs

Desenvolvimento de Soluções Utilizando Spark

Capítulo 4. Spark SQL

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 4.2. *User Defined Functions*

Prof. Pedro Calais

O que são UDFs?



- User-defined functions.
- Estendem a funcionalidade de sistemas de banco de dados.
- Função baseada em colunas, avaliada dentro de consultas SQL.

O que são UDFs?



- Combinação poderosa:
 - Operadores relacionais.
 - Funções analíticas complexas.

```
val model: LogisticRegressionModel = ...  
  
ctx.udf.register("predict",  
  (x: Float, y: Float) => model.predict(Vector(x, y)))  
  
ctx.sql("SELECT predict(age, weight) FROM users")
```

Mais exemplos de UDFs

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.udf

val spark = SparkSession
  .builder()
  .appName("Spark SQL UDF scalar example")
  .getOrCreate()

// Define and register a zero-argument non-deterministic UDF
// UDF is deterministic by default, i.e. produces the same result for the same input.
val random = udf(() => Math.random())
spark.udf.register("random", random.asNondeterministic())
spark.sql("SELECT random()").show()
// +-----+
// |UDF() |
// +-----+
// |xxxxxx|
// +-----+
```

Mais exemplos de UDFs

```
// Define and register a one-argument UDF
val plusOne = udf((x: Int) => x + 1)
spark.udf.register("plusOne", plusOne)
spark.sql("SELECT plusOne(5)").show()
// +---+
// |UDF(5)|
// +---+
// |      6|
// +---+
```

Mais exemplos de UDFs

```
// Define a two-argument UDF and register it with Spark in one step
spark.udf.register("strLenScala", (_: String).length + (_: Int))
spark.sql("SELECT strLenScala('test', 1)").show()
// +-----+
// |strLenScala(test, 1)|
// +-----+
// |                         5|
// +-----+
```

Mais exemplos de UDFs

```
// UDF in a WHERE clause
spark.udf.register("oneArgFilter", (n: Int) => { n > 5 })
spark.range(1, 10).createOrReplaceTempView("test")
spark.sql("SELECT * FROM test WHERE oneArgFilter(id)").show()
// +---+
// | id|
// +---+
// | 6|
// | 7|
// | 8|
// | 9|
// +---+
```

Mais exemplos de UDFs

```
val dataset = Seq((0, "hello"), (1, "world")).toDF("id", "text")

// Define a regular Scala function
val upper: String => String = _.toUpperCase

// Define a UDF that wraps the upper Scala function defined above
// You could also define the function in place, i.e. inside udf
// but separating Scala functions from Spark SQL's UDFs allows for easier testing
import org.apache.spark.sql.functions.udf
val upperUDF = udf(upper)

// Apply the UDF to change the source dataset
scala> dataset.withColumn("upper", upperUDF('text)).show
+---+-----+
| id| text|upper|
+---+-----+
|  0|hello|HELLO|
|  1|world|WORLD|
+---+-----+
```

Na próxima aula

Spark SQL e Formatos de Dados

Desenvolvimento de Soluções Utilizando Spark

Capítulo 4. Spark SQL

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 4.3. Formatos de Dados

Prof. Pedro Calais

Spark SQL suporta vários formatos de dados



- CSV.
- JSON.
- texto.
- Parquet.
- tabelas do Hive.

<https://spark.apache.org/docs/latest/sql-data-sources.html>

CSV

```
1 name;age;job
2 Jorge;30;Developer
3 Bob;32;Developer
```

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

CSV



```
// A CSV dataset is pointed to by path.  
// The path can be either a single CSV file or a directory of CSV files  
val path = "examples/src/main/resources/people.csv"  
  
val df = spark.read.csv(path)  
df.show()  
// +-----+  
// |          _c0|  
// +-----+  
// |    name;age;job|  
// |Jorge;30;Developer|  
// | Bob;32;Developer|  
// +-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

CSV



```
// Read a csv with delimiter, the default delimiter is ","
val df2 = spark.read.option("delimiter", ";").csv(path)
df2.show()
// +---+---+---+
// | _c0|_c1|    _c2|
// +---+---+---+
// | name|age|      job|
// |Jorge| 30|Developer|
// | Bob| 32|Developer|
// +---+---+---+
```

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

CSV



```
// Read a csv with delimiter and a header
val df3 = spark.read.option("delimiter", ";").option("header", "true").csv(path)
df3.show()
// +-----+
// | name|age| job|
// +-----+
// | Jorge| 30|Developer|
// | Bob| 32|Developer|
// +-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

CSV



```
// Read all files in a folder, please make sure only CSV files should present in the folder.  
val folderPath = "examples/src/main/resources";  
val df5 = spark.read.csv(folderPath);  
df5.show();  
// Wrong schema because non-CSV files are read  
// +-----+  
// |      _c0|  
// +-----+  
// |238val_238|  
// | 86val_86|  
// |311val_311|  
// | 27val_27|  
// |165val_165|  
// +-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-csv.html>

JSON



```
1  {"name": "Michael"}  
2  {"name": "Andy", "age": 30}  
3  {"name": "Justin", "age": 19}
```

<https://spark.apache.org/docs/latest/sql-data-sources-json.html>

JSON



```
// Primitive types (Int, String, etc) and Product types (case classes) encoders are
// supported by importing this when creating a Dataset.
import spark.implicits._

// A JSON dataset is pointed to by path.
// The path can be either a single text file or a directory storing text files
val path = "examples/src/main/resources/people.json"
val peopleDF = spark.read.json(path)

// The inferred schema can be visualized using the printSchema() method
peopleDF.printSchema()
// root
// |-- age: long (nullable = true)
// |-- name: string (nullable = true)
```

<https://spark.apache.org/docs/latest/sql-data-sources-json.html>

JSON



```
// Creates a temporary view using the DataFrame
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by spark
val teenagerNamesDF = spark.sql("SELECT name FROM people WHERE age BETWEEN 13 AND 19")
teenagerNamesDF.show()
+-----+
| name|
+-----+
| Justin|
+-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-json.html>

JSON



```
// Alternatively, a DataFrame can be created for a JSON dataset represented by
// a Dataset[String] storing one JSON object per string
val otherPeopleDataset = spark.createDataset(
    """>{"name":"Yin","address":{"city":"Columbus","state":"Ohio"}}""": Nil)
val otherPeople = spark.read.json(otherPeopleDataset)
otherPeople.show()
// +-----+----+
// |      address/name|
// +-----+----+
// |[Columbus,Ohio]/ Yin|
// +-----+----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-json.html>

Arquivo texto

```
1 Michael, 29
2 Andy, 30
3 Justin, 19
```

<https://spark.apache.org/docs/latest/sql-data-sources-text.html>

Arquivo texto

```
// You can use 'lineSep' option to define the line separator.  
// The line separator handles all '\r', '\r\n' and '\n' by default.  
val df2 = spark.read.option("lineSep", ",").text(path)  
df2.show()  
// +-----+  
// |      value|  
// +-----+  
// | Michael|  
// | 29\nAndy|  
// | 30\nJustin|  
// |      19\n|  
// +-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-text.html>

Arquivo texto

```
// "output" is a folder which contains multiple text files and a _SUCCESS file.  
df1.write.text("output")  
  
