Introdução

APACHE Spark

Guia de Estudo iniciante

Willdeglan de Sousa

Sumário

O que é Apache Spark?

O Apache Spark é um mecanismo de análise de código aberto usado para cargas de trabalho de big data. Ele consegue lidar com lotes (Batch), cargas de trabalho de análise e processamento de dados em tempo real (streaming de dados). O Apache Spark começou em 2009 como um projeto de pesquisa na Universidade da Califórnia, Berkeley. Os pesquisadores procuravam uma maneira de acelerar o processamento de jobs nos sistemas Hadoop. Ele é baseado no Hadoop MapReduce e amplia o modelo MapReduce para usá-lo com eficiência em mais tipos de cálculos, o que inclui queries interativas e processamento de stream.

O Spark é inteligente na operação com e dados; dados partições são agregados em um *cluster* de servidores, onde podem ser computados e movidos para um armazenamento de dados (DataWarehouse) diferente ou executados por meio de um modelo analítico. Você não precisará especificar o destino dos arquivos ou os recursos computacionais que precisam ser usados para armazenar ou recuperar arquivos.

Desde o seu lançamento, o Apache Spark, teve uma rápida adoção por empresas em diversos setores. Gigantes da internet, como Netflix, Yahoo e eBay, implementaram o Spark em grande escala, processando coletivamente vários petabytes de dados em clusters de mais de 8.000 nós. Ele se tornou rapidamente a maior comunidade de código aberto em big data, com mais de 1000 colaboradores de mais de 250 organizações.

O ecossistema Spark

O ecossistema Spark inclui cinco componentes principais:

Spark Core

É um mecanismo de processamento de dados distribuído de uso geral. Além disso, há bibliotecas para SQL, processamento de stream, machine learning e computação gráfica, sendo que todas elas podem ser usadas juntas em um aplicativo. O Spark Core é a base de todo um projeto, fornecendo despacho distribuído de tarefas, programação e funcionalidades básicas de I/O.

Spark SQL

É o módulo Spark para trabalhar com dados estruturados que oferece suporte a uma maneira comum de acessar uma variedade de fontes de dados. Ele permite consultar dados estruturados dentro de programas Spark, usando SQL ou uma API DataFrame familiar. O modo de servidor fornece conectividade padrão por meio de conectividade de banco de dados Java ou conectividade aberta de banco de dados.

O Spark Streaming

Facilita a criação de soluções de streaming escalonáveis e tolerantes a falhas. Ele traz a API integrada à linguagem Spark para o processamento de stream, para que você possa escrever jobs de streaming da mesma forma que os jobs em batch. O Spark Streaming oferece suporte a Java, Scala e Python, e apresenta semânticas "exatamente uma vez" com estado, prontas para uso.

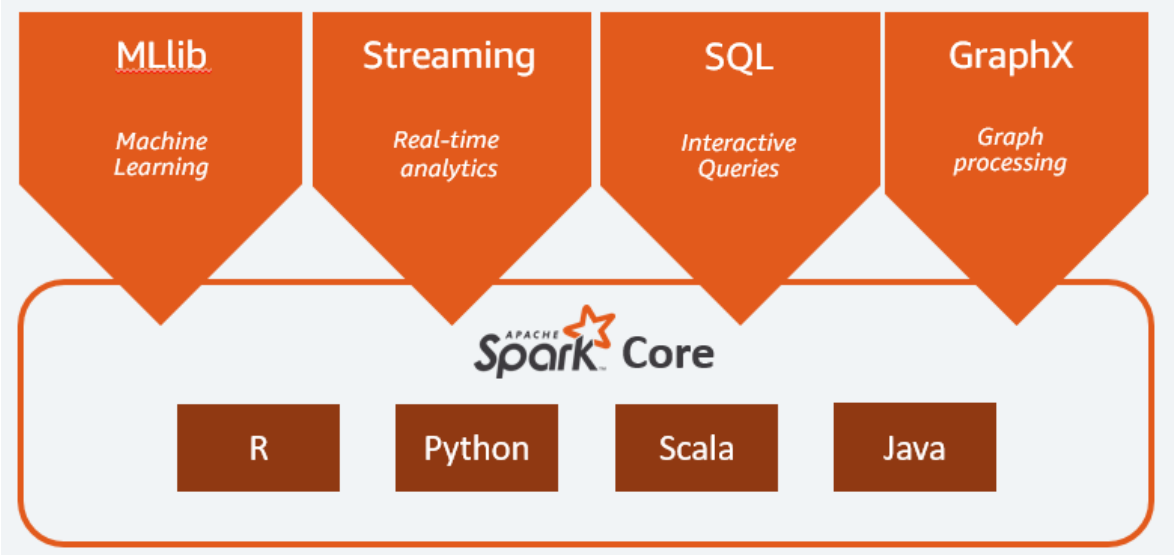
MLlib

É a biblioteca de machine learning escalonável do Spark com ferramentas que tornam a ML prática escalonável e fácil. MLlib contém muitos algoritmos de aprendizado comuns, como classificação, regressão, recomendação e clustering. Também contém fluxos de trabalho e outros

utilitários, incluindo transformações de recursos, construção de pipeline de ML, avaliação de modelo, álgebra linear distribuída e estatísticas.

