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?
Melhorar a eficiência do código, o comando cache permite que o resultado das operações possa ser utilizado repetidamente.

O mesmo código implementado em Spark é normalmente mais rápido que a implementação equivalente em MapReduce. Por quê?
O MapReduce leva mais tempo para ler e escrever os dados pois realiza a leitura do disco em cada query executada devido a isso tende a ser mais lento que o Spark que não possui essa restrição, pois pode realizar a leitura/trafego dos dados em cache via memória.

Qual é a função do **SparkContext**?
O SparkContext tem a função de estabelecer a conexão com o ambiente Spark “Clusters”, depois de criado podemos usar para criar RDDs, variáveis de transmissão para a execução de trabalhos.

Explique com suas palavras o que é **Resilient Distributed Datasets** (RDD).
RDD é o conjunto de instruções parametrizadas no SparkContext para a distribuição de dados em partições que permite o processamento e a gravação em paralelo

GroupByKey é menos eficiente que **reduceByKey** em grandes dataset. Por quê?
Porque o GroupByKey realiza a mesclagem dos valores para as chaves após o Shuffle (embaralhamento dos dados) trafegando dados desnecessários pela rede podendo gerar o problema de falta de memória.

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://...")
```

O código acima está fazendo a leitura de um arquivo de texto do HDFS e retornando como um RDD quebrando o arquivo em linhas ou subconjuntos em seguida fazendo a contagem das palavras de cada subconjunto e salvando os dados em novo arquivo de 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.