// You can specify the compression format using the 'compression' option.  
df1.write.option("compression", "gzip").text("output_compressed")
```

<https://spark.apache.org/docs/latest/sql-data-sources-text.html>

Parquet

IGTI



“Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language”

<https://spark.apache.org/docs/latest/sql-data-sources-parquet.html>

Parquet: armazenamento por colunas

IGTI

Row Storage

Last Name	First Name	E-mail	Phone #	Street Address

Columnar Storage

Last Name	First Name	E-mail	Phone #	Street Address

<https://spark.apache.org/docs/latest/sql-data-sources-parquet.html>

Parquet: armazenamento por colunas



- Orientado por colunas ao invés de linhas como o CSV.
- Compressão eficiente.

Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	\$5.75
Data stored in Apache Parquet format*	130 GB	6.78 seconds	2.51 GB	\$0.01
Savings / Speedup	87% less with Parquet	34x faster	99% less data scanned	99.7% savings

<https://spark.apache.org/docs/latest/sql-data-sources-parquet.html>

Parquet: armazenamento por colunas



```
val peopleDF = spark.read.json("examples/src/main/resources/people.json")

// DataFrames can be saved as Parquet files, maintaining the schema information
peopleDF.write.parquet("people.parquet")

// Read in the parquet file created above
// Parquet files are self-describing so the schema is preserved
// The result of loading a Parquet file is also a DataFrame
val parquetFileDF = spark.read.parquet("people.parquet")

// Parquet files can also be used to create a temporary view and then used in SQL statements
parquetFileDF.createOrReplaceTempView("parquetFile")
val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")
namesDF.map(attributes => "Name: " + attributes(0)).show()
// +-----+
// |      value|
// +-----+
// |Name: Justin|
// +-----+
```

<https://spark.apache.org/docs/latest/sql-data-sources-parquet.html>

Na próxima aula

Spark MLLib

Desenvolvimento de Soluções Utilizando Spark

Capítulo 5. Spark MLlib

Prof. Pedro Calais



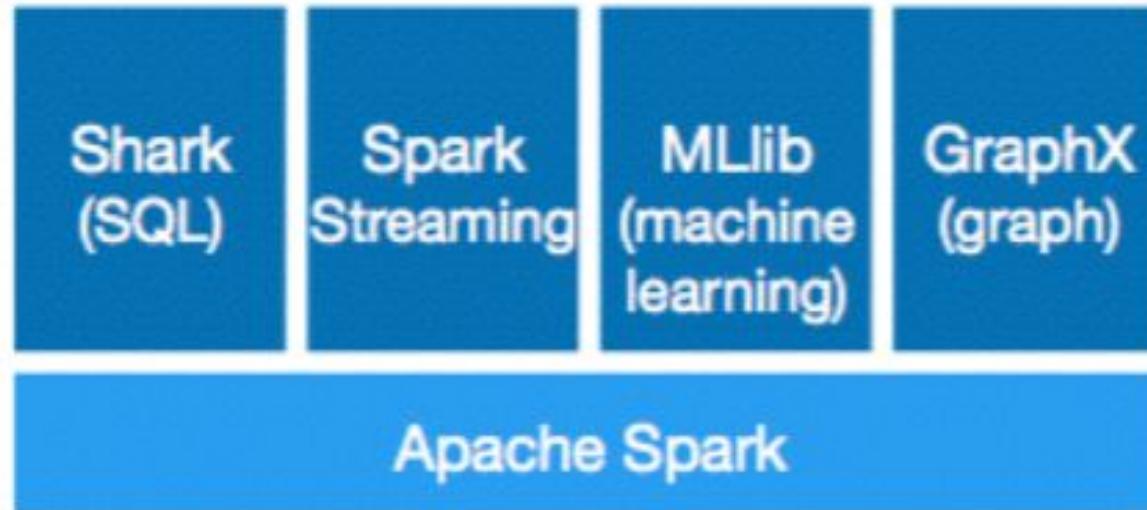
Desenvolvimento de Soluções

Utilizando Spark

Aula 5.1. O que é o Spark MLlib?

Prof. Pedro Calais

Spark MLLib



Spark MLLib



Journal of Machine Learning Research 17 (2016) 1-7

Submitted 5/15; Published 4/16

MLlib: Machine Learning in Apache Spark

Características principais



- Implementações distribuídas dos principais algoritmos.
- Integração com os outros módulos do Spark.
- Suporta criação de Pipelines de ML.

Algoritmos



- Classificação.
- Regressão.
- Filtragem Colaborativa.
- Agrupamento.

Spark SQL + Spark MLLib



```
// Data can easily be extracted from existing sources,  
// such as Apache Hive.  
val trainingTable = sql("""  
    SELECT e.action,  
        u.age,  
        u.latitude,  
        u.longitude  
    FROM Users u  
    JOIN Events e  
    ON u.userId = e.userId""")  
  
// Since 'sql' returns an RDD, the results of the above  
// query can be easily used in MLLib.  
val training = trainingTable.map { row =>  
    val features = Vectors.dense(row(1), row(2), row(3))  
    LabeledPoint(row(0), features)  
}  
  
val model = SVMWithSGD.train(training)
```

Streaming + MLLib



```
// collect tweets using streaming  
  
// train a k-means model  
val model: KMmeansModel = ...  
  
// apply model to filter tweets  
val tweets = TwitterUtils.createStream(ssc, Some(authorizations(0)))  
val statuses = tweets.map(_.getText)  
val filteredTweets =  
    statuses.filter(t => model.predict(featurize(t)) == clusterNumber)  
  
// print tweets within this particular cluster  
filteredTweets.print()
```

GraphX + MLLib

```
// assemble link graph
val graph = Graph(pages, links)
val pageRank: RDD[(Long, Double)] = graph.staticPageRank(10).vertices

// load page labels (spam or not) and content features
val labelAndFeatures: RDD[(Long, (Double, Seq((Int, Double))))] = ...
val training: RDD[LabeledPoint] =
  labelAndFeatures.join(pageRank).map {
    case (id, ((label, features), pageRank)) =>
      LabeledPoint(label, Vectors.sparse(features ++ (1000, pageRank)))
  }

