QUESTIONÁRIO

Qual o objetivo do comando cache em Spark?

Muitas situações, principalmente quando se trabalha com uma grande quantidade de dados, é interessante salvar resultados parciais e intermediários em memória para que possam ser reutilizados em etapas posteriores. O comando cache em spark permite o armazenamento em um cache em memória. Dessa forma, um pequeno conjunto de dados pode ser acessado repetidamente de forma eficiente (como por exemplo para algoritmos interativos como Random Forest).

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

Spark e MapReduce são soluções populares para o processamento de dados. Porém, eles trabalham de maneira diferente. O Spark pode fazer o processamento de dados em memória (RDDs), já o MapReduce precisa ler e grava em um disco. Tal fato, faz com que o Spark seja mais rápido para um mesmo código implementado em ambos os contextos. Principalmente levando em conta tarefas iterativas, em que em MapReduce há uma sobrecarga significativa.

Qual é a função do SparkContext?

Um SparkContext é um cliente do ambiente de execução do Spark e atua como o mestre de uma aplicação Spark. É o ponto de entrada para os serviços do Apache Spark (mecanismo de execução) e, portanto, o coração de um aplicativo Spark.

O SparkContext configura serviços internos e estabelece uma conexão com um ambiente de execução do Spark. Depois que um SparkContext é criado, pode-se usá-lo para criar RDDs, acumuladores e Broadcast, acessar os serviços do Spark e executar trabalhos (até que o SparkContext seja interrompido).

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

Um RDD é, essencialmente, a representação do Spark de um conjunto de dados, distribuído em várias máquinas, com APIs para permitir que você atue sobre ele (ou seja, é uma coleção distribuída imutável de objetos). Os RDDs podem conter qualquer tipo de objetos (seja em Python, Java ou Scala), até mesmo classes. RDDs podem ser criados a partir de arquivos no HDFS ou de coleções da linguagem Scala**.**

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

Enquanto reducebykey e groupbykey produzem a mesma resposta, o reduceByKey funciona muito melhor em um grande conjunto de dados. Isso porque o Spark sabe que pode combinar a saída com uma chave comum em cada partição antes de “embaralhar” (shuffling) os dados.

Por outro lado, ao chamar groupByKey, todos os pares de valores-chave são embaralhados. Isso representa é um monte de dados desnecessários para serem transferidos pela rede. O groupByKey pode causar problemas de disco quando os dados são enviados pela rede e coletados nos reduce workes.

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://...")
```

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

Abre arquivo de texto, cada elemento do RDD é uma linha do arquivo.

2. ***val counts = textFile.flatMap(line => line.split(" "))***

flatMap é usado para retornar cada palavra (separada por um espaço) na linha como uma array.

3. ***.map(word => (word, 1))***

Para cada palavra atribuir um valor de 1 para que elas possam ser somadas. Cria um tuple (palavra, 1)

4. ***.reduceByKey(_ + _)***

Sobre os tuples gerados na etapa anterior, através de uma agregação (somando), obtém um RDD que conta a quantidade de palavras únicas. Olhando o primeiro elemento do tuple e somando o segundo (1). Exemplo:

(casa,1) (carro,1) (carro,1) (casa,1) (casa,1).

Após reduceByKey (casa,3) (carro,2)

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

Salva o arquivo no HDFS.

Dessa forma essa função escrita em scala faz algumas transformações para construir um conjunto de dados (String, Int), em seguida, salva em um arquivo HDFS.

1. Número de hosts únicos.

Hosts unicos: 137933

2. O total de erros 404

Total de erros 404: 20899

3. Os 5 URLs que mais causaram erro 404

Host	Total
hoohoo.ncsa.uiuc.edu	251
piweba3y.prodigy.com	157
jbiagioni.npt.nuwc.navy.mil	132
piweba1y.prodigy.com	114
	112

4. Quantidade de erros 404 por dia

Dias	Total (erro 404)
1	559
2	291
3	778
4	705
5	733
6	1013
7	1107
8	691
9	627
10	713
11	734
12	667
13	748
14	700
15	581
16	516
17	677
18	721
19	848
20	740
21	639
22	480
23	578
24	748
25	876
26	702
27	706
28	504
29	420
30	571
31	526

5. O total de bytes retornados

Qtd_Bytes	Total_Bytes
3461613	65524314915