Machine Learning e IA em Ambientes Distribuídos Versão 2.0

Arquitetura do Apache Spark

O Spark é o projeto open source mais ativo no mundo atualmente, relacionado ao Big Data, sendo considerado o sucessor do Hadoop MapReduce. Conceitualmente, o Spark é semelhante ao Hadoop MapReduce; ambos são projetados para o processamento paralelo de Big Data. Ambos permitem o processamento de dados em grande escala usando hardware commodity (máquinas de baixo custo).

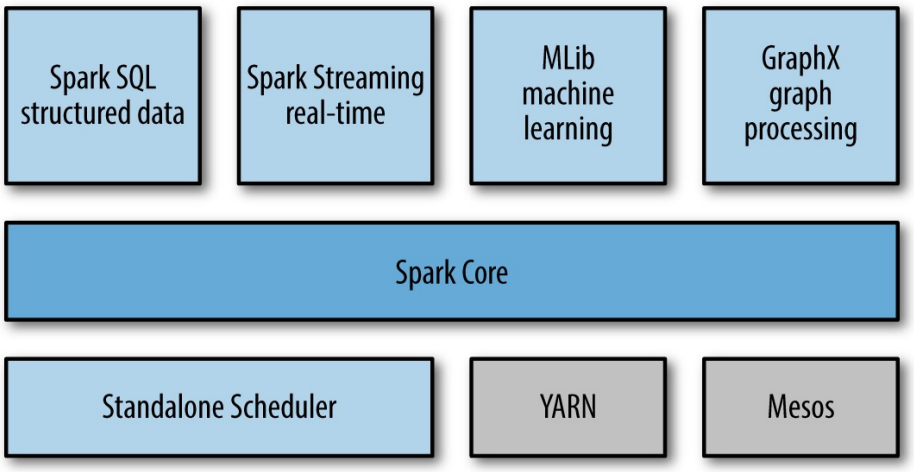
O Apache Spark é um framework de computação em cluster, open-source, para uso geral e distribuído, ou seja, permite operações através de vários computadores (nodes) em um cluster. Uma de suas principais características é o processamento de dados em memória. O Spark pode ser usado como ferramenta ETL, para soluções analíticas em tempo real, para Machine Learning ou processamento gráfico e tudo isso através de carga de dados em batch ou processamento de dados de streaming em tempo real. Vamos agora investigar a arquitetura do Spark.

Existem basicamente 3 perfis de profissionais que vão trabalhar com Spark:

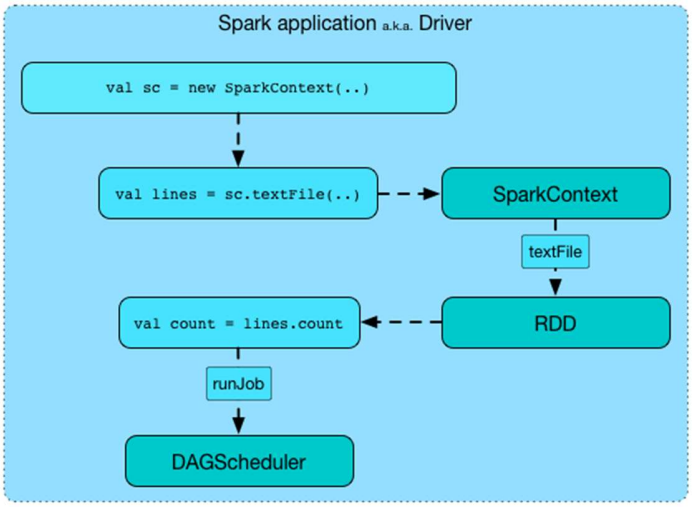
- Cientistas de Dados
- Engenheiros de Dados
- Administradores

O Cientista de Dados é quem desenvolve as aplicações de processamento de dados, utilizando o Spark como framework. O Engenheiro de Dados é quem cuida da infraestrutura e determina como será a organização do cluster. O administrador é quem mantém o serviço funcionando.

O Spark possui um engine central com as operações básicas do framework, o Spark core e as bibliotecas Spark SQL, Streaming, MLLib e GraphX, que desempenham diferentes funções e são intercambiáveis. Ao contrário do Hadoop MapReduce, o Spark realiza boa parte das operações computacionais em memória e exatamente por isso é tão veloz.

