DSA – Formação Cientista de Dados

6. Engenharia de Dados com Hadoop e Spark

6.1. Introdução

O Que é o Apache Hadoop?

Apache Hadoop é um projeto livre, da Apache Foundation, composto principalmente por 3 módulos: Apache HDFS, Hadoop Yarn (gerenciador de recursos), Hadoop Map Reduce (processamento de grandes conjuntos de dados).

Um dos grandes desafios computacionais da atualidade é armazenar, manipular e analisar, de forma inteligente, a grande quantidade de dados existente.

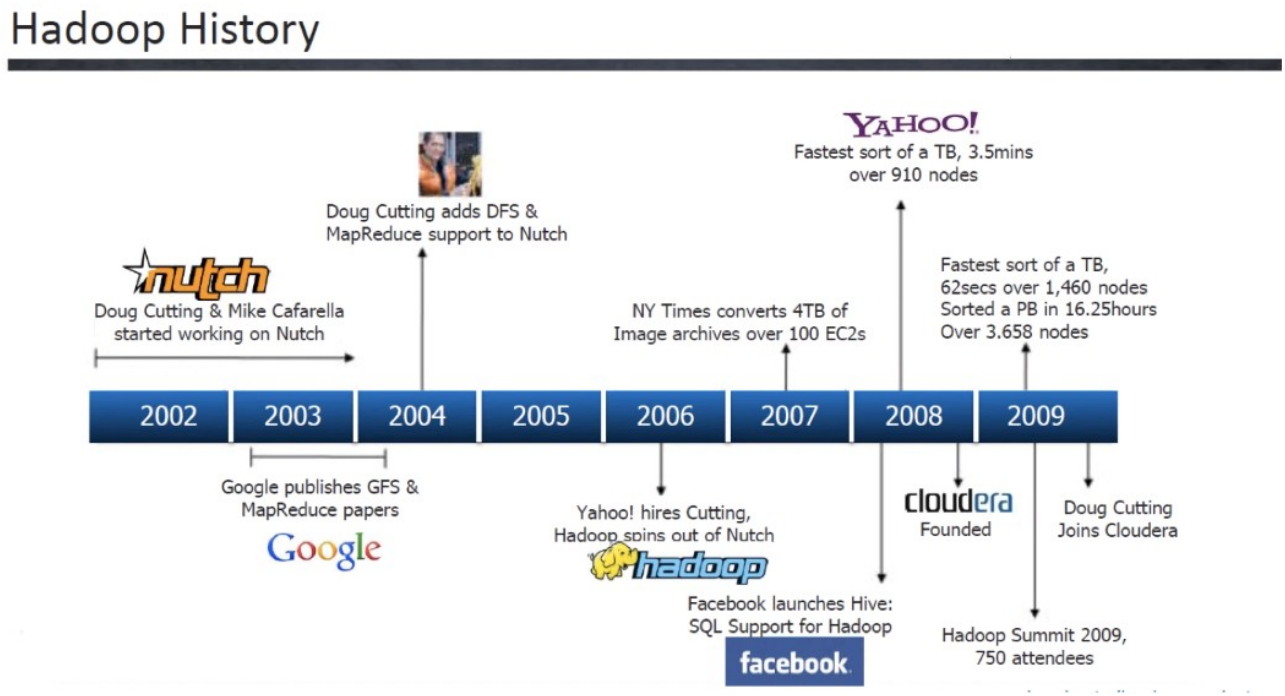
Sistemas corporativos, sistemas Web, mídias sociais, entre outros, produzem juntos um volume impressionante de dados, alcançando a dimensão de petabytes diários.

Os 4 V's do Big Data: Volume, Variedade, Velocidade e Veracidade.

Computação Paralela – impulsionada pela grande demanda de computação e com restrições físicas das arquiteturas convencionais, a computação paralela e distribuída acena como alternativa para otimizar alguns dos grandes desafios computacionais. Esse modelo de computação possui, atualmente, um papel fundamental no processamento e na extração de informação relevante das aplicações de big data. Essa computação é normalmente realizada em clusters, que são conjunto de computadores que conseguem agregar alto poder de processamento a um custo associado relativamente baixo.

O Apache Hadoop é um frammework para processamento e armazenamento de grandes quantidades de dados distribuídos em clusters de computadores.