// train a spam detector using logistic regression
val model = LogisticRegressionWithSGD.train(training)
```

Artigos que usam o Spark MLLib



The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks
(EUSPN 2018)

A Large-Scale Sentiment Data Classification for Online Reviews Under Apache Spark

Samar Al-Saqqa^{a,b,*}, Ghazi Al-Naymat^a, Arafat Awajan^a

^a*Princess Sumaya University for Technology, Amman, Jordan*

^b*The University of Jordan, Amman, Jordan*

Artigos que usam o Spark MLLib



Research | [Open Access](#) | [Published: 08 January 2019](#)

Large-scale e-learning recommender system based on Spark and Hadoop

[Karim Dahdouh](#) , [Ahmed Dakkak](#), [Lahcen Oughdir](#) & [Abdelali Ibriz](#)

[Journal of Big Data](#) **6**, Article number: 2 (2019) | [Cite this article](#)

Artigos que usam o Spark MLLib



(IJACSA) International Journal of Advanced Computer Science and Applications,
Vol. 9, No. 11, 2018

Predicting Potential Banking Customer Churn using Apache Spark ML and MLlib Packages: A Comparative Study

Hend Sayed, Manal A. Abdel-Fattah, Sherif Kholief
Information Systems Department
Faculty of Computers and Information
Helwan University
Cairo, Egypt

Artigos que usam o Spark MLLib



RESEARCH-ARTICLE

Breast Cancer Prediction Using Spark MLlib and ML Packages



Authors: [Phan Duy Hung](#), [Tran Duc Hanh](#), [Vu Thu Diep](#) [Authors Info & Claims](#)

Artigos que usam o Spark MLLib



Analyzing mobile phone usage using clustering in Spark MLLib and Pig.

- **Source:** International Journal of Advanced Research in Computer Science . Jan/Feb2017, Vol. 8 Issue 1, p144-146. 3p.
- **Author(s):** Arora, Shefali
- **Abstract:** K-means is a common method of clustering data points using a predefined number of clusters. Apache Spark is a computing technology used for fast computation of data. By making use of its machine learning library called MLLib, we analyze mobile data obtained from Opencellid.org by clustering according to latitude and longitude values, using K-means algorithm. Once each data point is assigned its cluster number, the dataset is loaded into Apache Pig to calculate the number of users in each cluster. Thus, we can analyse the number of users using a mobile network in a particular range of latitude and longitude.

Artigos que usam o Spark MLLib

Conferences > 2019 5th International Conference on...



Fraud Detection Using Apache Spark

Publisher: IEEE

Cite This

PDF

Abdelkbir ARMEL ; Dounia ZAIDOUNI All Authors

Abstract:

Fraud detection methods are continuously developed to defend criminals. They allow us to identify quickly and easily the frauds. In this work, we will focus on the problem of fraud detection in banking transactions. A single algorithm may not be suitable for every problem. Therefore, selecting an algorithm that performs best in a given situation is very crucial. In this work, we give a comparative analysis of four algorithms: Simple Anomaly detection algorithm, Decision Tree algorithm, Random Forest algorithm and Naïve Bayes algorithm. We use the machine learning library (MLlib) of Apache Spark to handle credit card fraud detection. The data used in our simulation is generated randomly following a normal distribution, this data includes two features Price and Distance that allow us to distinguish anomalies and valid transactions. The performance is analysed based on the parameters of the Total Running Time and the Accuracy. The results proved that the Random Forest algorithm gave the best results and the simple anomaly detection algorithm gave the worst results.

Artigos que usam o Spark MLLib



Topic modeling of news based on spark Mllib

Publisher: IEEE

[Cite This](#)

PDF

Jing Gui ; Qi Wang [All Authors](#)

Abstract:

In recent years, Spark has become one of the hottest data processing frameworks that makes up for Hadoop's flaws in implementing traditional data mining and machine learning algorithms. Based on the Spark platform, this paper deals with the news by using LDA model on MLlib to model the topics of them. The main work of this paper includes using the crawler to collect large-scale news corpus on the Internet, processing the data, and using the Scala language to call the LDA model in MLlib. The optimal number of iterations is determined first, then the optimal number of topics is estimated by the statistical method of V-fold cross validation, and finally the topic modeling of news corpus is realized. The topic probability distribution of each news obtained in the experiment is applied to the construction of the news corpus. In the Spark local mode, the experiment achieves the function of topic modeling by program coding and initial debugging.

Na próxima aula

Pipelines de ML

Desenvolvimento de Soluções Utilizando Spark

Capítulo 5. Spark MLlib

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 5.2. Pipelines de ML

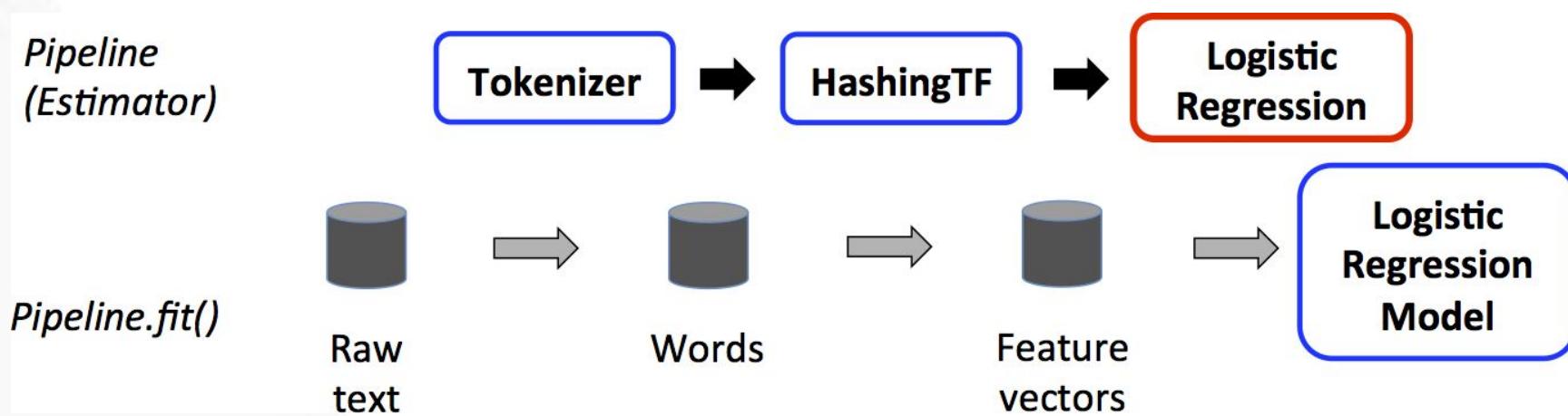
Prof. Pedro Calais

Spark MLLib



- Problemas típicos de ML envolvem um fluxo que tem várias etapas:
 - Limpeza dos dados.
 - Extração dos atributos.
 - Ajuste dos parâmetros do modelo.
 - Avaliação do modelo.
- A API spark.ml simplifica a criação de pipelines com múltiplos estágios:
 - API uniforme.
 - Possibilidade de personalizar etapas.

Exemplo típico de pipeline de ML



Exemplo típico de pipeline de ML



```
data = <DataFrame of (text, label) records>

tokenizer = Tokenizer()
    .setInputCol("text").setOutputCol("words")
tf = HashingTF()
    .setInputCol("words").setOutputCol("features")
lr = LogisticRegression()
    .setInputCol("features")

pipeline = Pipeline().setStages([tokenizer, tf, lr])
model = pipeline.fit(data)
```

Criação do Dataframe



```
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])
```

Criação do estimador

```
# Create a LogisticRegression instance. This instance is an Estimator.  
lr = LogisticRegression(maxIter=10, regParam=0.01)  
# Print out the parameters, documentation, and any default values.  
print("LogisticRegression parameters:\n" + lr.explainParams() + "\n")  
  
# Learn a LogisticRegression model. This uses the parameters stored in lr.  
model1 = lr.fit(training)
```

- Recebe um Dataframe.
- Produz um Model.

Criação do estimador

```
# Since model1 is a Model (i.e., a transformer produced by an Estimator),  
# we can view the parameters it used during fit().  
# This prints the parameter (name: value) pairs, where names are unique IDs for this  
# LogisticRegression instance.  
print("Model 1 was fit using parameters: ")  
print(model1.extractParamMap())
```

- Recebe um Dataframe.
- Produz um Model.

Criação do estimador

```
# We may alternatively specify parameters using a Python dictionary as a paramMap
paramMap = {lr.maxIter: 20}
paramMap[lr.maxIter] = 30 # Specify 1 Param, overwriting the original maxIter.
# Specify multiple Params.
paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55}) # type: ignore

# You can combine paramMaps, which are python dictionaries.
# Change output column name
paramMap2 = {lr.probabilityCol: "myProbability"} # type: ignore
paramMapCombined = paramMap.copy()
paramMapCombined.update(paramMap2) # type: ignore

# Now learn a new model using the paramMapCombined parameters.
# paramMapCombined overrides all parameters set earlier via lr.set* methods.
model2 = lr.fit(training, paramMapCombined)
print("Model 2 was fit using parameters: ")
print(model2.extractParamMap())
```

- Recebe um Dataframe.
- Produz um Model.

Teste do modelo

