# Big Data

## Apache Spark

Prof. Jean Paul Barddal

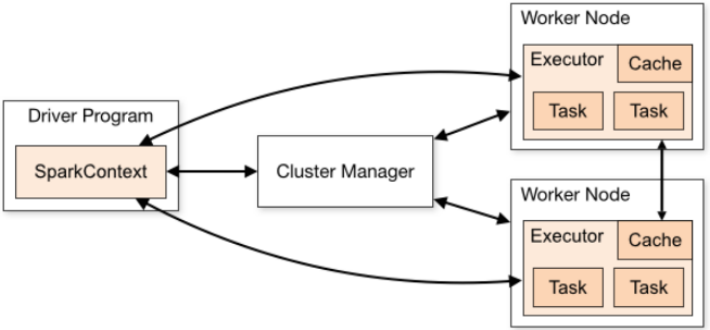
# Agenda

# Agenda

# Apache Spark

- Framework para processamento de dados em memória
- Altamente escalável
- Permite processamento em lote (batch) e em streams (mini-batch)
- Otimizado para SQL e Machine Learning
- Programação baseada em DAGs
- Flexível: uso em Java, Scala, Python e R

# Arquitetura Master-Slave



# Arquitetura Master-Slave

- Driver Program
  - Responsável por coordenar um Job
  - Executa a rotina principal, convertendo as tarefas a serem executadas nos nós Workers
  - Contém um SparkContext, que conecta o master aos slaves
- Worker Node
  - Responsável por conduzir tarefas individuais
  - Ciclo de vida independe da task
- Cluster Manager
  - Gerencia os recursos do cluster
  - Modos: Standalone (auto-gerenciável) ou distribuído (YARN, Mesos, etc)

# Agenda

# Fluxo Geral de Rotinas Spark

1. Criar um Spark Context
2. Criar um RDD inicial
3. Aplicar transformações
4. Executar operações
5. Mostrar / salvar resultados

# Atividade

- Vamos executar a nossa primeira rotina em Spark
- Vamos iniciar contando palavras

# Agenda

# Resilient Distributed Datasets (RDDs)

- RDD é uma coleção de dados homogênea, i.e., Integer, Float, Double, etc, ou qualquer outra classe desejada
- RDDs foram criados para armazenar e processar massivas quantidades de dados
- Uma vez criados, são imutáveis
- Persistidos em memória ou em disco
- Tolerante a falha
- Dados em um RDD são distribuídos automaticamente e de forma transparente

# Criando RDDs: paralelizando dados

- Forma mais simples de criar um RDD
- Usa uma estrutura de lista disponível na linguagem do host e a converte em um RDD
- Comum em processos de debug
- Não faz sentido para grandes quantidades de dados, uma vez que o Driver Program mantém todos os dados antes da paralelização

```
1 rdd = sc.parallelize([1,2,3,4,5])
```

# Criando RDDs: carregando arquivos

- Arquivo local
- HDFS
- Cloud
- Bases de dados via JDBC

```
1 rdd = sc.textFile('bible.txt')
```

# Transformações e Operações (Ações)

- Ambas podem ser realizadas com RDDs
- Tenha em mente o seguinte:
- Uma transformação recebe um ou mais RDDs como entrada e resulta em outro RDD
- Exemplos: map, filter, flatMap, sample, distinct, union, intersection, subtract, cartesian, etc
- Uma operação recebe um ou mais RDDs como entrada e resultado em algo além de um RDD
- Exemplos: collect, count, countByValue, take, saveAsTextFile, reduce
- Vamos trabalhar com todas essas funções

# Lazy Evaluation

- Tanto transformações quanto operações são usadas com RDDs
- Contudo, elas são processadas diferentemente pelo Spark
- Racional: Spark é preguiçoso e fará processamentos apenas quando necessário. Na prática, alguns processamentos podem ocorrer de forma conjunta (otimização)
- Transformações são atrasadas até que uma operação seja demandada
- Isso diminui o número de computações, dado que várias transformações podem ser combinadas e feitas de uma única vez

