Engenharia de Dados

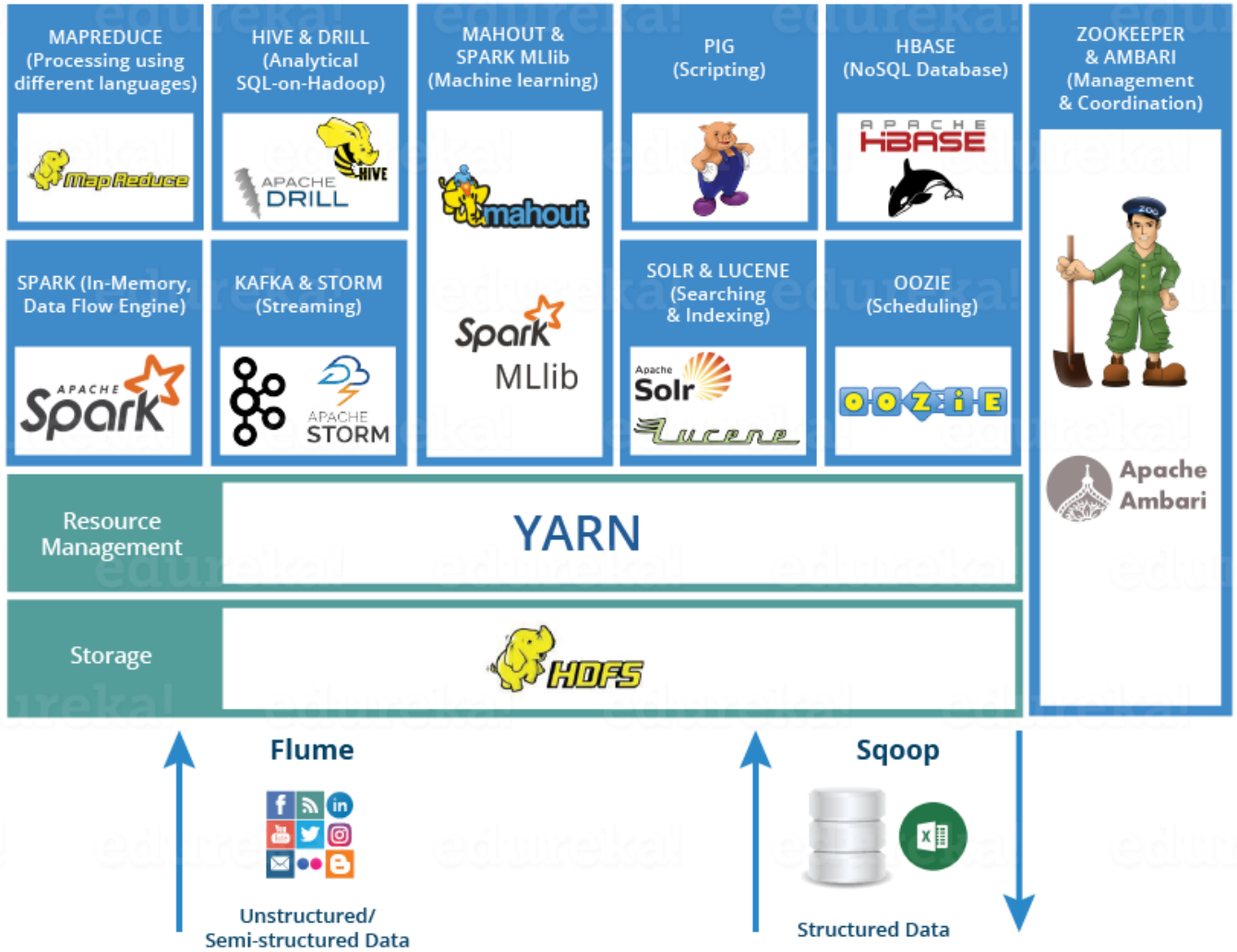
Introdução ao Apache Spark

DCA0132 – Engenharia de Dados

Prof. Carlos M. D. Viegas

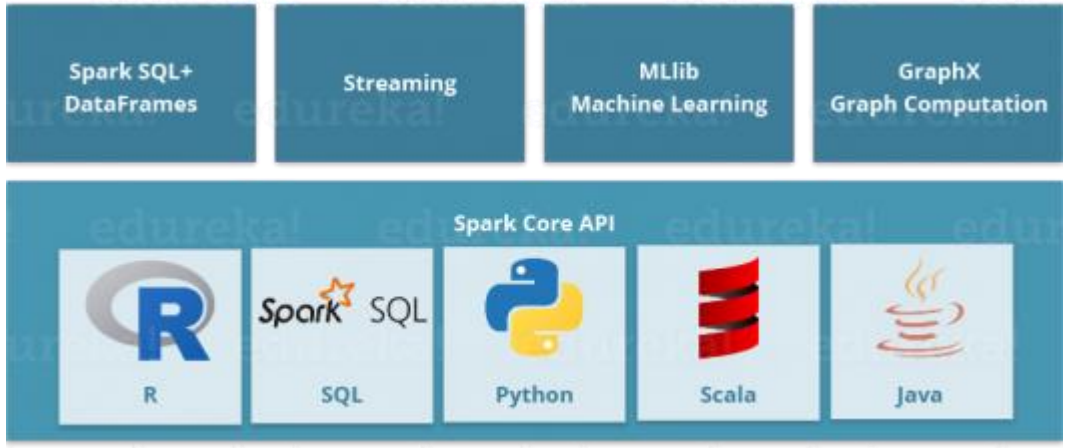
Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte

