Prezado candidato.

Gostaríamos de fazer um teste que será usado para sabermos a sua proficiência nas habilidades para a vaga. O teste consiste em algumas perguntas e exercícios práticos sobre Spark e as respostas e códigos implementados devem ser armazenados no GitHub. O link do seu repositório deve ser compartilhado conosco ao final do teste.

Quando usar alguma referência ou biblioteca externa, informe no arquivo README do seu projeto. Se tiver alguma dúvida, use o bom senso e se precisar deixe isso registrado na documentação do projeto.

Qual o objetivo do comando **cache** em Spark?
Carregar em memória o RDD e disponibilizá-lo para as operações.

O mesmo código implementado em Spark é normalmente mais rápido que a implementação equivalente em MapReduce. Por quê?
Em Spark os dados são cacheados em memória através do RDD. Cada operação pode disponibilizar seus resultados também em memória nas estruturas RDD que ficam disponíveis para as próximas operações. Já MapReduce, os resultados de cada operação devem ser armazenados em disco para que possam ser disponibilizadas para a próxima operação MapReduce, o que gera uma sobrecarga de acesso em disco, aumentando significativamente o tempo de processamento.

Qual é a função do **SparkContext**?
Obter os dados a serem processados e criar o RDD.

Explique com suas palavras o que é **Resilient Distributed Datasets** (RDD).
É a estrutura de dados fundamental do Spark. É uma coleção imutável de objetos que representa os dados carregados em memória e que serão utilizados no processamento. Por ser imutável, pode ser usada para computação distribuída, ou seja, como os objetos não poderão sofrer alteração nos seus dados, essa coleção pode ser particionada e processada em vários nós que fazem parte do cluster, o que garante que o resultado final será sempre o mesmo, independente do número de nós que foram utilizados.

GroupByKey é menos eficiente que **reduceByKey** em grandes dataset. Por quê?
No reduceByKey, os pares <key,value> são agrupados em cada nó antes de serem enviados para o agrupamento final. Já no groupByKey, os pares <key,value> são enviados como são obtidos, gerando assim uma quantidade transferida de dados pela rede muito maior. Por ser grande a massa de dados processada em cada nó, o número de pares <key,value> gerados também pode causar um problema de espaço em memória no nó ou, se configurado, uma sobrecarga de processamento em disco para armazenar os pares obtidos.

Explique o que o código Scala abaixo faz.

```
val textFile = sc.textFile("hdfs://...") carrega os dados a partir de um arquivo texto em uma coleção imutável
val counts = textFile.flatMap(line => line.split(" "))lê linha a linha da coleção e cria um Map com todas as palavras encontradas
             .map(word => (word, 1)) cria pares <key, value> onde key é uma palavra da coleção e value é 1
             .reduceByKey(_ + _) faz o agrupamento dos pares por key somando value; equivalente a expressão lambda (x,y)=>x+y
counts.saveAsTextFile("hdfs://...")grava o Map resultante (counts) em disco no formato de um arquivo texto
```

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.

```
número de hosts únicos: 112
total de erros 404: 20901
top 5 url com mais erros 404:
  hoohoo.ncsa.uiuc.edu -> 251
  piweba3y.prodigy.com -> 157
  jbiagioni.npt.nuwc.navy.mil -> 132
  piweba1y.prodigy.com -> 114
  www-d4.proxy.aol.com -> 91
bytes retornados: 369594524
```

```
erros 404 por dia:
  01/Jul/1995 -> 316    16/Jul/1995 -> 257    04/Aug/1995 -> 346    19/Aug/1995 -> 209
  02/Jul/1995 -> 291    17/Jul/1995 -> 406    05/Aug/1995 -> 236    20/Aug/1995 -> 312
  03/Jul/1995 -> 474    18/Jul/1995 -> 465    06/Aug/1995 -> 373    21/Aug/1995 -> 305
  04/Jul/1995 -> 359    19/Jul/1995 -> 639    07/Aug/1995 -> 537    22/Aug/1995 -> 288
  05/Jul/1995 -> 497    20/Jul/1995 -> 428    08/Aug/1995 -> 391    23/Aug/1995 -> 345
  06/Jul/1995 -> 640    21/Jul/1995 -> 334    09/Aug/1995 -> 279    24/Aug/1995 -> 420
  07/Jul/1995 -> 570    22/Jul/1995 -> 192    10/Aug/1995 -> 315    25/Aug/1995 -> 415
  08/Jul/1995 -> 302    23/Jul/1995 -> 233    11/Aug/1995 -> 263    26/Aug/1995 -> 366
  09/Jul/1995 -> 348    24/Jul/1995 -> 328    12/Aug/1995 -> 196    27/Aug/1995 -> 370
  10/Jul/1995 -> 398    25/Jul/1995 -> 461    13/Aug/1995 -> 216    28/Aug/1995 -> 410
  11/Jul/1995 -> 471    26/Jul/1995 -> 336    14/Aug/1995 -> 287    29/Aug/1995 -> 420
  12/Jul/1995 -> 471    27/Jul/1995 -> 336    15/Aug/1995 -> 327    30/Aug/1995 -> 571
  13/Jul/1995 -> 532    28/Jul/1995 -> 94     16/Aug/1995 -> 259    31/Aug/1995 -> 526
  14/Jul/1995 -> 413    01/Aug/1995 -> 243    17/Aug/1995 -> 271
  15/Jul/1995 -> 254    03/Aug/1995 -> 304    18/Aug/1995 -> 256
```