```
# Make predictions on test data using the Transformer.transform() method.  
# LogisticRegression.transform will only use the 'features' column.  
# Note that model2.transform() outputs a "myProbability" column instead of the usual  
# 'probability' column since we renamed the lr.probabilityCol parameter previously.  
prediction = model2.transform(test)  
result = prediction.select("features", "label", "myProbability", "prediction") \  
    .collect()  
  
for row in result:  
    print("features=%s, label=%s -> prob=%s, prediction=%s"  
        % (row.features, row.label, row.myProbability, row.prediction))
```

Teste do modelo

```
# Make predictions on test data using the Transformer.transform() method.  
# LogisticRegression.transform will only use the 'features' column.  
# Note that model2.transform() outputs a "myProbability" column instead of the usual  
# 'probability' column since we renamed the lr.probabilityCol parameter previously.  
prediction = model2.transform(test)  
result = prediction.select("features", "label", "myProbability", "prediction") \  
    .collect()  
  
for row in result:  
    print("features=%s, label=%s -> prob=%s, prediction=%s"  
        % (row.features, row.label, row.myProbability, row.prediction))
```

Exemplo de Pipeline de ML



```
# Make predictions on test data using the Transformer.transform() method.  
# LogisticRegression.transform will only use the 'features' column.  
# Note that model2.transform() outputs a "myProbability" column instead of the usual  
# 'probability' column since we renamed the lr.probabilityCol parameter previously.  
prediction = model2.transform(test)  
result = prediction.select("features", "label", "myProbability", "prediction") \  
    .collect()  
  
for row in result:  
    print("features=%s, label=%s -> prob=%s, prediction=%s"  
        % (row.features, row.label, row.myProbability, row.prediction))
```

Exemplo de Pipeline de ML

```
# Make predictions on test data using the Transformer.transform() method.  
# LogisticRegression.transform will only use the 'features' column.  
# Note that model2.transform() outputs a "myProbability" column instead of the usual  
# 'probability' column since we renamed the lr.probabilityCol parameter previously.  
prediction = model2.transform(test)  
result = prediction.select("features", "label", "myProbability", "prediction") \  
    .collect()  
  
for row in result:  
    print("features=%s, label=%s -> prob=%s, prediction=%s"  
        % (row.features, row.label, row.myProbability, row.prediction))
```

Exemplo de Pipeline de ML



```
# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.  
tokenizer = Tokenizer(inputCol="text", outputCol="words")  
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")  
lr = LogisticRegression(maxIter=10, regParam=0.001)  
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

Exemplo de Pipeline de ML



```
# Fit the pipeline to training documents.  
model = pipeline.fit(training)
```

Exemplo de Pipeline de ML

```
# Prepare test documents, which are unlabeled (id, text) tuples.
test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
for row in selected.collect():
    rid, text, prob, prediction = row # type: ignore
    print(
        "(%d, %s) --> prob=%s, prediction=%f" % (
            rid, text, str(prob), prediction # type: ignore
        )
    )
```

Aprendendo os hiperparâmetros



```
// We use a ParamGridBuilder to construct a grid of parameters to search over.  
// With 3 values for hashingTF.numFeatures and 2 values for lr.regParam,  
// this grid will have 3 x 2 = 6 parameter settings for CrossValidator to choose from.  
val paramGrid = new ParamGridBuilder()  
  .addGrid(hashingTF.numFeatures, Array(10, 100, 1000))  
  .addGrid(lr.regParam, Array(0.1, 0.01))  
  .build()  
  
// We now treat the Pipeline as an Estimator, wrapping it in a CrossValidator instance.  
// This will allow us to jointly choose parameters for all Pipeline stages.  
// A CrossValidator requires an Estimator, a set of Estimator ParamMaps, and an Evaluator.  
// Note that the evaluator here is a BinaryClassificationEvaluator and its default metric  
// is areaUnderROC.  
val cv = new CrossValidator()  
  .setEstimator(pipeline)  
  .setEvaluator(new BinaryClassificationEvaluator)  
  .setEstimatorParamMaps(paramGrid)  
  .setNumFolds(2) // Use 3+ in practice  
  .setParallelism(2) // Evaluate up to 2 parameter settings in parallel
```

Na próxima aula

Algoritmos implementados no Spark MLLib

Desenvolvimento de Soluções Utilizando Spark

Capítulo 6. Spark GraphX

Prof. Pedro Calais



Desenvolvimento de Soluções

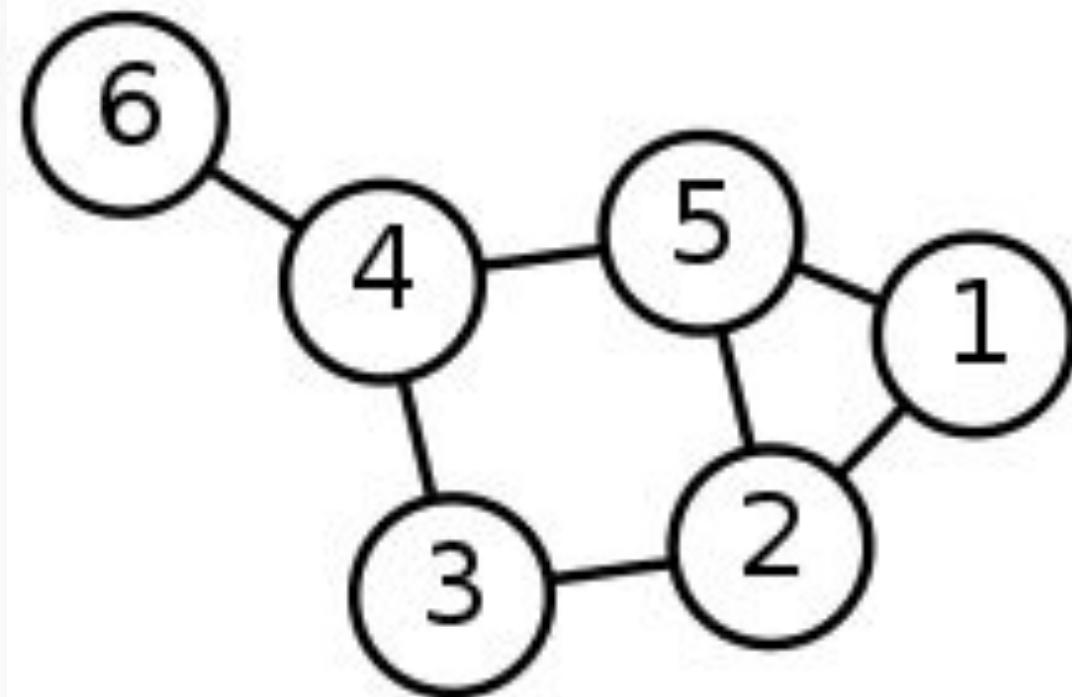
Utilizando Spark

Aula 6.1. O que é o Spark GraphX?

Prof. Pedro Calais

O que são grafos?

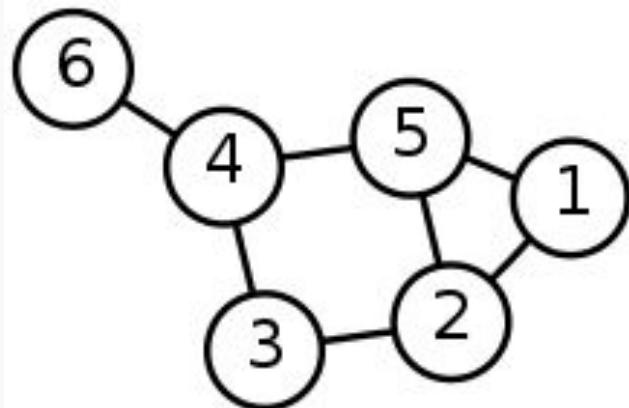
- Uma estrutura matemática das mais importantes em Computação!
- Conjunto de vértices e arestas.