O Ecossistema Hadoop



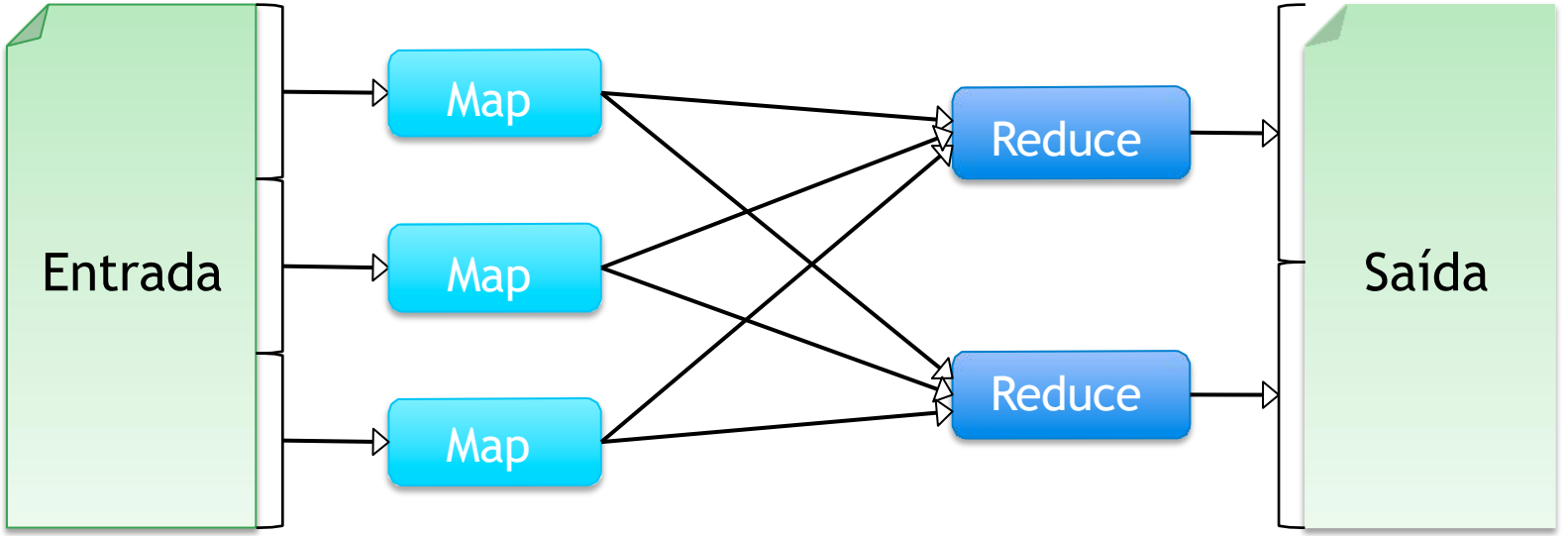
Apache Spark

- Framework para análise de dados em tempo real em sistemas distribuídos (clusters)
- Realiza o processamento dos dados em memória, apresentando melhor desempenho que o Map Reduce
  - Apesar de ser mais rápido, requer mais poder de processamento
- O Spark conta com uma série de bibliotecas de alto nível para processamento dos dados:



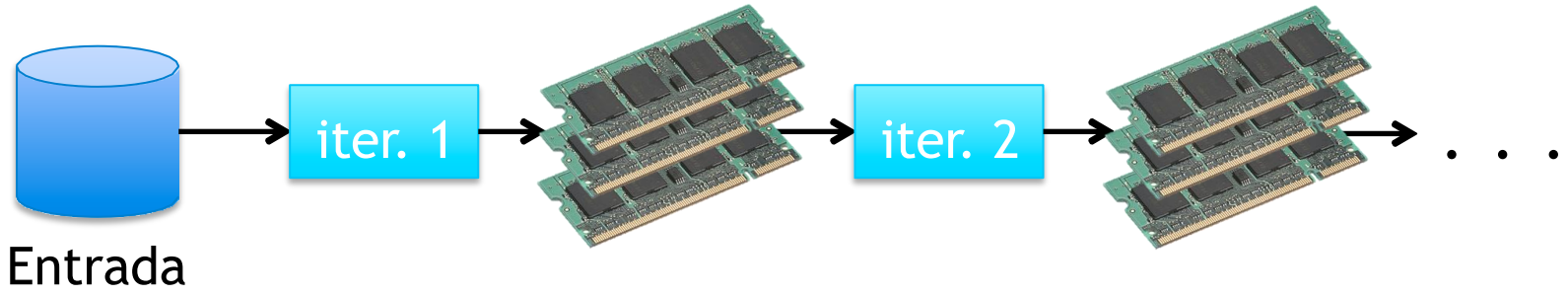
Apache Spark

- Frameworks baseados em disco
  - Resultados intermediários são escritos nos discos rígidos
  - Os dados são recarregados no disco a cada query
  - Fácil de recuperar em caso de falhas
  - Ideal para ETL *workloads* (*extract*, *transform and load*)