GraphX

É a API Spark para gráficos e computação paralela a gráficos. É flexível e funciona perfeitamente com gráficos e coleções. Unifica extrair, transformar, carregar, análise exploratória, e computação gráfica iterativa em um sistema. Além de uma API altamente flexível, GraphX vem com uma variedade de algoritmos de gráfico. Ela compete em desempenho com os sistemas gráficos mais rápidos, mantendo a flexibilidade, tolerância a falhas e facilidade de uso do Spark



Benefícios do Apache Spark

Velocidade

Projetado de baixo para cima para melhorar o desempenho, o Spark pode ser 100 vezes mais rápido do que o Hadoop para processamento de dados em grande escala, explorando a computação em memória e outras otimizações. O Spark também é rápido quando os dados são armazenados em disco e, atualmente, detém o recorde mundial de classificação em disco em grande escala.

Facilidade de uso

O Spark tem APIs fáceis de usar para operar em grandes conjuntos de dados. Isso inclui uma coleção de mais de 100 operadores para transformar dados e APIs familiares de data frame para manipulação de dados semiestruturados.

Mecanismo unificado

O Spark vem repleto de bibliotecas de nível superior, incluindo suporte para consultas SQL, streaming de dados, machine learning e processamento de gráficos. Essas bibliotecas padrão aumentam a produtividade do desenvolvedor e podem ser combinadas perfeitamente para criar fluxos de trabalho complexos.

Desvantagens do Apache Spark

Spark consome uma grande quantidade de memória, o que pode causar problemas de desempenho em alguns casos.

Apesar de sua facilidade de uso relativa, ainda existe uma curva de aprendizado para se tornar proficiente no uso do Spark, especialmente para aqueles que são novos no processamento de Big Data.

Não é ideal para processamento de dados em pequena escala: O Spark é projetado para operar em um ambiente de cluster, o que o torna menos adequado para tarefas que envolvem o processamento de pequenas quantidades de dados.

Provisionamento de Hardware

Uma pergunta comum recebida por desenvolvedores do Spark é como configurar o hardware para ele. Embora o hardware certo dependa da situação, fazemos as seguintes recomendações.

Sistemas de Armazenamento

Como a maioria dos trabalhos do Spark provavelmente terá que ler dados de entrada de um sistema de armazenamento externo (por exemplo, o Hadoop File System ou HBase), é importante colocá-lo o mais próximo possível desse sistema. Recomendamos o seguinte:

- Se possível, execute o Spark nos mesmos nós que o HDFS. A maneira mais simples é configurar um cluster de modo autônomo do Spark nos mesmos nós e configurar o uso de memória e CPU do Spark e do Hadoop para evitar interferência (para o Hadoop, as opções relevantes são mapred.child.java.opts para a memória por tarefa e mapreduce.tasktracker.map.tasks.maximum e mapreduce.tasktracker.reduce.tasks.maximumo para número de tarefas). Como alternativa, você pode executar o Hadoop e o Spark em um gerenciador de cluster comum como o Mesos ou o Hadoop YARN.

- Se isso não for possível, execute o Spark em nós diferentes na mesma rede local que o HDFS.

- Para armazenamentos de dados de baixa latência, como o HBase, pode ser preferível executar trabalhos de computação em nós diferentes do sistema de armazenamento para evitar interferência.

Discos locais

Embora o Spark possa executar grande parte de sua computação na memória, ele ainda usa discos locais para armazenar dados que não cabem na RAM, bem como para preservar a saída intermediária entre os estágios. Recomendamos ter de 4 a 8 discos por nó, configurados sem RAID (apenas como pontos de montagem separados). No Linux, monte os discos com a opção noatime de reduzir gravações desnecessárias. No Spark, configure a spark.local.dir variável para ser uma lista separada por

vírgulas dos discos locais. Se você estiver executando o HDFS, não há problema em usar os mesmos discos do HDFS.

Memória

Em geral, o Spark pode rodar bem com qualquer lugar de 8 GiB a centenas de gigabytes de memória por máquina. Em todos os casos, recomendamos alocar no máximo 75% da memória para o Spark; deixe o resto para o sistema operacional e o cache de buffer.

A quantidade de memória que você precisará dependerá do seu aplicativo. Para determinar quanto seu aplicativo usa para um determinado tamanho de conjunto de dados, carregue parte do seu conjunto de dados em um Spark RDD e use a guia Armazenamento da IU de monitoramento do Spark (http://<driver-node>:4040) para ver seu tamanho na memória. Observe que o uso de memória é muito afetado pelo nível de armazenamento e formato de serialização – consulte o guia de ajuste para obter dicas sobre como reduzi-lo.

Por fim, observe que a Java VM nem sempre se comporta bem com mais de 200 GiB de RAM. Se você comprar máquinas com mais RAM do que isso, poderá iniciar vários executores em um único nó. No modo autônomo do Spark, um trabalhador é responsável por iniciar vários executores de acordo com sua memória e núcleos disponíveis, e cada executor será iniciado em uma Java VM separada.

Rede

Em nossa experiência, quando os dados estão na memória, muitos aplicativos Spark são vinculados à rede. Usar uma rede de 10 Gigabits ou mais é a melhor maneira de tornar esses aplicativos mais rápidos. Isso é especialmente verdadeiro para aplicativos de “redução distribuída”, como group-bys, reduce-bys e junções SQL. Em qualquer aplicativo, você pode ver quantos dados o Spark embaralha pela rede a partir da interface de usuário de monitoramento do aplicativo (http://<driver-node>:4040).

Núcleos de CPU

O Spark escala bem para dezenas de núcleos de CPU por máquina porque ele realiza compartilhamento mínimo entre threads. Você provavelmente deve provisionar pelo menos 8-16 núcleos por máquina. Dependendo do custo de CPU da sua carga de trabalho, você também pode precisar de mais: uma vez que os dados estão na memória, a maioria dos aplicativos são vinculados à CPU ou à rede.