

Claudio Marcelino

1.Qual o objetivo do comando cache em Spark?

Carregar os dados em memória, assim qualquer análise a partir daí não precisará reler e transformar novamente o arquivo original. É ótimo para ETL com pequenos e médios datasets.

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

A principal diferença entre eles está na abordagem do processamento: o Spark pode fazer isso na memória, enquanto o Hadoop MapReduce precisa ler e gravar em um disco.

3.Qual é a função do SparkContext?

Principal ponto de entrada para executar as funcionalidade Spark. Quando executamos qualquer aplicativo Spark, um programa de driver é iniciado, que tem a função principal e seu SparkContext é iniciado aqui. O programa do driver executa as operações dentro dos executores nos nós do trabalhador. (#tutorialpoints spark)

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

Os dados distribuídos resilientes (RDD em inglês) servem para processamento paralelo com spark, um mesmo conjunto de dados é copiado para cada operação que está sendo executada disponibilizando a todos os processamentos que forem utiliza-los em paralelo ou reutilizando em processamento futuro (distribuído e armazenando em memória), caso aconteça a falha de algum node ele se recupera automaticamente (isso o torna resiliente). O Spark suporta dois tipos de variáveis compartilhadas: broadcast e acumuladores. Broadcast que são usadas para armazenar em cache um valor na memória em todos os nós. Acumuladores que são as variáveis calculadas.

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

GroupByKey embaralha os dados antes de agrega-los enquanto o reduceByKey não embaralha os dados no início da função, é primeiro feito a redução e agregação e o resultado é embaralhado. tornando o processo mais eficiente.

6.Explique o que o código Scala abaixo faz.

`val textFile = sc.textFile("hdfs://...")` → nesta linha, lê o arquivo de texto no caminho do hdfs hadoop e cria um dataset de resposta chamado textFile,

`val counts = textFile.flatMap(line → line.split(" "))` → nesta linha, cria uma resposta chamado 'counts', com a função flatMap junto da split realiza uma transformação separando todos os caracteres com espaço " ",

`.map(word => (word, 1))` → nesta linha, mapeia todos os caracteres e dizendo que eles valem 1, `.reduceByKey(_ + _)` nesta linha, é feito a redução e agregação por soma de todos os caracteres (1+1),

`counts.saveAsTextFile("hdfs://...")` → nesta linha, é feito a contagem de caracteres e salvo o resultado em um arquivo texto no path do hdfs (hadoop).