Qual o objetivo do comando **cache** em Spark?

Guardar resultado de consulta na memória. Assim, o resultado de alguma ação pode persistir tornando mais rápidos acessos futuros a estes dados consultados.

O mesmo código implementado em Spark é normalmente mais rápido que a implementação equivalente em MapReduce. Por quê?

Código MapReduce persiste os dados em disco enquanto que código em Spark persiste em memória, tornando o processo mais rápido por isso.

Qual é a função do **SparkContext** ?

Na arquitetura Spark o SparkContext é o ponto de entrada para qualquer funcionalidade. Quando executamos qualquer aplicativo Spark, um programa de driver é iniciado, que tem a função principal e seu SparkContext é iniciado aqui. Resumindo é o orquestrador do Spark.

Explique com suas palavras o que é **Resilient Distributed Datasets** (RDD).

São os elementos executados em vários nós para fazer o processamento paralelo em clusters. Depois de criar um RDD, você não pode alterá-lo. Os RDDs são tolerantes a falhas; em caso de falha, eles se recuperam automaticamente. Em um Job é possível realizar varias execuções em RDD.

GroupByKey é menos eficiente que **reduceByKey** em grandes dataset. Por quê?

Ao utilizar GroupByKey em um conjunto de dados de pares (i, j), os dados são embaralhados de acordo com o valor da chave i em outro RDD. Nessa transformação, muitos dados desnecessários são transferidos pela rede causando uso desnecessário de memória, acontece muito de ocorrer quebra de disco pelo alto volume de dados. Ao contrario do ReduceByKey que realiza redução dos valores iguais em cada partição antes de realizar a operação de agregação para o resultado final.

Explique o que o código Scala abaixo faz.

```
1- val textFile = sc.textFile ( "hdfs://..." )
2- val counts = textFile.flatMap ( line => line.split ( " " ))
3-        .map ( word => ( word , 1 ))
4-       .reduceByKey ( _ + _ )
5- counts.saveAsTextFile ( "hdfs://..." )
```

1: criar valores para textFile: a partir da leitora do arquivo no formato texto de origem hdfs (Hadoop);

2*: criar valores para counts: contagem das palavras nas variável textFile separadas por espaço;

3*: mapeamento por palavras;

4*: agregação das palavras iguais com soma;

5: salva no formato texto em hdfs (Hadoop) os valores da soma das palavras agregadas em counts.

(*fazem parte da mesma linha de comando)

Questões

Responda as seguintes questões devem ser desenvolvidas em Spark utilizando a sua linguagem de preferência.

1. Número de hosts únicos.

Número de hosts únicos é 137.978;

2. O total de erros 404.

total de erros 404 é 20.901;

3. Os 5 URLs que mais causaram erro 404.

ts8-1.westwood.ts.ucla.edu/images/Nasa-logo.gif - 37
nexus.mlckew.edu.au/images/nasa-logo.gif - 34
203.13.168.17/images/nasa-logo.gif - 25
203.13.168.24/images/nasa-logo.gif - 25
onramp2-9.onr.com/images/nasa-logo.gif – 22

4. Quantidade de erros 404 por dia.

01/Jul/1995: 316, 02/Jul/1995: 291, 03/Jul/1995: 474, 04/Jul/1995: 359, 05/Jul/1995: 497, 06/Jul/1995: 640, 07/Jul/1995: 570, 08/Jul/1995: 302, 09/Jul/1995: 348, 10/Jul/1995: 398, 11/Jul/1995: 471, 12/Jul/1995: 471, 13/Jul/1995: 532, 14/Jul/1995: 413, 15/Jul/1995: 254, 16/Jul/1995: 257, 17/Jul/1995: 406, 18/Jul/1995: 465, 19/Jul/1995: 639, 20/Jul/1995: 428, 21/Jul/1995: 334, 22/Jul/1995: 192, 23/Jul/1995: 233, 24/Jul/1995: 328, 25/Jul/1995: 461, 26/Jul/1995: 336, 27/Jul/1995: 336, 28/Jul/1995: 94, 01/Aug/1995: 243, 03/Aug/1995: 304, 04/Aug/1995: 346, 05/Aug/1995: 236, 06/Aug/1995: 373, 07/Aug/1995: 537, 08/Aug/1995: 391, 09/Aug/1995: 279, 10/Aug/1995: 315, 11/Aug/1995: 263, 12/Aug/1995: 196, 13/Aug/1995: 216, 14/Aug/1995: 287, 15/Aug/1995: 327, 16/Aug/1995: 259, 17/Aug/1995: 271,

18/Aug/1995: 256, 19/Aug/1995: 209, 20/Aug/1995: 312, 21/Aug/1995: 305, 22/Aug/1995: 288, 23/Aug/1995: 345, 24/Aug/1995: 420, 25/Aug/1995: 415, 26/Aug/1995: 366, 27/Aug/1995: 370, 28/Aug/1995: 410, 29/Aug/1995: 420, 30/Aug/1995: 571, 31/Aug/1995: 526

5. O total de bytes retornados.

65.524.314.915 bytes