# Lazy Evaluation

```
# carregamento de arquivo
rdd = sc.textFile('file://' + SparkFiles.get('bible.txt'))

# cada linha do arquivo eh convertido em palavras
palavras = rdd.flatMap(lambda l: l.split(" "))

# removendo caracteres especiais e colocando tudo em caixa alta
palavras_limpas = palavras.map(lambda x: x.replace('.', '').
                               replace(',', '').
                               replace(';', '').
                               upper())

# filtrando palavras que comeca com vogal
palavras_inicio_vogal = palavras_limpas.filter(lambda x: x.startswith('A') or
                                                         x.startswith('E') or
                                                         x.startswith('I') or
                                                         x.startswith('O') or
                                                         x.startswith('U'))
# contagem de ocorrencias (PRIMEIRA OPERACAO)
contagem = palavras_inicio_vogal.countByValue()

# apresentacao dos resultados
for p, c in contagem.items():
  print(f'{p} \t {c}')
```

# Atividade

- Vamos criar uma rotina que carregue o arquivo airports.txt e apresente os aeroportos nos EUA
- Dica sobre o formato do arquivo:
  - Airport ID
  - Airport Name
  - City
  - Country
  - IATA/FAA code
  - ICAO Code
  - Latitude
  - Longitude
  - etc

# Agenda

# Transformações

- Recebem como entrada um ou mais RDDs e resultam em outro RDD
- Exemplos:
  - Map
  - FlatMap
  - Filter
  - Sample
  - Distinct
  - Union
  - Intersect
  - Subtract
  - Cartesian Product

# map

- Função 1 para 1
- Aplica uma função passada como argumento a cada item do RDD de entrada
- Para cada item do RDD de entrada, teremos um RDD de saída

```
1 rdd_uppercase = rdd.map(lambda x: x.upper())
```

# flatMap

- Similar ao map, mas em formato 1 para n (muitos)
- Para cada item de entrada, temos a possibilidades de múltiplas saídas
- Cada item de saída é 'concatenado' de forma linear no RDD de saída
- Exemplo clássico: para cada sentença em um RDD de entrada, geramos múltiplas palavras

```
1  rdd_palavras = rdd.flatMap(lambda l: l.split(' '))
```

# filter

- Como o nome sugere, aplicamos um filtro aos elementos do RDD de entrada
- Usado para manter elementos que aderem a um predicado (condição)

```
1 rdd_pares = rdd.filter(lambda x: x % 2 == 0)
```

# sample

- Função que cria uma amostragem do RDD de entrada
- Comumente utilizado para teste/debug
- Parâmetros:
- Resampling: boolean
- fraction (probabilidade de seleção de um elemento): float

```
1 rdd = sc.parallelize(range(1,101))
2 rdd_amostra = rdd.sample(withReplacement=False, fraction =
    0.01)
```

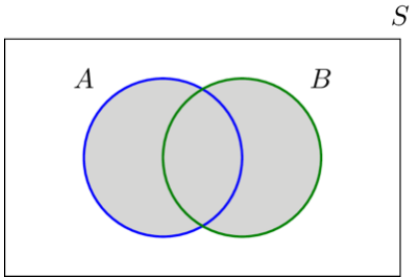
# distinct

- Remove dados duplicados de um RDD de entrada
- Computacionalmente caro
- Requer que todos os dados sejam comparados com os demais
- Lembrete: os dados estão distribuídos em diferentes computadores, o que demanda uso de rede!

```
1 rdd = sc.parallelize([1,2,3,4,5,1,2,3])
2 rdd_dist = rdd.distinct()
```

# union

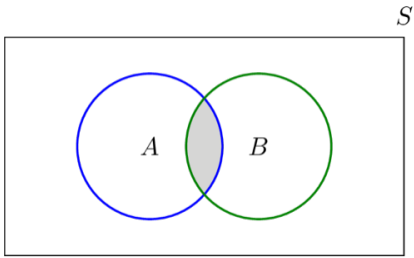
- Similar à definição matemática de união
- Contudo, caso existam dados duplicados, eles são mantidos no RDD de saída (concatenação)




```
1 rdd_a = sc.parallelize([1,2,3,4,5])
2 rdd_b = sc.parallelize([1,2,3,7,8,9])
3 rdd_union = rdd_a.union(rdd_b)
```

