Spark 2 For Devs

MATERIAL DO LABORATÓRIO 1

Lab 3.1: RDD básico

Nesse lab vamos trabalhar com RDD manipulando vários arquivos de tamanho diferente e exercitando o que já vimos

Tempo aproximado: 30 – 40 minutos

- Copiamos para o diretório /home/ec2-user/lab da sua máquina todos os arquivos necessários para seguir com os laboratórios
- Entre na sua maquina e vá até o diretório /opt/polynote
- Agora vamos monitorar tudo o que iremos fazer no polynote
  > tail -f nohup.out
- Entre no polynote "http://<seuip>:8192" e configure para que seja possível rodarmos o spark nele.
- Na parte central do Polynote haverá uma seção chamada "Configuration & dependencies", abra essa seção e vá até a sub-seção "Spark Config" entre com os seguintes valores:
  - spark.master e no campo seguinte local[*]
  - **Clique em Save & Restart**
- Pode minimizar a seção de Configuração
- Agora vamos iniciar uma nova célula passando o mouse lentamente abaixo da seção até que apareça um sinal de + clique para iniciarmos
- Começaremos lendo o arquivo abaixo

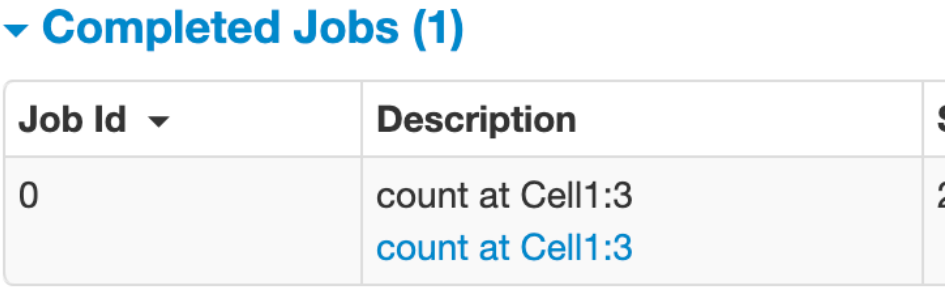
```
val f = spark.sparkContext.textFile("/home/ec2-user/lab/twinkle/sample.txt")
```

- Vamos agora verificar quantas linhas com a palavra "twinkle"existe no dataset. Faremos isso usando a função filter.

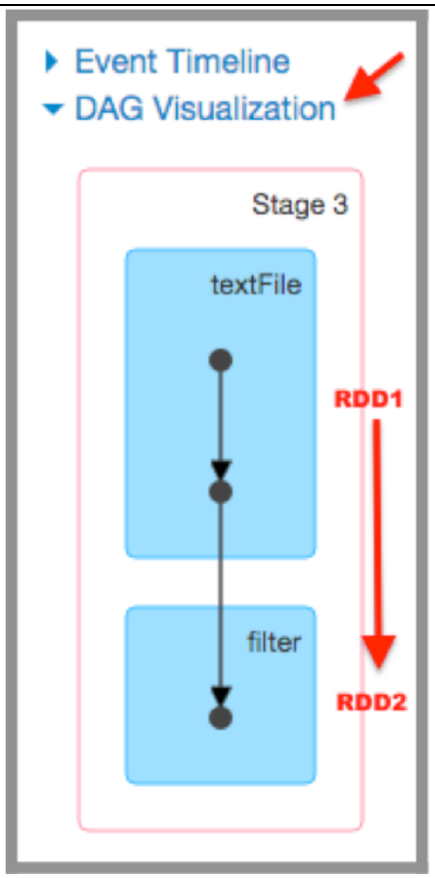
```
val filtered = f.filter(line => line.contains("twinkle"))
```

- **Quantas linhas vieram?**
  - Você pode usar a função count() sobre a variável filtered

- Vá para o Spark UI http://<seuip>:4040
- Clique na seção "Completed Jobs" no link "count at .."



