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

R.: Comando para salvar o dado na memória apenas, gerando iteração eficiente com baixo tempo de latência. Pode ser utilizado quando os dados são acessados de forma recorrente tanto em datasets pequenos com poucos registros como também em grandes datasets.

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

R.: Devido ao Spark introduzir uma nova forma de processamento onde os dados são distribuídos em memória com significativa melhoria de performance próxima de tempo real.

**Qual é a função do SparkContext?**

R.: SparkContext é o portão de entrada da Spark API (execution engine), sendo assim o ponto principal de toda aplicação Spark. Uma vez que o Spark Shell instancia um contexto e o habilita com o nome sc (por convenção), o mesmo pode ser utilizado para criar RDDs e acessar recursos e serviços Spark.

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

R.: RDD é a unidade fundamental do modelo de programação do Spark e representa uma coleção de objetos distribuídos entre os nodes. Nesses objetos serão executados os processamentos dos dados, geralmente na memória principal. Podem estar armazenados em sistemas de arquivo tradicional, no HDFS e em alguns Banco de Dados NoSQL, como Cassandra e HBase.,

Resilient: se os dados em memória são perdidos, eles podem ser recriados.

Distributed: armazenado em memória através do cluster.

Dataset: dados iniciais podem vir de um arquivo ou serem criados por meio de um programa.

Existem três formas de criar um RDD: de um arquivo ou conjunto de arquivos, de dados na memória ou de outro RDD.

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

R.: Em reduceByKey, os pares na mesma máquina com a mesma chave são combinados antes que os dados sejam embaralhados (shuffle). Em seguida, a função é chamada novamente para reduzir todos os valores de cada partição e produzir um resultado final. Por outro lado, em groupByKey, todos os pares de valores-chave são embaralhados (shuffle). Assim, muitos dados desnecessários são transferidos pela rede gerando uma sobrecarga para transferência entre partições.

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

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

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

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

.reduceByKey (_+_)

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

R.: Neste trecho de código para contagem de palavras, algumas funções de transformação (map, reduceByKey) são utilizadas para gerar um dataset de pares (String, Int) denominado counts, onde o resultado final será salvo em um arquivo texto no HDFS.

**HTTP requests to the NASA Kennedy Space Center WWW server**

**Fonte oficial do dateset : http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html**

**Dados :**

**- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed , 205.2 MB.**

**- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed , 167.8 MB.**

**Sobre o dataset : Esses dois conjuntos de dados possuem todas as requisições HTTP para o servidor da NASA Kennedy Space Center WWW na Flórida para um período específico.**

**Os logs estão em arquivos ASCII com uma linha por requisição com as seguintes colunas:**

**- Host fazendo a requisição . Um hostname quando possível, caso contrário o endereço de internet se o nome não puder ser identificado.**

**- Timestamp no formato "DIA/MÊS/ANO:HH:MM:SS TIMEZONE"**

**- Requisição (entre aspas)**

**- Código do retorno HTTP**

**- Total de bytes retornados**

**Questões**

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

**1. Número de hosts únicos.**

**2. O total de erros 404.**

**3. Os 5 URLs que mais causaram erro 404.**

**4. Quantidade de erros 404 por dia.**

**5. O total de bytes retornados.**

R.: spark-nasa-example.py

```
#! python

import sys

from pyspark import SparkContext, SparkConf

from pyspark.sql import functions

from pyspark.sql import Row

from pyspark.sql.types import *

// create Spark context with Spark configuration

conf = SparkConf().setAppName("NASA Example")

sc = SparkContext(conf=conf)

// read input text file to RDD

lines = sc.textFile("file:///home/cloudera/access_log_Jul95")

// Specify Schema

parts = lines.map(lambda l: l.split(" "))

nasa = parts.map(lambda p:
Row(host=p[0],tx=p[1],ty=p[2],timestamp=p[3],req=p[4],cod=p[5],bytes=int(p[6].strip())))

schemaString = "host tx ty timestamp req cod bytes"

fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split()]

schema = StructType(fields)

nasadf = spark.createDataFrame(nasa, schema)

// Numero de hosts unicos

nasadf.sumDistinct(host)

// O total de erros 404

nasadf.select(when(nasadf.cod == 404, nasadf.cod + 1)).collect()

// Os 5 URLs que mais causaram erro 404

nasadf.sort(col("host").desc()).limit(5).show()

// Quantidade de erros 404 por dia

nasadf = spark.createDataFrame(nasadf.timestamp.substr(1, 12)).toDF("date", "val")

w = nasadf.select(when(nasadf.cod == 404, nasadf.cod + 1)).agg(sum("val"))

w.collect()

//O total de bytes retornados

nasadf.groupBy().sum('bytes').collect()
```