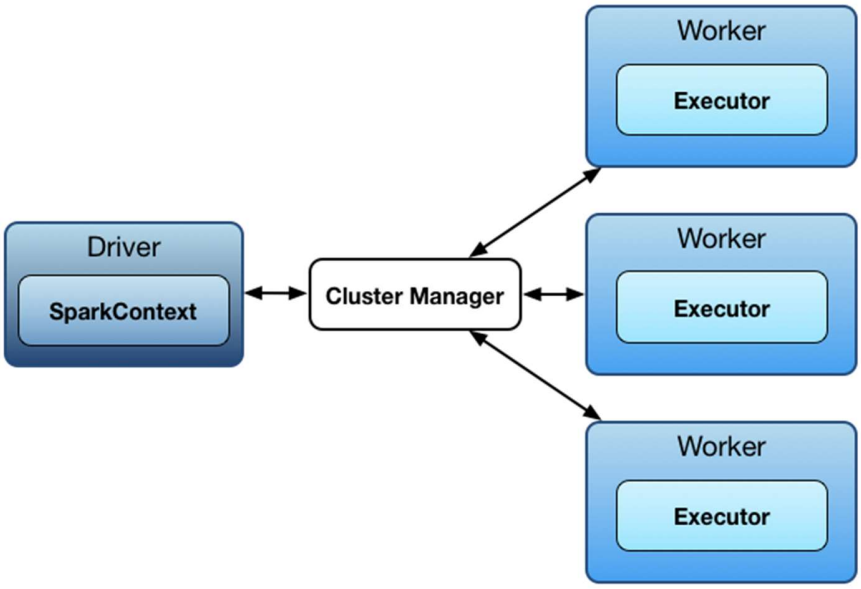


Cada aplicação Spark inicia uma instância de um Spark Context. Sem um Spark Context, nada pode ser feito no Spark. Cada aplicação Spark é uma instância de um Spark context. O Spark context é basicamente uma espécie de cliente que estabele a conexão com o ambiente de execução do Spark e age como o processo principal da sua aplicação.



Com o Spark Context criado, podemos então definir os nossos RDD's ou Dataframes, que são os objetos que vão armazenar os dados.

O Spark utiliza uma arquitetura Master/Worker. Existe o *driver program* que coordena o que chamamos de master. O Master é o chefe que por sua vez, gerencia os workers (os trabalhadores) e cada um possui um executor. Nada muito diferente da vida real, não é mesmo? Vamos ver o que são esses processos.



Cada worker ou slaver é na verdade um computador em um cluster de computadores, ou simplesmente um node. Cada node possui um processo chamado executor, que por sua vez executa tasks, tarefas. Tudo isso é gerenciado por um computador chamado Master ou Cluster Manager.

O Spark Driver é um processo que roda na sua própira máquina virtual java e que é responsável pelo Spark Context. O Spark driver é o que podemos chamar de centro de operações, pois ele quem dispara os jobs e as tarefas que serão executadas nos workers e gerenciadas pelo Master (cluster manager).

O spark driver é responsável pelo SparkContext para cada aplicação Spark. O spark driver é responsável por diversas tarefas, tais como cache manager, security manager, memory manager, etc…Um Spark Driver nada mais é que um aplicativo (escrito em Java, Scala, Python ou R) que usa Spark como uma biblioteca. Ele fornece o código de processamento de dados que o Spark executa em cada node do cluster. Um Spark Driver pode inciar um ou mais trabalhos em um cluster de Spark (que são os jobs). O Spark Driver inicia um job, que será executado nos workers e gerenciado pelo cluster manager, que controla e coordena todas as operações paralelas no cluster. Portanto, Cada execução do Spark driver é um job.

O SparkContext fornece acesso as funcionalidades. Tudo que você fizer no Spark, será feito através de um SparkContext. Ele representa uma conexão para o cluster, sendo usado para criar RDD's ou Dataframes, particionar e distribuir os RDD pelo cluster e gerenciar os Executors. O Spark Context então coleta os resultados e apresenta para o Spark Driver. Ou seja, o Spark Context é a ligação entre o Driver Program (a aplicação que você escreveu em alguma linguagem de programação) e o cluster Spark.

O Spark utiliza um gerenciador de clusters para adquirir recursos de cluster para a execução de um job. Um cluster manager, como o nome indica, gerencia recursos de computação em um cluster de diversos nodes. Ele fornece a programação de baixo nível de recursos de cluster em todos os aplicativos. Ele permite que vários aplicativos possam compartilhar recursos do cluster. É possível utilizar diferentes tipos de cluster manager com Spark, tal como Mesos ou YARN, que também é usado pelo Hadoop. O cluster manager também é chamado de resource manager.

Um worker é um computador (host ou node) e quem fornece CPU, memória e recursos de armazenamento para uma aplicação Spark. Os workers executam as aplicações Spark.

Um executor é um processo JVM (Java Virtual Machine) que o Spark cria em cada worker para uma aplicação. Ele executa o código do aplicativo simultaneamente em vários segmentos, ou seja, de forma paralela. Ele também pode armazenar os dados em cache na memória ou disco. Um executor tem o mesmo tempo de vida que a aplicação para o qual foi criado (ou seja, ele nada mais é do que um processo criado para execução do job. Quando o job termina, o processo também termina).

Uma tarefa é a menor unidade de trabalho que o Spark envia para um executor. Cada tarefa executa alguns cálculos, quer retornam um resultado para um Spark Driver. O spark cria uma tarefa para cada partição de dados (um RDD, o conjunto de dados que vc criou, pode ser particionado para execução em paralelo). Um processo Executor pode executar uma ou mais tarefas simultaneamente. A quantidade de paralelismo é determinada pelo número de partições. Mais partições significam mais dados de processamento de tarefas em paralelo.

Existe claro uma relação entre todos os processos, que podem estar sendo executados em uma mesma máquina ou em diversas máquinas em um cluster. Em resumo, você define um Spark driver, que usa um Spark Context para conectar aos serviços do cluster. O Spark driver envia jobs que são trabalhos de análise de dados por exemplo, que serão processados nos workers. Cada worker possui um serviço executor, que pode processar uma ou mais tarefas em paralelo. A gestão do cluster e da comunicação entre os nodes é feita pelo cluster manager. Ao fim da execucão das tarefas, os workers comunicam o Spark Context que então apresenta o resultado ao Spark driver. Esse é o resultado que por exemplo você vê quando executa um comando no console do Spark ou via pyspark.