- Visualize a DAG



- Crie uma nova célula
- Podemos mudar o comportamento do shell alterando o nível do log

Altere o nível do log para “INFO”, e repare ao fazer o experimento que a definição da variável filtered não gera qualquer saída de log

```
spark.sparkContext.setLogLevel("INFO")
```

Crie a variável filtered novamente, observe o log (veja o console da sua máquina e não do polynote) e depois rode o count()

```
val filtered_2 = f.filter(line =>
line.contains("twinkle"))
```

- Você verá muito log no output apenas depois que o count() é executado

```
filtered_2.count()
```

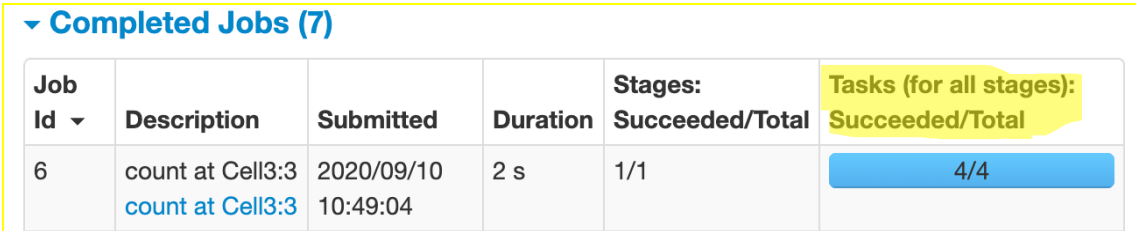
- Por que o log só aparece depois do count() ?
- Volte o log para “WARN”

```
spark.sparkContext.setLogLevel("WARN")
```

- Vamos processar agora arquivos maiores e então ver o que acontece no Spark UI
- Crie uma nova célula (.. esperamos que você comece a entender qual o melhor momento para criar uma nova célula, repare que a

chamada de execução por célula gera uma unidade de processamento apenas daquela célula, porém, aproveitando as variáveis de células anteriores)

- Use o spark.sparkContext para carregar o arquivo 100M.data
- Conte o número de linhas que tem a palavra “diamond”
- Verifique no Spark UI quantas tasks foram usadas

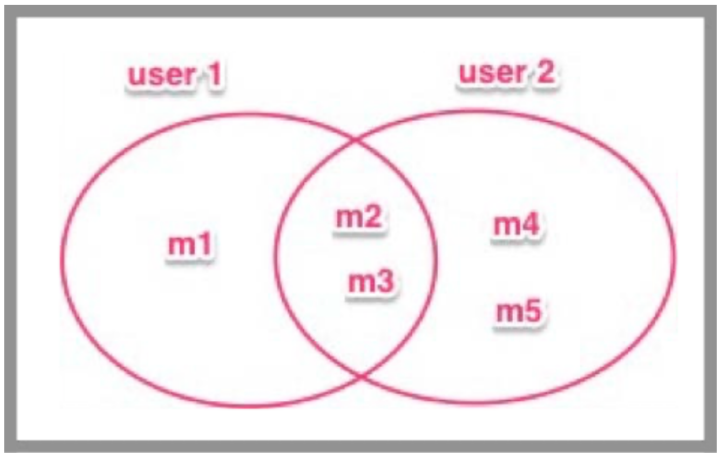


- Conte agora o número de linhas que **NÃO** tem a palavra “diamond”
- Agora verifique se o número de linhas que NÃO tem com os que tem equivale ao número total de linhas do arquivo
- Repita o mesmo processo para os arquivos 500M.data e 1G.data
    - Note o tempo e número de tasks que serão usados em cada um do caso
    - Clique no “link” count para ver detalhes dos Stages
    - Repare em como o arquivo vou quebrado na coluna “Input Size / Record” da seção Tasks
- Vamos carregar agora múltiplos arquivos especificando como endereço para os arquivos a expressão “*.data”
- Revise o Spark UI, repare novamente no “Input Size/Records”

