Qual o objetivo do comando “cache” em Spark?

Um dos recursos mais importantes do Spark que não existe no MapReduce, Comando serve para salvar dados apenas na memória, isso pode ajudar em casos onde precisamos de acesso rápido a informação em passos subsequentes e não necessitamos de níveis de persistências diferentes de “MEMORY\_ONLY”.

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

O MapReduce utiliza o HDFS para armazenamento de suas operações, apesar de ser muito rápido comparado com outras tecnologias o Spark utiliza a memória do cluster para seus processamentos, dessa forma a velocidade de leitura e gravação supera consideravelmente o MapReduce.

Qual é a função do SparkContext?

Ponte entre a aplicação e o conjunto de serviços do Spark, também pode ser considerado como coração de uma solução Spark e podemos afirmar que uma aplicação é realmente uma aplicação Spark quando o SparkContext é utilizado direta ou indiretamente.

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

Basicamente a unidade fundamental do Spark, Uma coleção de objetos distribuídos que são imutáveis.

Tem como características principais, ser resiliente onde se por ventura os dados em memória forem perdidos eles podem ser recriados, são armazenados em memória através do cluster e pode conter uma coleção de [dados particionados](https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-partitions.html) com valores primitivos ou valores de valores, por exemplo, tuplas ou outros objetos.

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

Quando utilizamos um groupByKey em um RDD, os dados nas partições são embaralhados na rede para formar uma chave e uma lista de valores, assim causando um grande consumo de recursos. Agora quando utilizamos um reduceByKey em um RDD ele retorna um conjunto de dados de pares onde os valores de cada chave são agregados usando a função de redução fornecida, assim capaz de receber argumentos de algum tipo e retornar o mesmo tipo de dados de resultado. Como em groupByKey porem com uma redução significativa no tráfego pela rede.

Explique o que o código Scala abaixo faz.

O código é um contador de palavras, abaixo segue comentários do que o código está fazendo.

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

/\*Carrega os dados do HDFS em um RDD chamado textfile\*/

val counts = textFile.flatMap(line => line.split(" ")) .map(word => (word, 1)) .reduceByKey(\_ + \_)

/\*

1).flatMap: pegamos o RDD de linhas e o transformamos em um RDD de palavras

2).map: transformamos o RDD de palavras em RDD de tuplas (Ex: palavra, 1 )

3).reduceByKey: para cada chave (word) reduzimos todos os valores somando todos os valores juntos.

\*/

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

/\*

Salvamos o arquivo no HDFS

\*/