# intersection

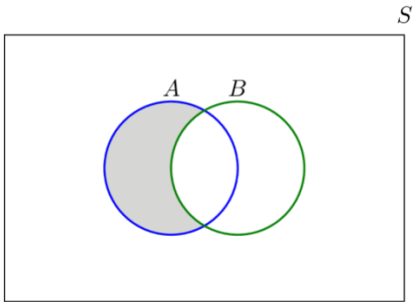
- Interseção entre RDDs
- Resulta em um RDD com os elementos em comum entre dois RDDs de entrada
- Computacionalmente caro, uma vez que comparações são necessárias




```
1 rdd_a = sc.parallelize([1,2,3,4,5])
2 rdd_b = sc.parallelize([1,2,3,7,8,9])
3 rdd_intersection = rdd_a.intersection(rdd_b)
```

# subtract

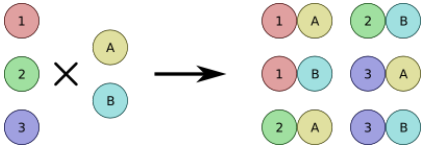
- Subtração entre RDDs
- Retorna um RDD cujos elementos participem do primeiro RDD mas não do segundo




```
1 rdd_a = sc.parallelize([1,2,3,4,5])
2 rdd_b = sc.parallelize([1,2,3,7,8,9])
3 rdd_intersection = rdd_a.subtract(rdd_b)
```

# cartesian

- Combina dois RDDs com as suas combinações de elementos




```
1 rdd_a = sc.parallelize([1,2,3,4,5])
2 rdd_b = sc.parallelize(['A', 'B'])
3 rdd_intersection = rdd_a.cartesian(rdd_b)
```

# Agenda

# Operações / Ações

- Processam os dados de um RDD de forma que o resultado seja enviado para o Driver Program ou que os dados resultantes sejam persistidos em disco
- Operações demandam a execução das transformações (lazy evaluation)
- Operações que utilizaremos:
  - collect
  - count
  - countByValue
  - take
  - saveAsTextFile
  - reduce

# collect

- Retorna um RDD ao Driver Program no formato de lista
- Exemplo: se temos um RDD de strings, o resultado será uma lista de strings
- Importante: a RAM do Driver Program deve ser suficiente para armazenar o RDD inteiro, caso contrário, o sistema pode travar

```
1 rdd = sc.parallelize([1,2,3,4])
2 rdd.collect()
```

# count

- Retorna o número de itens em um RDD

```
1 rdd = sc.parallelize([1,2,3,4])
2 rdd.count()
```

# countByValue

- Dado um RDD, retorna um dicionário com todos os valores únicos e a respectiva frequência

```
1 rdd = sc.parallelize([1,2,3,4,5,1,2,3])
2 rdd.countByValue()
```

# take

- Retorna uma lista com $n$ elementos de um RDD
- Não necessariamente segue a ordem original dos dados

```
1 rdd = sc.parallelize(range(1,101))
2 rdd.take(5)
```

# saveAsTextFile

- Persistência em arquivo (local, HDFS, Amazon S3, etc)
- Importante: não sobrescreve arquivos existentes
- A saída segue o padrão de blocos existente no HDFS

```
1 rdd = sc.parallelize(range(1,10))
2 rdd.saveAsTextFile('teste.txt')
```

part-00000

```
1 1
2 2
3 3
4 4
5
```

part-00001

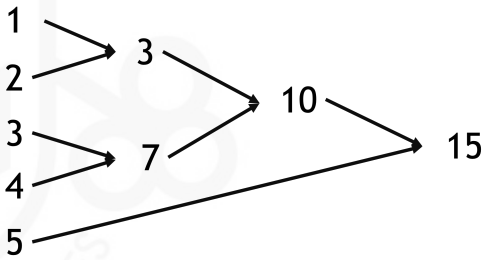
```
1 5
2 6
3 7
4 8
5 9
6
```

# reduce

- Recebe como entrada uma função que processa dois itens de um RDD e retorna um elemento do mesmo tipo
- A função é aplicada repetidamente até que um exemplo único seja obtido

# Example

```
1 rdd = sc.parallelize(range(1,6))
2 soma = rdd.reduce(lambda x, y: x + y)
```



# Agenda

# Exercícios

1. A partir do arquivo airports.txt, filtre todos os aeroportos dos "United States" com latitude maior que 40. Salve esses resultados em arquivo após converter a linha completa em caixa alta.
2. Depois, encontre a média das latitudes de todos os aeroportos nos "United States" e apresente o resultado na tela
3. Dica: latitude está no índice 6 e país está no índice 3