Onde encontramos grafos?

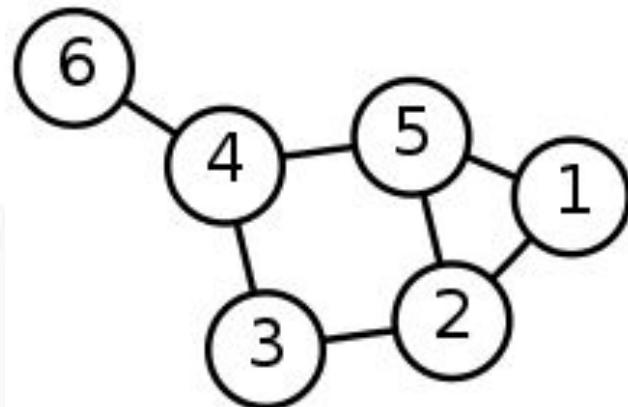
IGTI

- Redes sociais.
- Sistemas de telecomunicação.
- Páginas da web.
- Citação de artigos.

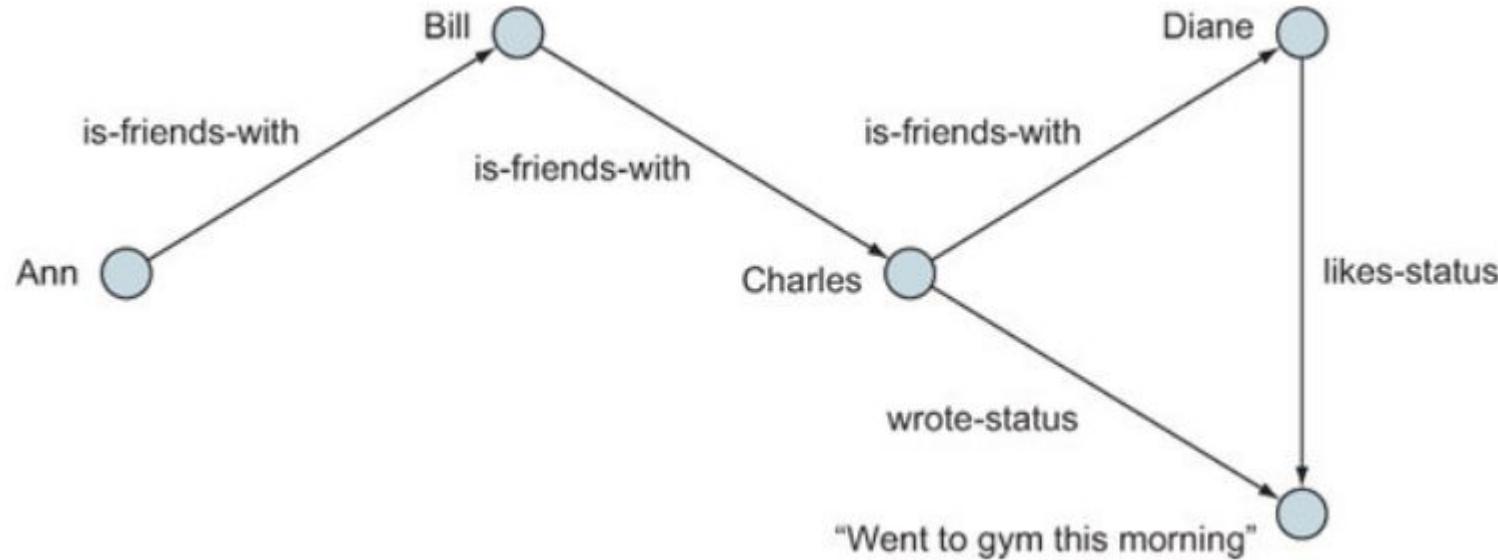


Aplicações

- Datapoints são importante individualmente, mas os grafos exploram as conexões entre eles!
- Aplicações:
 - Detecção de fraude.
 - Recomendação.
 - Bioinformática.
 - Ranking de documentos da Web.



Relações podem ter rótulos!



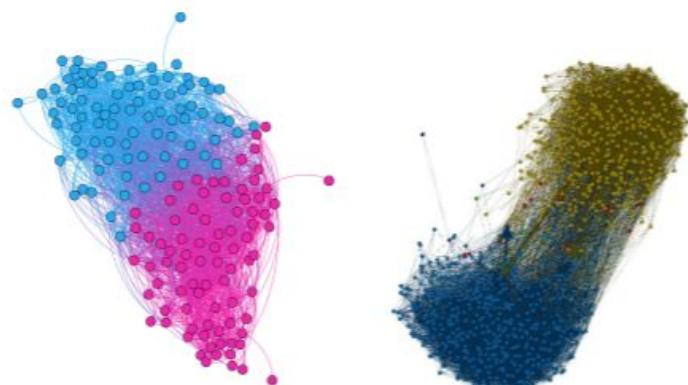
Fonte: Spark GraphX in Action.

O que são grafos?

A Measure of Polarization on Social Media Networks Based on Community Boundaries

Pedro H. Calais Guerra, Wagner Meira Jr.
Dept. of Computer Science
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG, Brazil

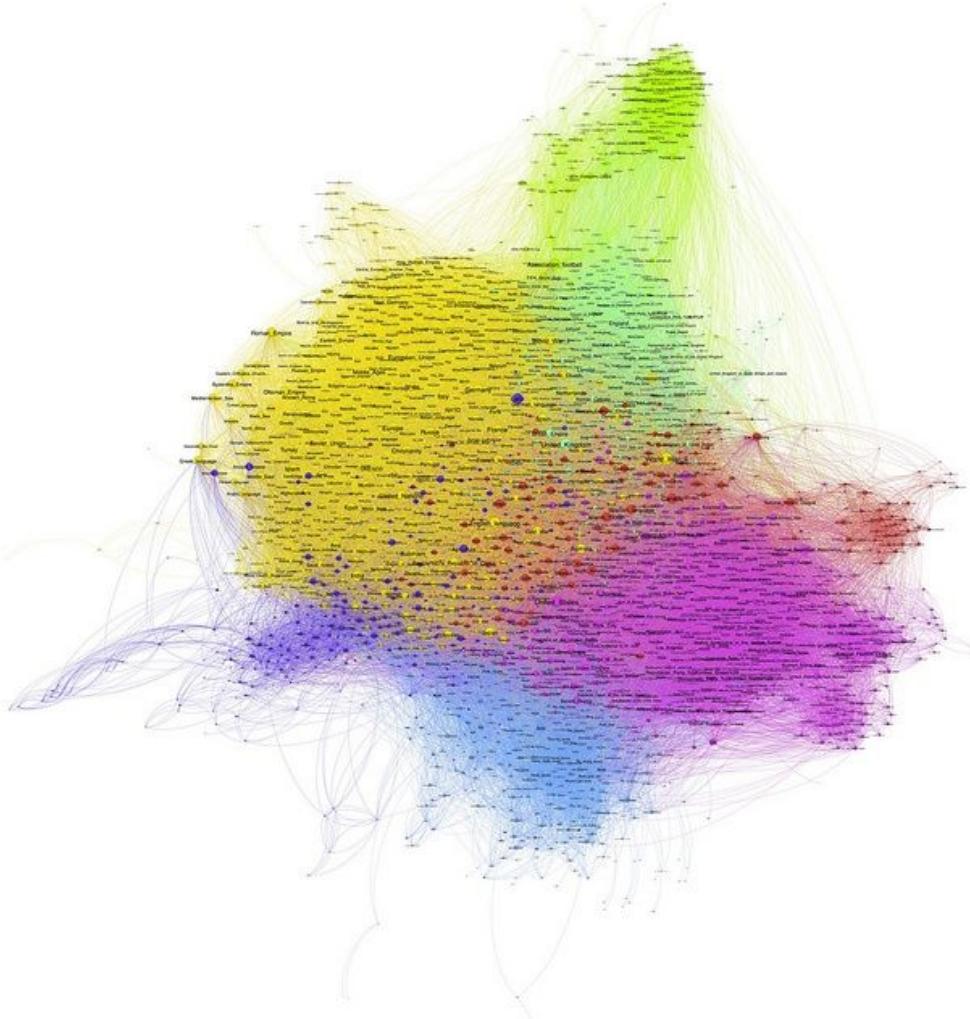
Claire Cardie, Robert Kleinberg
Dept. of Computer Science
Cornell University
Ithaca, NY 14853 USA



