Qual o objetivo do comando cache em Spark?

O comando cache no *Spark* tem como objetivo otimizar as consultas de dados, pois estes estarão na memória, não sendo necessário busca-los em disco, reduzindo o tempo quando forem acessados. Esse desempenho é melhor percebido quando usamos estes conjuntos de dados repetidamente, como por exemplo, em algoritmos interativos.

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

Geralmente as implementações *Spark* são mais rápidas que implementações correspondentes em *MapReduce* por que o seu processamento é executado em memória e o *MapReduce* precisa de leitura de disco rígido, essa diferença aumenta quando se trada de algoritmos interativos.

Também podemos considerar a característica do RDD, lazy evaluation, na qual contribui para um melhor desempenho.

Além do mais o *MapReduce* é mais lento para inicializar as tarefas que o *Spark*.

Qual é a função do SparkContext?

O SparkContext tem como função, ser uma conexão do algoritmo a ser desenvolvido com os recursos oferecidos pelo Apache Spark, ou seja, através dele podemos criar RDD, acumuladores e variáveis broadcast.

Explique com suas palavras o que é Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset é a principal estrutura do Apache Spark, é uma coleção de objetos que são distribuídos e processados em diferentes nós do cluster. Esta estrutura é tolerante a falha, ou seja, se tiver alguma partição perdida ou danificada devido uma falha no nó ela será reprocessada, também armazena resultados intermediários em memória distribuída em vez de disco e

pode ser configurada uma estratégia de persistência, seja em memória ou armazenando em disco.

Outra característica é ser de avaliação preguiçosa *(lazy evaluation*), pois não computam os seus resultados de imediato e sim quando os seus resultados são apresentados ou persistidos.

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

O GroupByKey é menos eficiente que o ReduceByKey em grande datasets, por que realiza primeiro o suffle para depois organizar os pares de chave e valor, transferindo desnecessariamente pela rede um conjunto de dados maior e, além disso, o Spark envia os dados para o disco quando há mais dados num único executor do que os que podem caber na memória, prejudicando ainda mais o desempenho.

Por outro lado, o ReduceByKey combina todas as chaves antes de realizar o suffle, enviando pela rede apenas o resultado da função de redução.

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://...")
```

Este código Scala é um algoritmo para contagem de palavras, onde busca um arquivo texto localizado no HDFS e após a contagem o mesmo salva em um arquivo texto, também no HDFS.

Este algoritmo funciona em três passos, sendo:

1º Passo: Efetua a leitura do conteúdo de um arquivo de texto como uma coleção de linhas localizado no sistema de arquivo HDFS, utilizando o *SparkContext* para atribuir esta coleção a uma variável textFile, um *RDD*.

2º Passo: Realiza um mapeamento de palavras, linhas que foram separadas por espaços em branco pela função *split*, através da função *flatMap*.

Para aplicar a função *split* fui utilizado o recurso da expressão lambda para navegar pelas linhas da coleção.

Posteriormente é realizado outro mapeamento através da função *map*, retornando um *RDD* contendo chave-valor. Usando novamente a expressão lambda para trazer a palavra e atribuir a cada uma delas o valor 1.

Por fim este passo termina utilizando a função *reduceByKey*, que em cada nó do cluster é agrupado pela chave e seus valores somados, sendo um resultado parcial, que oferece uma redução dos dados, transfere esses dados reduzidos pela rede e por fim agrupados novamente pela chave e somando seus valores.

3º Passo: No Final *RDD* contendo o par de palavras e contagens através da função saveASTextFile é salvo em um arquivo de texto localizado no sistema de arquivo do HDFS