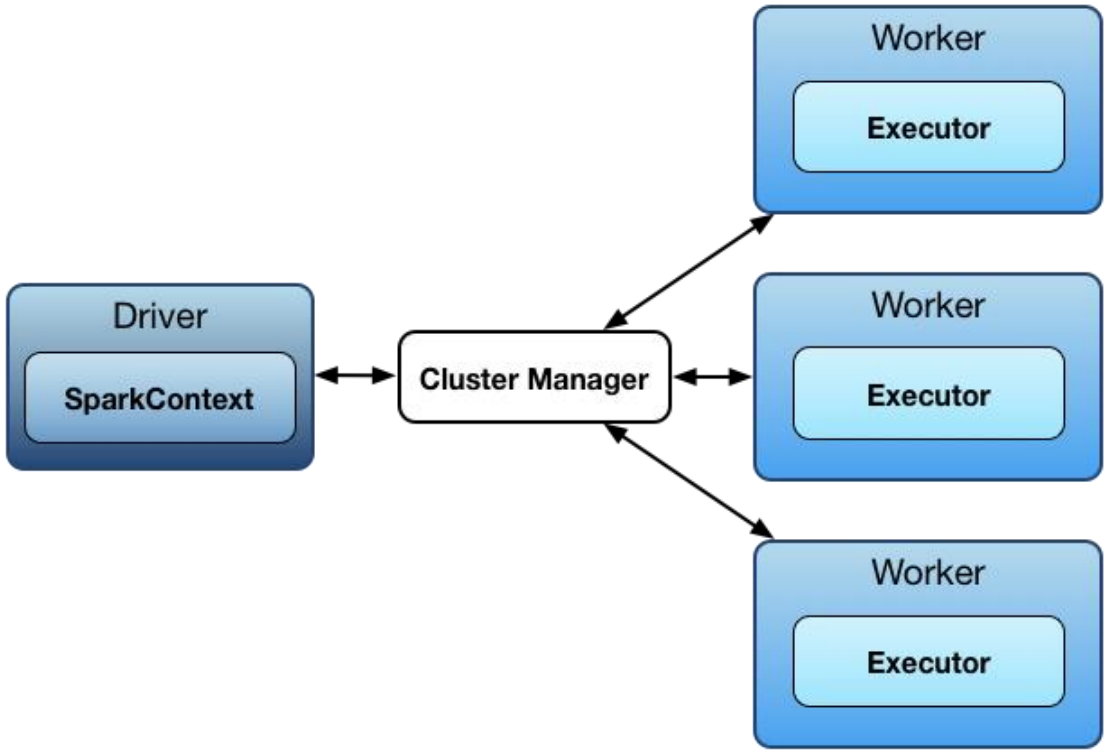
Apache Spark

- Frameworks baseados em memória
  - Remove o custo de I/O nos discos ao manter os resultados intermediários na memória
  - Sensível à disponibilidade da memória
  - Ideal para *workloads* iterativos



Apache Spark

- Arquitetura
  - Uma tarefa do Spark consiste em:
    - Spark Executors que executam as tarefas
    - Spark Driver que escalona as tarefas para os Executors



Apache Spark

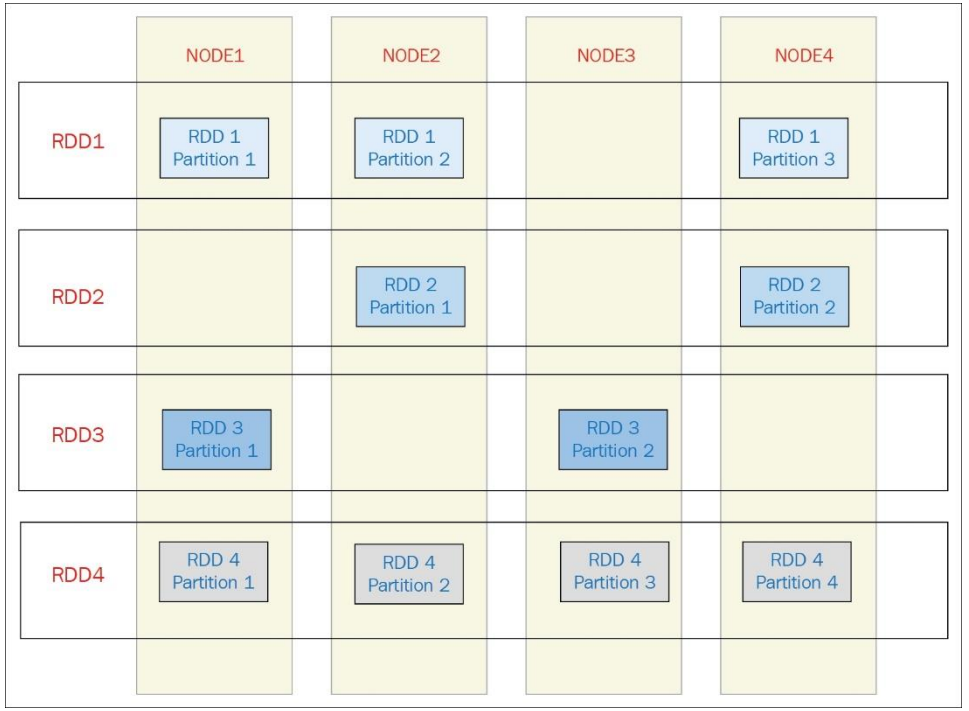
- As tarefas do Spark podem ser executados com Yarn de dois modos:
  - Modo cluster:
    - Todas as tarefas são executadas no cluster
    - O Spark Driver é encapsulado no Application Master do Yarn
    - É mais indicado para tarefas de processamento intenso (demoradas)
  - Modo cliente:
    - O Spark Driver é executado em um cliente (como o seu laptop)
    - Os Executors rodam no cluster
    - É mais indicado para tarefas iterativas, mas as aplicações falham caso o cliente seja encerrado