(a) Network of Facebook Friends (b) 2004 U.S. Political Blo-
from an University Department gosphere

O que são grafos?

iGTi



Um grafo realmente grande

IGTI

The screenshot shows the homepage of the Common Crawl website. The header features a yellow background with the text "Common Crawl" and a small spider icon on the left. On the right, there is a navigation bar with links: "BIG PICTURE ▾", "THE DATA ▾", "ABOUT ▾", and "BLOG". Below the header, a large white box contains the word "Us" in a large, bold, white font. To the right of this box, a yellow callout box contains the text: "We build and maintain an open repository of **web crawl data** that can be **accessed and analyzed by anyone**."

Common Crawl

BIG PICTURE ▾ THE DATA ▾ ABOUT ▾ BLOG

Us

We build and maintain an open repository of **web crawl data** that can be **accessed and analyzed by anyone**.

Um grafo realmente grande

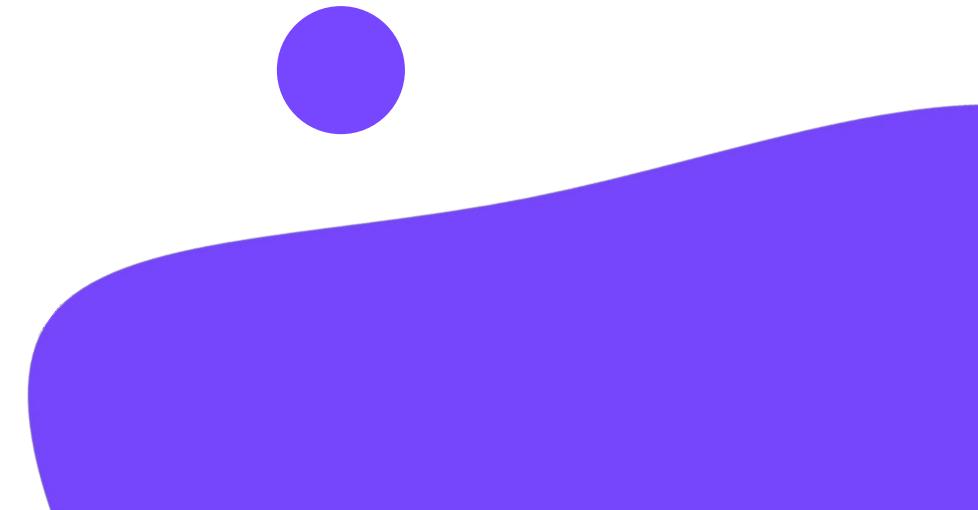
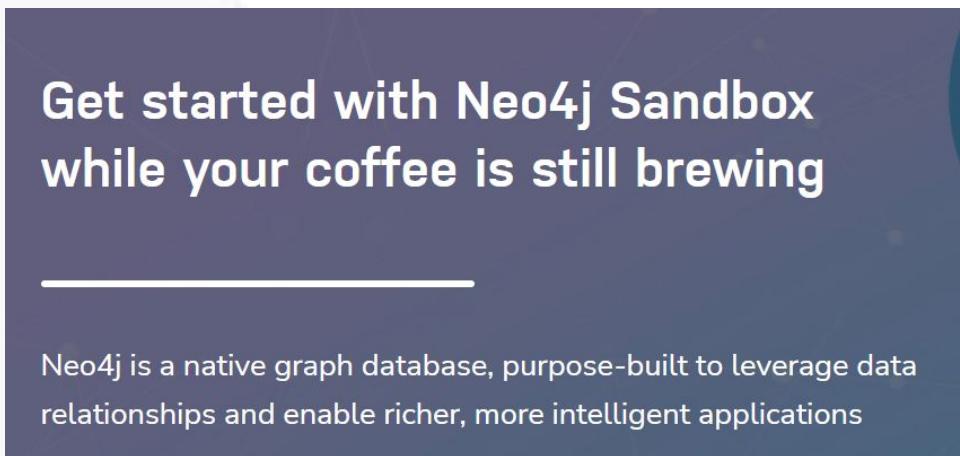
IGTI



Processando grafos



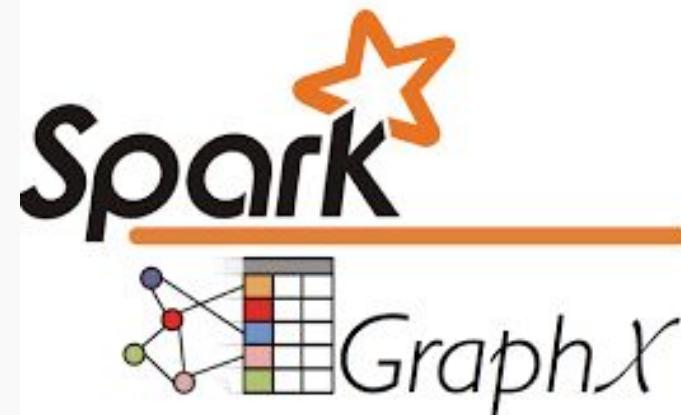
- Ferramentas tradicionais como bancos de dados relacionais não funcionam bem.
- SQL para pesquisar uma sequência de conexões é difícil de escrever e manter.
- Existem sistemas de processamento de grafos como o Neo4J:



O que é o GraphX?