Lab 3.2: RDD na prática

Tempo aproximado: 30 – 40 minutos

- Nosso cenário de teste vamos assumir que temos usuários que participam de vários meetups. Começaremos com 2 usuários:
    - User1 atende aos meetups: m1, m2 e m3
    - User2 atende aos meetups: m2, m3, m4 e m5



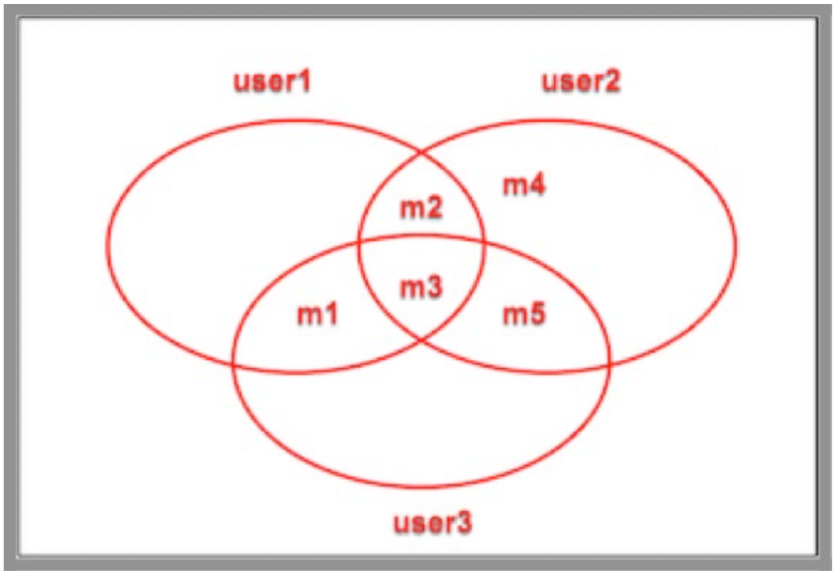
- Cada meetup dos usuários estarão separados por dois RDD. Devemos analizar os dados dos usuários aplicando operações sobre os RDD. Utilizaremos operações como union, intersection, distinct and subtract
- Iniciamos criando os dois RDD que representam os meetups de cada usuário

```
val u1 = spark.sparkContext.parallelize(List("m1", "m2",
"m3"))

val u2 = spark.sparkContext.parallelize(<agora eh com
vcs>)
```

- O que desejamos:
  - Usando os dois RDD's, encontre meetups que são comuns a ambos os usuários
    - Qual operação vc usou para fazer isso?
    - Olhando os Jobs pelo Spark UI você vê algo? (Deveria)
    - Vá nos detalhes do job (clicando na coluna description) e veja a DAG.
    - Clicando nas caixas azuis do diagrama você terá mais detalhes .
  - Encontre meetups que são do usuário User1 e User2
    - Como você fez isso? E se tiver resultados duplicados? Como você remove eles?
    - Olhe a DAG depois de ter feito essas transformações
  - Encontre os meetups atendidos apenas pelo u1.
    - Ou seja, aqueles que o u1 atende mas o u2 não
    - Olhe a DAG

- Cria dois datasets de recomendação de meetups, onde cada usuário recomenda ao outro aquele que outro ainda não atende
  - u1 recomendará para o u2 : m1
  - u2 recomandará para o u1 : m4 e m5
  - Como vc pode fazer isso? Que operações são necessárias?
  - Olhe a DAG
- Recomendações com 3 Usuários
  - Vamos contar agora com mais um usuário o u3:
    - u3 atende aos meetups: m1, m3 e m5



- Considere fazer recomendações considerando os seguintes requerimentos:
  - O usuário não deve estar atendendo aquele meetup para pode ser recomendado a ele.
  - O meetup deve ser atendido aos outros dois usuários para que possa ser recomendado a ele
  - A resposta esperada é:
    - U1: m5
    - U2: m1
    - U3: m2
  - Olhe a DAG depois de feito.