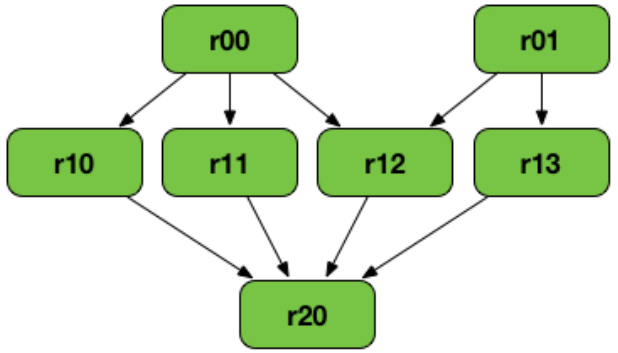
Apache Spark

- Atividades com Spark
  - Duas fases:
    - A primeira será instalado e configurado o ambiente
    - A segundo será o estudo da arquitetura, a programação e a execução de tarefas

Resilient Distributed Datasets (RDD)

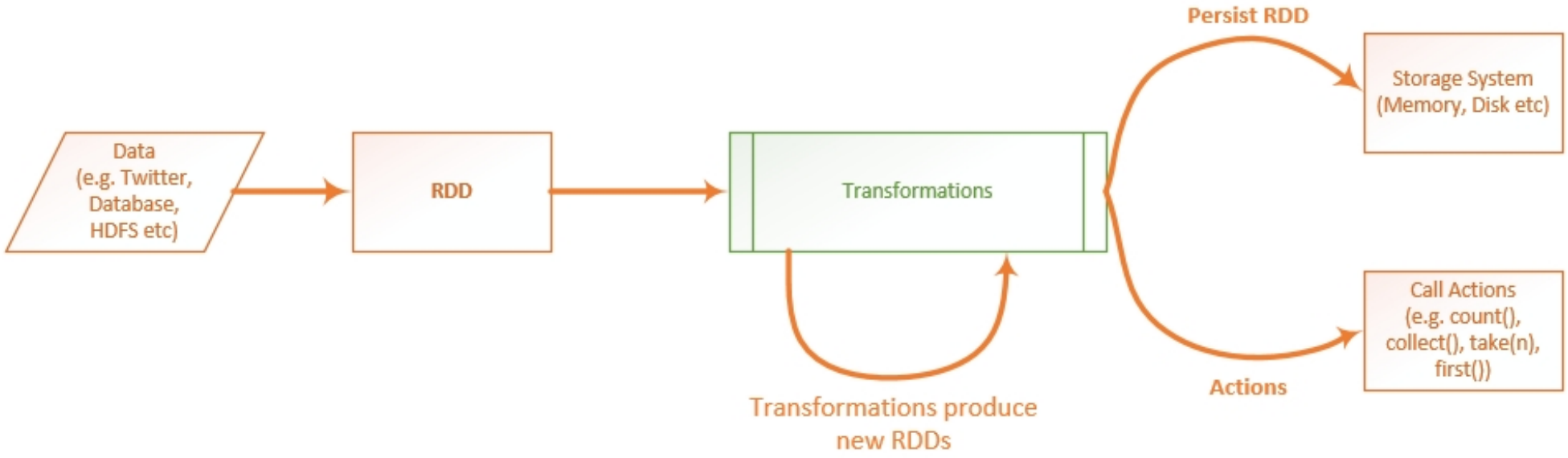
- O Spark recorre a uma nova abstração de dados chamada RDD
  - RDD são conjuntos de dados distribuídos pelos nós do cluster, e que são resilientes a falhas
  - São capazes de recuperar as informações perdidas, recorrendo a análise dos dados a partir de quem os gerou (*lineage*)
- Além disso, o RDD:
  - Pode ser armazenado em memória até o máximo disponível de capacidade e de tempo
  - É imutável (read-only) e só pode ser alterado se criada uma cópia de um RDD
  - É processado em paralelo