IGTI

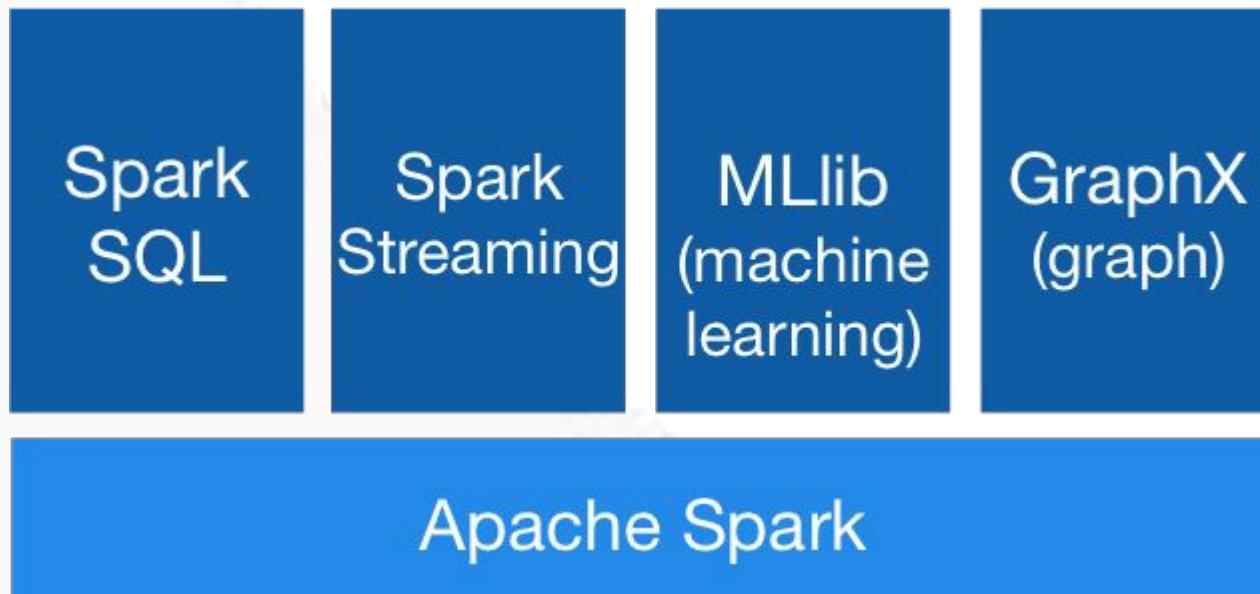
- Camada de processamento de grafos que funciona sobre o Spark.
- Útil para grafos que não cabem na memória de uma máquina.



O que é o GraphX?



- Componente do Spark para grafos e computações em paralelo sobre grafos.
- API para processamento de grafos.
- Permite extrair insights de grandes grafos.



O artigo oficial do GraphX!



GraphX: A Resilient Distributed Graph System on Spark

Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica

AMPLab, EECS, UC Berkeley
`{rxin, jegonzal, franklin, istoica}@cs.berkeley.edu`

O artigo oficial do GraphX!

ABSTRACT

From social networks to targeted advertising, big graphs capture the structure in data and are central to recent advances in machine learning and data mining. Unfortunately, directly applying existing data-parallel tools to graph computation tasks can be cumbersome and inefficient. The need for intuitive, scalable tools for graph computation has lead to the development of new *graph-parallel* systems (*e.g.*, Pregel, PowerGraph) which are designed to efficiently execute graph algorithms. Unfortunately, these new graph-parallel systems do not address the challenges of graph construction and transformation which are often just as problematic as the subsequent computation. Furthermore, existing graph-parallel systems provide limited fault-tolerance and support for interactive data mining.

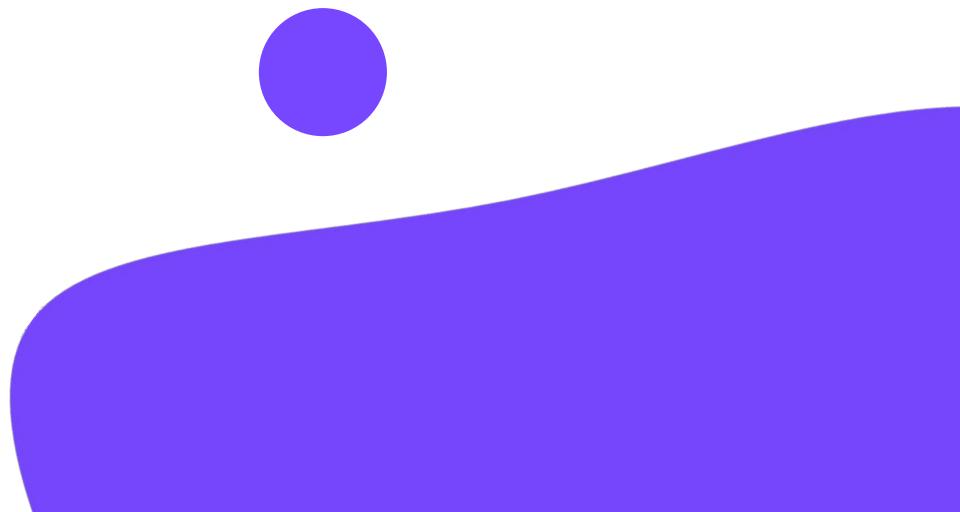
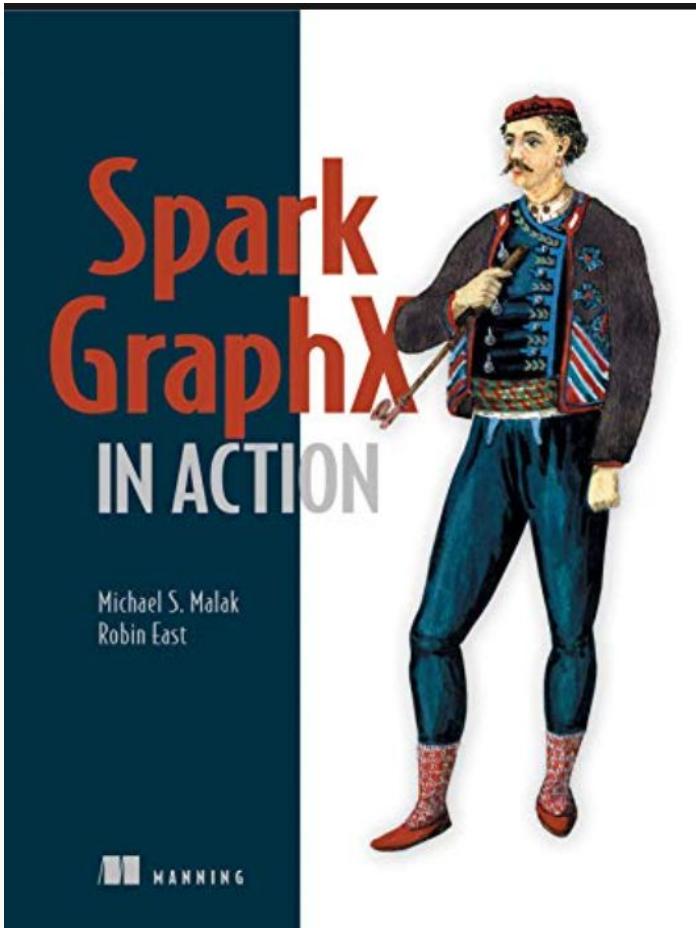
We introduce GraphX, which combines the advantages of both data-parallel and graph-parallel systems by efficiently expressing graph computation within the Spark data-parallel framework. We leverage new ideas in distributed graph representation to efficiently distribute graphs as tabular data-structures. Similarly, we leverage advances in data-flow systems to exploit in-memory computation and fault-tolerance. We provide powerful new operations to simplify graph construction and transformation. Using these primitives we implement the PowerGraph and Pregel abstractions in less than 20 lines of code. Finally, by exploiting the Scala foundation of Spark, we enable users to interactively load, transform, and compute on massive graphs.

O artigo oficial do GraphX!

1. a new graph abstraction called the Resilient Distributed Graph (RDG) that supports a wide range of graph operations on top of a fault-tolerant, interactive platform.
2. a tabular representation of the efficient vertex-cut partitioning described by [6] and data-parallel partitioning heuristics.
3. implementations of the PowerGraph and Pregel graph-parallel frameworks using RDGs in less than 20 lines of code each.
4. preliminary performance comparisons between a popular data-parallel and graph-parallel frameworks running PageRank on a large real-world graph.

