Aula 5

Big Data

Prof. Luis Henrique Alves Lourenço

1

Conversa Inicial

2

Spark

- Spark Core
- Spark SQL
- MLLib
- Spark Streaming
- GraphX

3

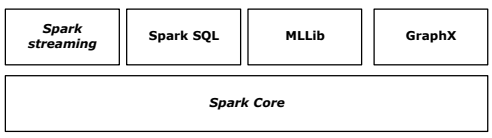
Spark

4

- Fundação Apache
- Sistema de computação em cluster
- Grafos Acíclicos Dirigidos
- Python, Scala ou Java

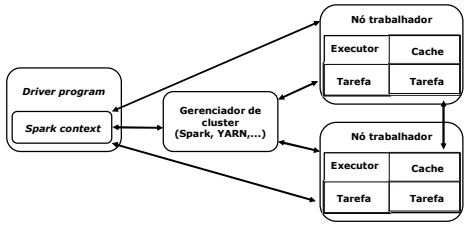
5

Ecossistema Spark



6

Arquitetura Spark



7

RDD

- **Conjunto de dados**
- **Distribuído**
- **Resiliente**

8

SparkContext

- **Criado pelo *Driver Program***
- **RDDs**
- **Resiliência e distribuição de RDDs**
- **conf = SparkConf.setAppName(App).setMaster(master)**
- **sc = SparkContext(conf=conf)**

9

Criando RDDs

- **num = sc.paralelize([1,2,3,4])**
- **distData = sc.textFile(file///caminho/arquivo.txt)**
- **distData = sc.textFile(hdfs://caminho/arquivo.txt)**

10

Transformações RDD

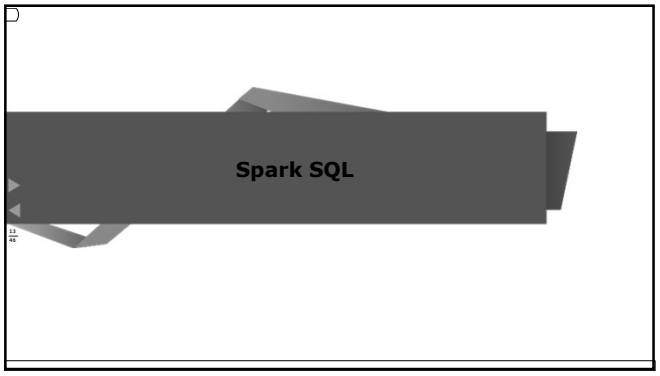
- ***map***
- ***flatMap***
- ***filter***
- ***distinct***
- ***sample***
- ***union***
- ***intersection***
- ***cartesian***

11

Ações RDD

- ***reduce***
- ***collect***
- ***take***
- ***first***
- ***takeSample***
- ***takeOrdered***
- ***foreach***

12



13

- Processamento de dados estruturados
- Suporte ao *Hive*
- *Structured Streaming*

14

Carregando dados

- df = spark.read.load("examples/src/main/
- resources/people.json", format="json")
- df.select("name","age").write.save(
- "namesAndAges.parquet", format="parquet")

15

Fontes de dados

- *Parquet*
- JSON
- ORC
- CSV
- LibVSM
- JDBC

16

Datasets

- RDD com otimizações do Spark SQL
- Não possui API para Python e R

17

DataFrame

- Extensão de RDDs e *Datasets*
- Possui *schema*
- inputData = spark.read.json(arquivoJson)
- hiveContext = HiveContext(sc)
- df = hiveContext.sql("SELECT foo FROM bar ORDERBY foobar")

18

- df.show()
- df.select("campoX")
- df.filter(df['campoX'] > 200)
- df.rdd().map(mapperFunction)

