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?

Cache permite que um RDD permaneça registrado na memória (caso haja espaço o suficiente) após ser processado para que possa ser reaproveitado em actions futuras feitos por uma aplicação de Spark, tendo como objetivo dispensar reprocessamentos da mesma informação.
Dessa forma, é possível realizar uma action em cima do mesmo DataFrame sem ter que recarregá-lo novamente.

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

Após ler os dados que precisa no disco (ou em um sistema como o HDFS), o Spark trabalha com RDDs, que conseguem armazenar o estado da memória como um objeto entre os jobs realizados por uma aplicação, o que permite que esse objeto fique muito mais acessíavel e que os jobs ocorram de maneira mais rápida.
O MapReduce, por sua vez, escreve o dataset completo no HDFS após completar todo um job, o que torna suas operações custosas.

Qual é a função do **SparkContext**?

SparkContext é o ponto de entrada para uma aplicação de Spark que atua como um client para a execução da aplicação, funcionando dessa forma como um master para a mesma. Esse método é criado a partir de uma configuração passada para ele (SparkConf) e, após criado, irá permitir que sua aplicação converse com um gerenciador de recursos (como o Yarn) e trabalhe em um ambiente clusterizado para criar seus executores.

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

RDD é uma abstração coleção de objetos imutáveis e resilientes computados de forma lazy que o Spark utiliza para poder paralelizar os seus processos.

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

GroupByKey une todos os elementos com uma mesma chave que estejam dentro de um Dataset (com elementos de chave-valor) e os enviam, através da rede, para outros executores fazerem a operação de reduce em seu método. Com uma quantidade grande de dados isso não só sobrecarrega a rede mas também pode onerar as tasks dos executores que farão a redução ao final do processo, podendo até mesmo fazer com que estes comecem a utilizar os discos de suas máquinas (o que é indesejável).

ReduceByKey, ao contrário, primeiro faz uma operação de reduce nos elementos do seu RDD para enviar apenas um output para os executores finais, que então farão uma operação final de reduce com os dados que receberam. Dessa forma, diferentemente do que ocorre com o groupByKey, Isso não apenas permite com que menos dados sejam enviados em massa para alguns executores (e os sobrecarregue) como também diminui a quantidade de transformações que eles teriam que fazer, diminuindo a chance de utilizar recursos de disco.

Explique o que o código Scala abaixo faz.

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

O código acima lê um arquivo salvo no HDFS, faz uma chave-valor para cada palavra do arquivo para fazer um count e, em seguida, salvar o resultado em um outro arquivo 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.

   R: 137979

2. O total de erros 404.

   R: 20871

3. Os 5 URLs que mais causaram erro 404.

```
+-----------------------------------------------------------+-----+
|URLs                                                       |count|
+-----------------------------------------------------------+-----+
|GET /pub/winvn/readme.txt HTTP/1.0                         |2004 |
|GET /pub/winvn/release.txt HTTP/1.0                        |1732 |
|GET /shuttle/missions/STS-69/mission-STS-69.html HTTP/1.0|682  |
|GET /shuttle/missions/sts-68/ksc-upclose.gif HTTP/1.0    |426  |
|GET /history/apollo/a-001/a-001-patch-small.gif HTTP/1.0 |384  |
+-----------------------------------------------------------+-----+
```

4. Quantidade de erros 404 por dia.

```
+-----------+-----+
|       Data|count|
+-----------+-----+
|02/Jul/1995|  291|
|21/Aug/1995|  305|
|06/Aug/1995|  373|
|16/Jul/1995|  257|
|07/Aug/1995|  537|
|11/Aug/1995|  263|
|27/Jul/1995|  336|
|07/Jul/1995|  569|
|17/Jul/1995|  406|
|15/Jul/1995|  254|
|18/Jul/1995|  465|
|26/Jul/1995|  336|
|03/Aug/1995|  303|
|18/Aug/1995|  256|
|17/Aug/1995|  271|
|14/Aug/1995|  287|
|10/Jul/1995|  398|
|04/Jul/1995|  359|
|20/Aug/1995|  312|
|20/Jul/1995|  428|
+-----------+-----+
only showing top 20 rows
```

5. O total de bytes retornados.

   R: 65524319796