Resilient Distributed Datasets (RDD)

- O fluxo de uma aplicação em Spark com RDD inclui:
  - Criação do RDD com os dados de entrada
  - Um conjunto de transformações (*map*, *filter*, *join*, entre outros)
  - Persistência do RDD para evitar execuções repetidas (gravar os dados)
  - Executar ações sobre o RDD para iniciar o processamento em paralelo dos dados

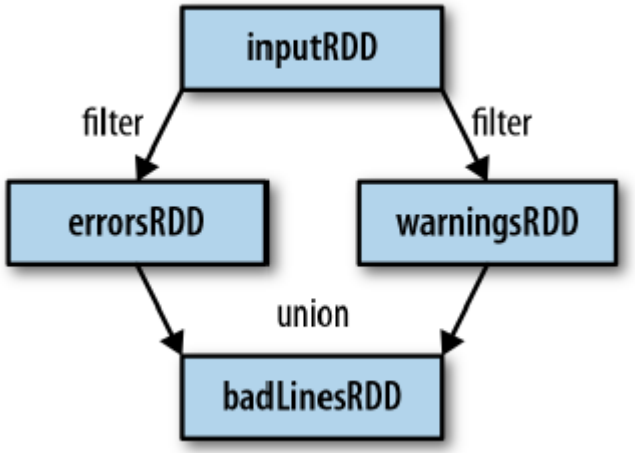


Resilient Distributed Datasets (RDD)

- Exemplo de transformações

```
inputRDD = sc.textFile("log.txt")
errorsRDD = inputRDD.filter(lambda x: "error" in x)
```

```
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```



- Exemplo de ações

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
        print line
```

Criando Aplicações com Apache Spark em RDD

- Primeiro exemplo com PySpark
  - É um shell para executar programar aplicações Spark em linguagem python
  - Para executar:

```
~$ pyspark
```

Criando Aplicações com Apache Spark em RDD

- Primeiro exemplo com PySpark
  - Criar um RDD para busca de strings
    - Subir um arquivo de texto para o HDFS (em um terminal fora do shell PySpark)
      - `~$ hdfs dfs –put arquivo.txt`
    - Ler o arquivo com a função *textFile()* (dentro do shell PySpark)
      - `>>> texto = sc.textFile(“hdfs://node-master:9000/user/root/arquivo.txt”)`
    - Verificar o que foi carregado no RDD
      - `>>> texto.collect()`
    - Definir a string a ser buscada
      - `>>> resultado = texto.filter(lambda line: “texto a ser procurado" in line)`
    - Para exibir a primeira ocorrência da string
      - `>>> resultado.first()`
    - Para contar a quantidade de ocorrências da string
      - `>>> resultado.count()`
    - Para gravar os dados e poder utilizar novamente
      - `>>> resultado.persist`

Criando Aplicações com Apache Spark em RDD

- Primeiro exemplo com PySpark
  - Criar um RDD para busca de strings
    - Exibir apenas algumas linhas de texto
      - `>>> resultado.take(5)`

