**Questões ​– Prova Teórica**

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

**R.** O comando cache() é utilizado para armazenar o RDD criado no passo anterior, então o Spark não tem de computá-lo em cada uso nas consultas posteriores. O cache() é uma operação lazy, portanto o Spark não armazena imediatamente os dados na memória. Um caching eficaz acelera os algorítimos iterativos e ele permite que esses resultados intermediários de operações lazy possam ser armazenados e reutilizados repetidamente de forma mais rápida.

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

**R.** Segundo a fundação Apache, o Spark permite que aplicações em clusters Hadoop executem até 100 vezes mais rápido que o Hadoop MapReduce. Isso ocorre porque o Spark funciona na memória RAM e não em unidades de disco rígido.

O Hadoop MapReduce lê os dados do cluster, executa uma operação e escreve os resultados de volta no cluster, o que leva tempo.

No entanto, o volume de dados processados ​​também difere: o Hadoop MapReduce é capaz de trabalhar com conjuntos de dados muito maiores do que o Spark.

1. **Qual é a função do SparkContext?**

**R.** O objeto SparkContext representa uma conexão com um cluster do Spark, e pode ser usado para criar RDDs, acumuladores e variáveis ​​de difusão nesse cluster. É através dele também que são passadas as configurações que vão ser utilizadas na alocação de recursos, como memória e processadores.

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

**R.** Ele é o objeto principal do modelo de programação do Spark, pois são nesses objetos que serão executados os processamentos dos dados. Eles são chamados Resilient por serem tolerantes à falha, isto é, são capazes de recomputar partes de dados perdidas devido a falhas nos nós e são Distributed porque podem estar divididos em partições através de diferentes nós em um cluster.

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

**R.** Ao contrário de groupByKey, o reduceByKey não mistura os dados no início. Como ele sabe que a operação de redução pode ser aplicada na mesma partição primeiro, somente o resultado da função de redução é embaralhado na rede. Isso causa uma redução significativa no tráfego pela rede.

1. **Explique o que o código Scala abaixo faz.**

**R.** Word count em Scala

# Cria um RDD a partir de um arquivo lido do HDFS

**val textFile = sc.textFile("hdfs://...")**

# Cria uma coleção de itens separando o conteúdo do RDD com base no delimitador “ “

**val counts = textFile.flatMap(line => line.split(" "))**

# Mapeamento “chave valor” onde a palavra é a chave e o número 1 é um valor

**.map(word => (word, 1))**

# Redução do mapeamento executado no passo anterior agregando e somando o valor correspondente a cada chave

**.reduceByKey(\_ + \_)**

# Persistência do resultado no HDFS em um arquivo no formato de texto

**counts.saveAsTextFile("hdfs://...")**

**Questões ​– Prova Prática**

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

1. **Número​ ​de​ ​hosts​ ​únicos.**

**R.** 137.978

1. **O​ ​total​ ​de​ ​erros​ ​404.**

**R.** 20.901

1. **Os​ ​5​ ​URLs​ ​que​ ​mais​ ​causaram​ ​erro​ ​404.**

**R**. '/pub/winvn/readme.txt', 2004

'/pub/winvn/release.txt', 1732

'/shuttle/missions/STS-69/mission-STS-69.html', 682

'/shuttle/missions/sts-68/ksc-upclose.gif', 426

'/history/apollo/a-001/a-001-patch-small.gif', 384

1. **Quantidade​ ​de​ ​erros​ ​404​ ​por​ ​dia.**

**R.** 360,34

1. **O​ ​total​ ​de​ ​bytes​ ​retornados.**

**R.** 65.524.314.915