Qual o objetivo do comando cache em Spark?

RDD's são a principal abstração de dados em Spark e em geral as operações em RDD's são do tipo "lazy", ou seja, essa operações só são avaliadas depois de terem os valores obtidos do RDD. Em códigos com muitas iterações é aconselhável utilizar o comando cache em Spark pois ele calcula resultados intermediários que podem ser reutilizados e armazenados repetidamente. O objetivo do comando cache em Spark é possibilitar nesse tipo de operação uma maior eficiência.

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

Dois fatores princiapais estão associados a isso. Com MapReduce o resultado de cada job é escrito em disco e deve ser lido novamente para dar prosseguimento quando existe uma sequência de jobs. Spark atráves do cache reduz significativamente a necessidade de escrita e leitura em disco, e, redução de escrita e leitura em disco acarreta ganho de performance. Outro fator é a JVM que em MapReduce precisa ser iniciada a cada job enquanto em Spark a execução da JVM é constante.

Qual é a função do SparkContext ?

O SparckContext é a interface entre o driver e os recursos, funciona com um cliente do ambiente de execução Spark. Por ele são configuradas memória e processadores por exemplo. Além disso é também utilizado para criar RDD's, criar variáveis de broadcast, criar acumuladores e executar tarefas.

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

RDDs são abstrações de dados que podem ser manipulados pelo spark, dentre outras é a principal do spark. O "R" vem de resiliente pois são tolerantes a falhas nos nós. O primeiro "D" vem de Distributed por estarem divididos em partições. RDD's são imutáveis, só podem ser lidos. As operações realizadas em RDD's podem ser realizadas de forma paralela e são do tipo "lazy", ou seja, os dados só são transformados se alguma ação é executada.

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

Apesar de ambos teoricamente produzirem o mesmo resultado, isso pode não acontecer no caso de grandes datasets com limitação de memória ou se dar de forma mais lenta. Isso

ocorre pois utilizando reduceByKey a operação é passada como parâmetro em todos os elementos de mesma chave em cada partição para obter um resultado parcial antes de calcular o resultado final que fica a cargo dos executores. Já no caso de groupByKey o calculo dos resultados parciais não é realizado anteriormente e um volume grande de dados pode ser transferido, dependendo do tamanho do dataset. Como existem limitações de memória e isso acarretaria em ter que escrever os dados em um disco a morosidade do processo aumenta.

Explique o que o código Scala abaixo faz:

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

Primeiramente na linha (1) arquivo de texto é lido e armazenado na variável textFile.

Na linha (2) começa a configuração da variável counts. O arquivo de texto é separado por linhas onde cada linha é uma palavra encontrada. O separador das palavras, o que determina uma palavra nesse contexo é o espaço (“ “).

Na linha (3) cada palavra é transformada em um mapeamento com chave igual a própria palavra e valor 1.

Na linha (4) o comando reduceByKey faz a agregação dessas palavras pela chave, através da operação soma. Como a chave é única, isso resulta na contagem das palavras do arquivo.

Na linha (5) a variável counts é salva em formato texto.