**Respostas:**

**Qual o objetivo do comando cache em Spark?**

O objetivo do comando cache em Spark é recomendável quando precisamos acessar o dataframe várias vezes. Desta forma será armazenado em memória ao invés de refazer a leitura várias vezes. Para maiores detalhes é possível visualizar o plano de execução é visualizar que o serviço irá fazer I/O toda vez que for acessar o dado.

Com a habilitação do cache o plano de execução irá mostrar na próxima leitura do dataframe que o recurso utilizado está em cache.

Temos situações onde o cache nativo não é tão eficiente, e funciona bem para pequenos volumes de dados, ou em casos que precisamos armazenar dados intermediários.   
Para casos de particionamento o cache não é tão eficiente. Algumas técnicas como particionamento e outras open source irão possibilitar melhor performance para trabalhos em cache.

Outro ponto importante é a execução através do YARN, pois teremos o spark-submit separados em lotes de execução, portanto o uso eficiente da memória.

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

O código implementado no framework Spark executa análises gerais de dados em clusters através de computação distribuída, como o Hadoop, O spark fornece cálculos na memória para maior velocidade e processamento de dados através do MapReduce. Ele é executado sobre um cluster Hadoop existente, pode acessar o data store Hadoop (HDFS) e pode processar dados estruturados no Hive transmitir dados de fontes como HDFS, portanto os aplicativos nos clusters do Hadoop são executados até 100 vezes mais rápido na memória e 10 vezes mais rápidos quando executados em disco, além da facilidade de pode escrever o código em Java, Scala ou principalmente em PYTHON.

Outra diferença importante é a combinação de SQL, Streaming e Análise Complexa, além de operações simples de Map / REDUCE ( Modo antigo ), o Spark suporta consultas SQL, dados de fluxo contínuo, análises complexas como aprendizado de máquina e algoritmos gráficos prontos para uso.

O Spark foi projetado para executar em diferentes plataformas, como o Hadoop ou Apache Mesos, também pode ser executado de forma independente utilizados os serviços de nuvens. E o principal pode acessar HDFS, o Cassandra, O HBASE , S3 , Hive, e diversas fontes de dados de forma bem mais simples que o equivalente em MapReduce.

**Qual é a função do SparkContext?**

A função do SparkContext é ser o ponto principal de entrada para conexão e execução de qualquer aplicativo Spark em cluster, portanto para melhor entendimento, quando desejamos criar uma instância no ambiente, utilizamos esta Classe.

Com a execução do SparkContext temos a inicialização do drive que roda sobre os executores e work loaders, temos uma lista de parâmetros que podem ser passados para o driver que são úteis para a utilização do serviço.

No SparkContext, temos vários parâmetros de configuração, alguns parâmetros uteis são: Master = url cluster, appName = nome job, sparkHome = Diretório de instalação do Spark.

Com a criação da instância do Spark podemos criar os RDD, e interagir no ambiente, acessar os serviços e executar JOBS.

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

Com a criação do SparkContext é possível criar vários RDD para interagir com fontes diversas de dados, assim temos a abstração de memória distribuída, onde pode se programar grandes clusters de uma forma tolerante a falhas.

Os RDDs são motivados por dois tipos de aplicativos que as atuais estruturas de computação lidam de maneira ineficiente: algoritmos iterativos e ferramentas interativas de mineração de dados.Para os dois cenários manter as informações em memória pode melhorar a performance da aplicação. Outro ponto importante a se destacar é a tolerância a falhas em memória, pois há acesso restrito.

Como o spark force suporte a vários formatos e fontes de dados, com a implementação mais recente Spark, temos novos formatos para abstração de dados que são os Dataframes e Datasets.

Um ponto interessante sobre o RDD nas primeiras implementações é uma alta sobrecarga de Garbage Colletion, e principalmente a utilização de APIs versão 1.x, a menos que se queira criar RDD personalizados, o indicado é utilizar as outras abordagens acima citadas.

Caso o objetivo seja programar com maior flexibilidade e controle nas transformações em baixo nível, RDD pode ser uma boa opção para tratamentos dos datasets.

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

O groupByKey é menos eficiente em grandes datasets, porque ele não faz um merge entre as chaves, assim temos muitos dados enviados para cada partição e com shuffles dos dados nas partições podem ocorrer problemas como a falta de memória.

O GroupbyKey também pode gerar outros problemas em disco quando os dados são enviados pela rede e coletados nos Reducers Works.

Os GroubyKey são apenas para agrupar conjuntos de dados baseados em chaves, enquanto o ReduceByKey temos o agrupamento e agregação além de ser voltado para execução em grandes datasets.

<https://stackoverflow.com/questions/43364432/spark-difference-between-reducebykey-vs-groupbykey-vs-aggregatebykey-vs-combineb>

Explique o que o código Scala abaixo faz.