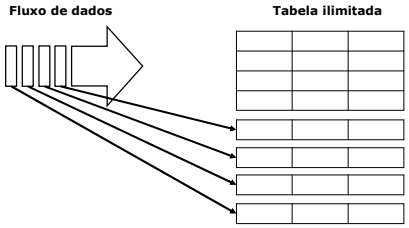
19

Submetendo aplicações

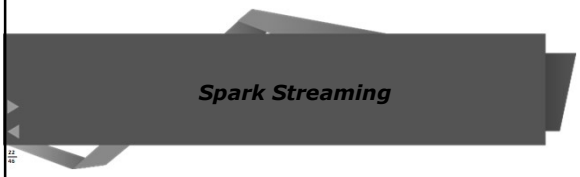
- ./bin/spark-submit \
- --class <classe-principal> \
- --master <url-do-cluster> \
- --deploy-mode <modo-de-deploy> \
- --conf <chave>=<valor> \
- ... # outras opções
- <applicação-jar> \
- [argumentos-da-aplicação]

20

Structured Streaming



21



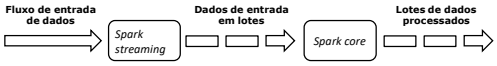
22

- Extensão do *Spark core*
- Escalável
- Altamente eficiente
- Tolerante a falhas

23

- Kafka, Flume, Kinesis, Sockets TCP
- HDFS, S3
- *Map*, *reduce*, *join*, *window*
- *Machine learning*, processamento de grafos, consultas SQL

24



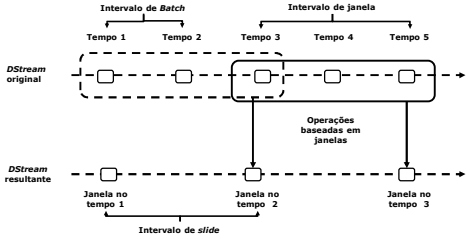
25

DStreams

- ***Discretized Streams***
- **Fluxo contínuo de dados**
- **Sequência de RDDs**
- **Estados dos dados**

26

Transformações de janela



27

DStreams

- ***from pyspark import SparkContext***
- ***from pyspark.streaming import StreamingContext***
- **# Cria um *StreamingContext* em 2 *thread* e *batch interval* de 1 segundo**
- **sc = *SparkContext*("local[2]", "*NetworkWordCount*")**
- **ssc = *StreamingContext*(sc, 1)**

28

- **# Cria um DStream que conecta a uma porta de um *host***
- ***lines = ssc.socketTextStream("localhost", 9999)***
- **# Separa cada linha em palavras**
- ***words = lines.flatMap(lambda line: line.split(" "))***

29

- **# Conta palavras em cada *batch***
- ***pairs = words.map(lambda word: (word, 1))***
- ***wordCounts = pairs.reduceByKey(lambda x, y: x + y)***
- **# Imprime as 10 primeiras palavras de cada RDD**
- ***wordCounts.pprint*()**

30

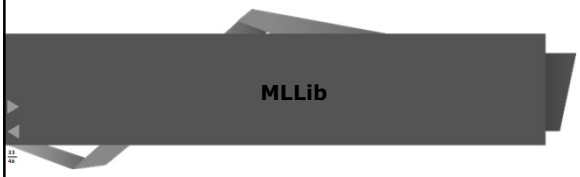
- # Inicia computação
- *ssc.start*()
- # Aguarda a finalização da computação
- *ssc.awaitTermination*()

31

Structured Streaming

- Novo motor de processamento de fluxos
- Fluxos de dados como tabelas
- Baseado em Spark SQL
- *Datasets* e *DataFrames*
- Processamento de *micro-batches*

32



33

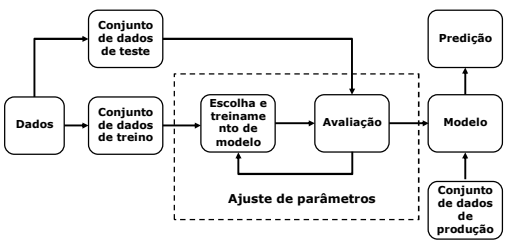
- Aprendizado de máquina
- Escalável e fácil
- Nova API baseada em *DataFrame*
- Fornece classes para simplificar o desenvolvimento e implantação de *pipelines*

34

Aprendizado de máquina

- Aprendizado supervisionado
- Aprendizado não supervisionado
- Aprendizado por reforço