Criando Aplicações com Apache Spark em RDD

- Aplicação *standalone* em Python (sem recorrer ao shell)

```
"""SimpleApp.py"""
from pyspark import SparkContext

sc = SparkContext("local", "SimpleApp")

logData = sc.textFile("hdfs://node-master:9000/user/root/README.md")

numAs = logData.filter(lambda s: 'c' in s).count()
numBs = logData.filter(lambda s: 'd' in s).count()

print "\n\nLines with c: %i, lines with d: %i\n" % (numAs, numBs)
```

- PS: é necessário subir o arquivo README.md ou qualquer outro para o exemplo funcionar

Criando Aplicações com Apache Spark em RDD

- Combinar map() e reduce()
  - Map() é uma transformação que se aplica sobre cada dado do RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
        print "%i " % (num)
```

  - Reduce() é uma transformação que opera sobre dois elementos e retorna um resultado

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

Criando Aplicações com Apache Spark em RDD

- Tarefa:
  - Desenvolver uma aplicação utilizando os conceitos anteriores, que realize as transformações de map e reduce sobre o conjunto de dados “conjunto2”. Este conjunto de dados deve ser inserido em um RDD no HDFS. Pretende-se que sejam lidos todos os cargos e apresentada a quantidade de pessoas nos mesmos.

  - Conjunto de dados disponível em: `https://goo.gl/A3MhFS`
    - Datasets > conjunto2.csv

Criando Aplicações com Apache Spark em RDD

- A “função” **lambda**
  - Permite criar pequenas funções "anônimas" que retornam o resultado de uma operação sobre os seus argumentos
  - Como exemplo `lambda a,b: a + b`, retorna o resultado da soma de `a + b`.
  - Estas funções podem ser utilizadas para quaisquer operações que forem necessárias, entretanto estão restritas a apenas uma expressão

  - Exemplos (em Python):
    - Função lambda como retorno de uma função

```
def make_incrementor(n):
    return lambda x: x + n
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

Criando Aplicações com Apache Spark em RDD

- Guia para programação de RDD em Python: https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds

Criando Aplicações com Apache Spark em RDD

- Exemplos
  - Contagem de palavras

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

  - Criar sequências de RDD e salvar para arquivo

```
rdd = sc.parallelize(range(1, 4)).map(lambda x: (x, "palavra " * x))
rdd.saveAsSequenceFile("hdfs://node-master:9000/user/ubuntu/saida.rdd")
sorted(sc.sequenceFile("hdfs://node-master:9000/user/ubuntu/saida.rdd").collect())
```

Criando Aplicações com Apache Spark em RDD

- Exemplos
  - Passagem de parâmetros em funções

```
"""MyScript.py"""
if __name__ == "__main__":
    def myFunc(s):
        words = s.split(" ")
        return len(words)

    sc = SparkContext(...)
    sc.textFile("arquivo.txt").map(myFunc)
```

Criando Aplicações com Apache Spark em RDD

- Tarefa:
  - Suponha que você precisa desenvolver um aplicativo móvel que exiba as informações das linhas de ônibus de uma determinada cidade.
  - Para isso, lhe é fornecido um conjunto de dados “onibus.tar”, disponível em: `https://goo.gl/A3MhFS`
    - Datasets > onibus.tar
  - Deve-se recorrer ao Spark, utilizando os conceitos estudados até o momento, e implementar as transformações de **map()** e **reduce()** sobre o conjunto de dados fornecido. Este conjunto de dados deve ser inserido em um RDD no HDFS.
  - Pretende-se que a aplicação seja capaz de associar as informações dos três arquivos que compõem o conjunto de dados e apresente como saída:
    - O número e nome da linha de ônibus
    - Os horários e o nome dos pontos em que essa linha passa

Fonte dos dados:

Dados abertos da UFPR: http://dadosabertos.c3sl.ufpr.br/curitibaurbs/