Spark GraphX in Action

IGTI



Artigos na academia



COMPUTER SCIENCE *and* ENGINEERING

[Home](#) | [Author Index](#) | [Proceedings](#) | [Series List](#)

COMMUNITY DETECTION OF HETEROGENEOUS NETWORKS WITH RANKCLUS BASED ON SPARK GRAPHX

YU YANG, YONG GUO, BO DENG, HAILONG LI

RESEARCH-ARTICLE

Improving the shortest path finding algorithm in apache spark graphX



Artigos na academia



Overlap Graph Reduction for Genome Assembly using Apache Spark

Alexander J. Paul

Bioinformatics and Computational
Biology Program
Saint Louis University
St. Louis, MO 63103
apaul7@slu.edu

Dylan Lawrence

Computational and Systems Biology
Program
Washington University in St. Louis
St. Louis, MO 63130
dylan.lawrence@wustl.edu

Tae-Hyuk Ahn

Department of Computer Science
Saint Louis University
St. Louis, MO 63103
ahnt@slu.edu

Na próxima aula

Algoritmos implementados no Spark GraphX

Desenvolvimento de Soluções Utilizando Spark

Capítulo 7. Spark Streaming

Prof. Pedro Calais



Desenvolvimento de Soluções

Utilizando Spark

Aula 7.1. Spark Streaming

Prof. Pedro Calais

Dados em batch vs dados em stream



- Dados em batch:
 - São coletados ao longo do tempo.
 - Depois de coletados, dados são enviados para processamento.
 - Processamento pode ser demorado.
- Dados em stream:
 - São produzidos continuamente.
 - São processados um a um.
 - Processamento deve ser rápido e o resultado é necessário imediatamente.

Exemplos de dados em stream



- Finanças: preços de ações, identificar oportunidades de arbitragem.
- Monitoramento de logs e detecção de fraudes / hacking / DDOS.
- Sites de e-commerce: Clickstream.

Big Data e *streams*



 B2C BRANDVIEWS › UPWORK

Streaming Data: Big Data at High Velocity



Tyler Keenan – August 9, 2016

Big Data e *streams*

iGTD



Quatro principais características

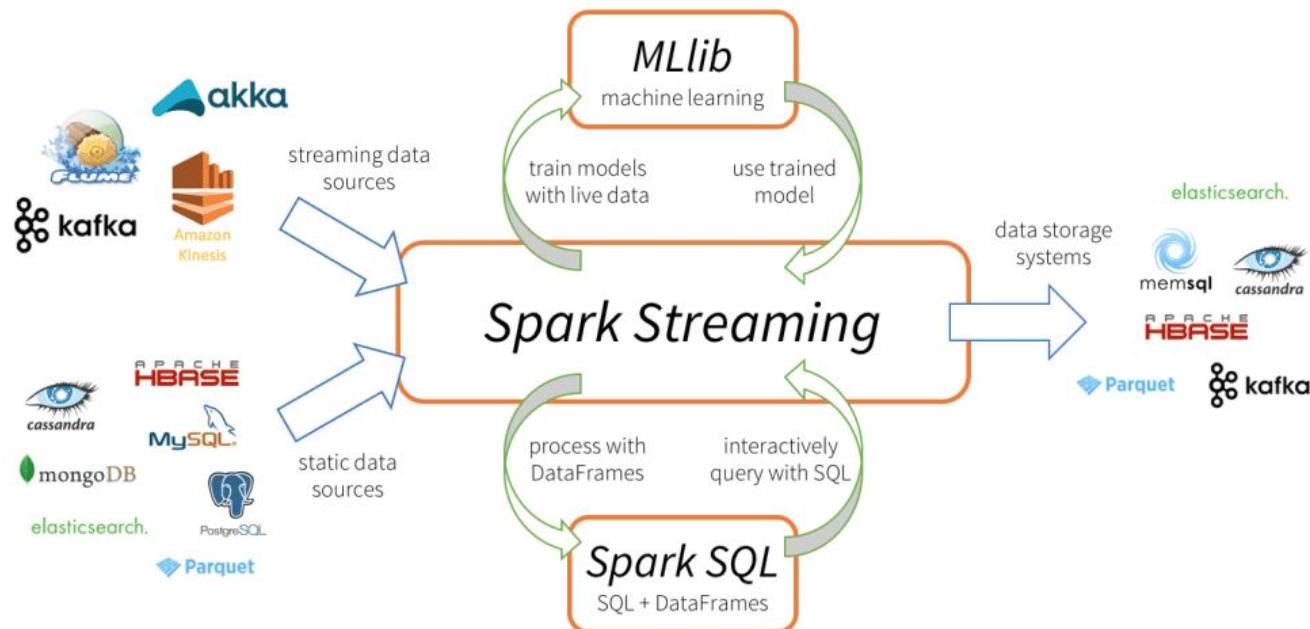
IGTI

- Recuperação rápida de falhas e lentidões.
- Balanceamento de carga e otimização dos recursos.
- Combinação de dados em stream e dados estáticos.
- Integração com Spark SQL, GraphX, MLlib

 **Spark**
Streaming

Spark Streaming

- Extensão da API core do Spark que permite que cientistas e engenheiros de dados lidem com dados em *streams* oriundos de várias fontes.



Conferences > 2018 IEEE Workshop on Environ... 

Sensor data collection and analytics with thingsboard and spark streaming

Publisher: IEEE

[Cite This](#)

 [PDF](#)

Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming

Publisher: IEEE

Cite This

PDF

An enforcement of real time scheduling in Spark Streaming

Publisher: IEEE

[Cite This](#)[PDF](#)

Xinyi Liao ; Zhiwei Gao ; Weixing Ji ; Yizhuo Wang [All Authors](#)

A real-time big data sentiment analysis for iraqi tweets using spark streaming

Nashwan Dheyaa Zaki, Nada Yousif Hashim, Yasmin Makki Mohialden, Mostafa Abdulghafoor Mohammed, Tole Sutikno, Ahmed Hussein Ali

Online Internet traffic monitoring system using spark streaming

Publisher: TUP

[Cite This](#)[PDF](#)

Baojun Zhou ; Jie Li ; Xiaoyan Wang ; Yu Gu ; Li Xu ; Yongqiang Hu ; Lihua Zhu [All Authors](#)

Artigos

IGTI

Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming

Publisher: IEEE

Cite This

PDF

Machine-Learning-Based Online Distributed Denial-of-Service Attack Detection Using Spark Streaming

Publisher: IEEE

Cite This

PDF

Open Access | Published: 04 February 2020

SWEclat: a frequent itemset mining algorithm over streaming data using Spark Streaming

Wen Xiao  & Juan Hu

ORIGINAL PAPER

An improved approach for mining association rules in parallel using Spark Streaming

Longtao Liu, Jiabao Wen✉, Zexun Zheng, Hansong Su,

Processing Moving Objects and Traffic Events Based on Spark Streaming

Publisher: IEEE

Cite This

PDF

Dojin Choi ; Seokil Song ; Bosung Kim ; Insu Bae [All Authors](#)

Online Internet Traffic Measurement and Monitoring Using Spark Streaming

Publisher: IEEE

Cite This

PDF

Baojun Zhou ; Jie Li ; Song Guo ; Jinsong Wu ; Yongqiang Hu ; Lihua Zhu [All Authors](#)

Towards online graph processing with spark streaming

Publisher: IEEE

[Cite This](#)[PDF](#)

Tariq Abughoba ; Farhana Zulkernine

[All Authors](#)

Efficient Bus Arrival Time Prediction Based on Spark Streaming Platform

Publisher: IEEE

[Cite This](#)[PDF](#)

Jing Liu ; Guanfeng Xiao [All Authors](#)

Na próxima aula

Como o Spark Streaming funciona?