1. Qual o objetivo do comando cache() em Spark?

É um mecanismo para acelerar aplicações que acessam o mesmo RDD várias vezes. Um RDD que não é armazenado em cache ou com ponto é reavaliado cada vez que uma ação é invocada nesse RDD. Existem duas chamadas de função para armazenar em cache um RDD: cache() e persist (StorageLevel). A diferença entre eles é que cache() armazenará em cache o RDD na memória, enquanto persist(StorageLevel) pode armazenar em cache na memória ou no disco. Note que persist() sem argumento é equivalente a cache().

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

Uma das principais limitações do MapReduce é que ele persiste o conjunto de dados completo após a execução de cada job. Isso incorre em um grande fluxo de entrada e saída de dados em disco e em rede. Já no Spark, quando a saída de uma operação precisa ser alimentada em outra operação, o Spark passa os dados diretamente sem persistir.

A principal inovação do Spark foi a introdução de uma abstração de cache na memória. Isso torna o Spark ideal para cargas de trabalho em que várias operações acessam os mesmos dados de entrada. Os usuários podem instruir o Spark a armazenar em cache os conjuntos de dados de entrada na memória, para que não precisem ser lidos do disco para cada operação.

Outro fato é que o Spark consegue iniciar tarefas muito mais rapidamente. O MapReduce inicia uma nova JVM para cada tarefa, o que pode levar segundos para carregar JARs, JIT, analisar a configuração XML etc. O Spark mantém uma JVM executora em execução em cada nó.

3. Qual é a função do SparkContext?

O SparkContext configura serviços internos e estabelece uma conexão com um ambiente de execução do Spark. Você pode criar RDDs, acumuladores e variáveis de broadcast, acessar serviços Spark e executar tarefas após a criação do SparkContext.
A primeira etapa de qualquer programa Spark é criar um SparkContext. O SparkContext permite que a aplicação acesse o cluster por meio de um gerenciador de recursos.
Algumas funcionalidades que o SparkContext oferece são:
1. Obter o status atual de um aplicativo Spark, como configuração, nome do aplicativo.
2. Definir a configuração como URL principal, nível de log padrão.
3. Pode-se criar entidades distribuídas como RDDs.

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

RDD é a uma maneira como o dado é armazenado em memória e particionado através de vários clusters para que sejam realizadas transformações mais rapidamente de maneira distribuída e com tolerância a falhas.
Por default, os dados são armazenados em memória mas também podem ser armazenados em disco caso não haja memória suficiente.

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

Ao utilizar reduceByKey, o Spark sabe que pode combinar a saída com uma chave comum em cada partição antes de enviar os dados para outro executor calcular o resultado final. Por outro lado, ao chamar groupByKey todos os pares de valores-chave são enviados. Isso faz com que um volume maior de dados desnecessários sejam transferidos pela rede.

Fica mais fácil entender visualizando os diagramas disponíveis no link a seguir:

https://databricks.gitbooks.io/databricks-spark-knowledge-base/content/best_practices/prefer_reducebykey_over_groupbykey.html

##
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://..." )
```

Segue explicação passo a passo do código:

1. Ler o dado que está armazenado no hdfs. É criado então um RDD de strings chamado textFile.

2. O RDD textfile é composto por cada linha do arquivo de texto inicial alocada por linha.

3. Todas as palavras de cada linha são separadas gerando um RDD de palavras.

4. O RDD de palavras é então transformados num RDD de tuplas do tipo (palavra,1)

5. Então é utilizado reduceByKey para somar as ocorrências de cada palavra gerando tuplas do tipo (palavra, quantidade de ocorrências).

6. Esse resultado é então salvo em forma de texto no hdfs.