35



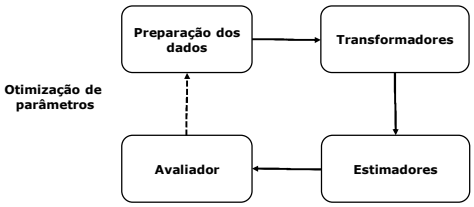
36

Componentes da MLLib

- Algoritmos de aprendizagem de máquina
- Caracterização
- *Pipelines*
- Persistência
- Utilidades

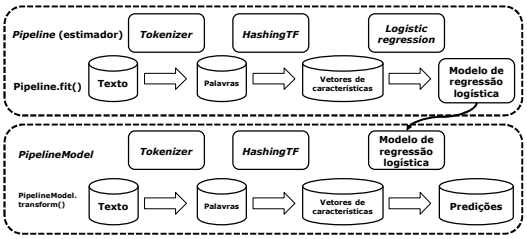
37

Conceitos de um *pipeline*



38

Exemplo de *pipeline*



39

GraphX

40

- Computação de grafos para sistemas distribuídos de larga escala
- Teoria dos grafos
- Sistemas paralelos
- Extensão de RDDs

41

Algoritmos

- ***PageRank*: determinar importância de nós**
- ***Triangle Counting*: coesão de comunidades**
- ***Connected Components*: busca componentes conectados**

42

API GraphX

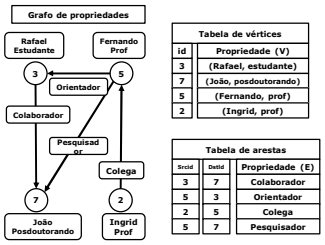


Tabela de vértices

id	Propriedade (V)
3	(Rafael, estudante)
7	(João, posdoutorando)
5	(Fernando, prof)
2	(Ingrid, prof)

Tabela de arestas

SrcId	DstId	Propriedade (E)
3	7	Colaborador
5	3	Orientador
2	5	Colega
5	7	Pesquisador

43

- val sc: SparkContext
- // Cria um RDD para os vértices
- val usuarios: RDD[(VertexId, (String, String))] =
- sc.parallelize(Array((3L, ("Rafael", "estudante")),
- (7L, ("João", "posdoutorando")),
- (5L, ("Fernando", "prof")),
- (2L, ("Ingrid", "prof"))))

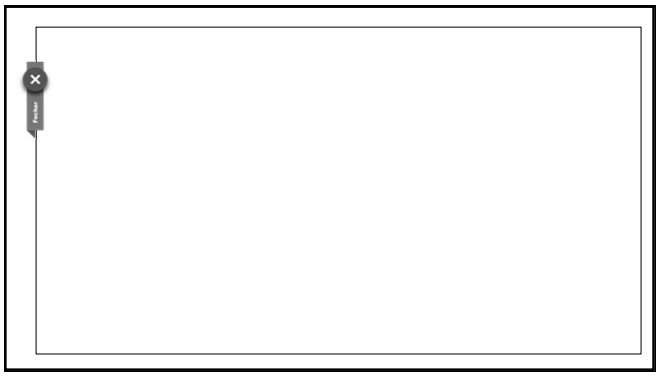
44

- // Cria um RDD para as arestas
- val relacionamentos: RDD[Edge[String]] =
- sc.parallelize(Array(Edge(3L, 7L, "Colaborador"),
- Edge(5L, 3L, "Orientador"),
- Edge(2L, 5L, "Colega"),
- Edge(5L, 7L, "Pesquisador")))

45

- // Define um usuário padrão, caso exista uma relacionamento
- sem usuário definido
- val defaultUser = ("John Doe", "Missing")
- // Constrói o grafo inicial
- val graph = Graph(usuarios, relacionamentos, defaultUser)
- // Contabiliza o número de professores
- graph.vertices.filter { case (id, (name, pos)) => pos == "prof" }.count
- // Contabiliza as arestas onde src > dst
- graph.edges.filter(e => e.srcId > e.dstId).count

46



47