Uma Breve História do Apache Hadoop



2003 → Google publica artigos sobre o GFS e o MapReduce
2006 → Criação do Hadoop
2013 → Lançado o Hadoop 2.0
2018 → Lançado o Hadoop 3.0
2019 → Lançado o Hadoop 3.2.x

Quais os Benefícios para as Empresas ao Utilizar o Hadoop?

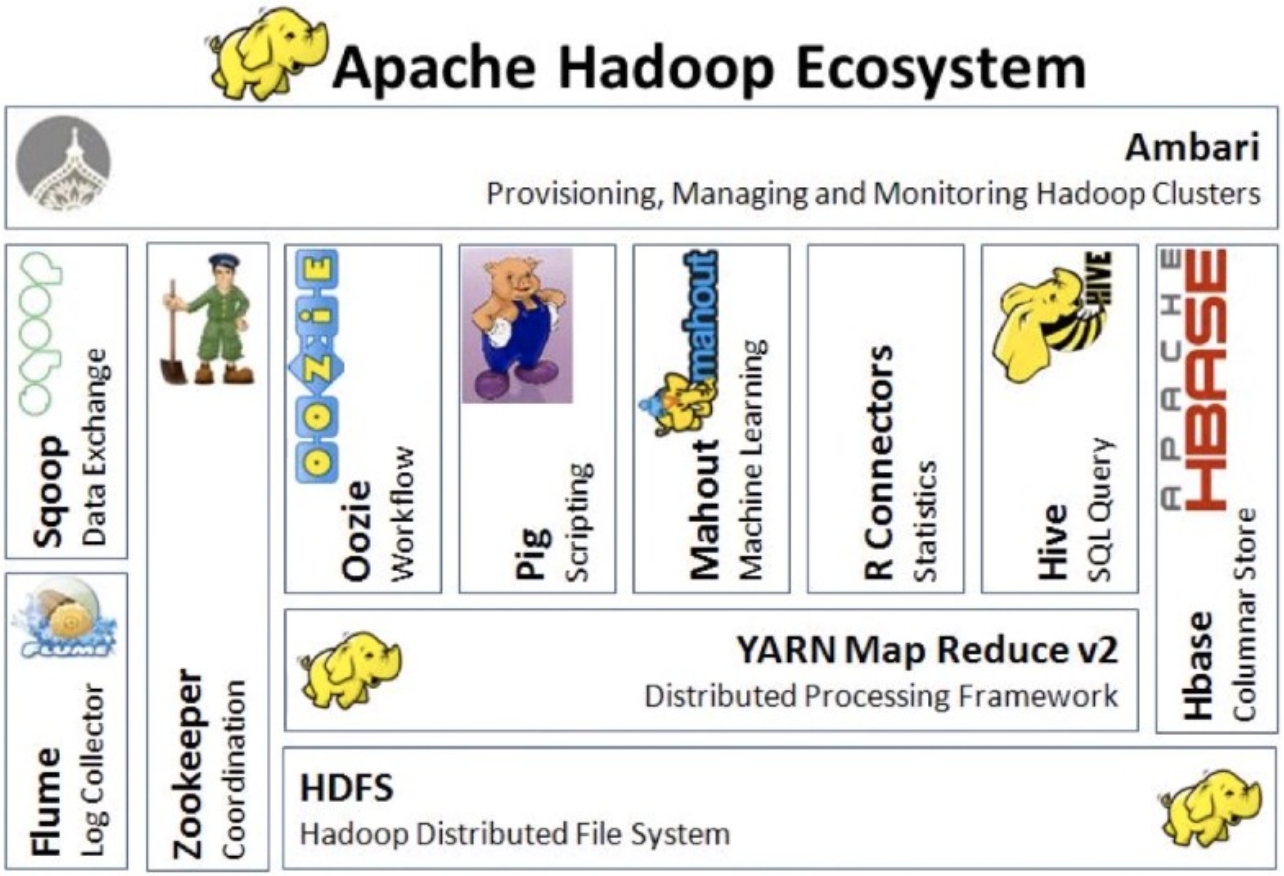
Benefícios do Hadoop:

- Open Source (sofware livre e de grande comunidade ativa)
- Economia (não precisa de aquisição de licenças, poder processar grandes quantidades de dados em máquinas convencionais, e poder utilizar serviços em nuvem, como da Amazon ou da Microsoft)
- Escalabilidade (mudanças de ambiente requerem pequenas mudanças em arquivos de configuração, sem necessidade de alteração das aplicações de análise de dados)
- Robustez (criado para tolerância de falhas, oferecendo estratégias de recuperação automática para essas situações)

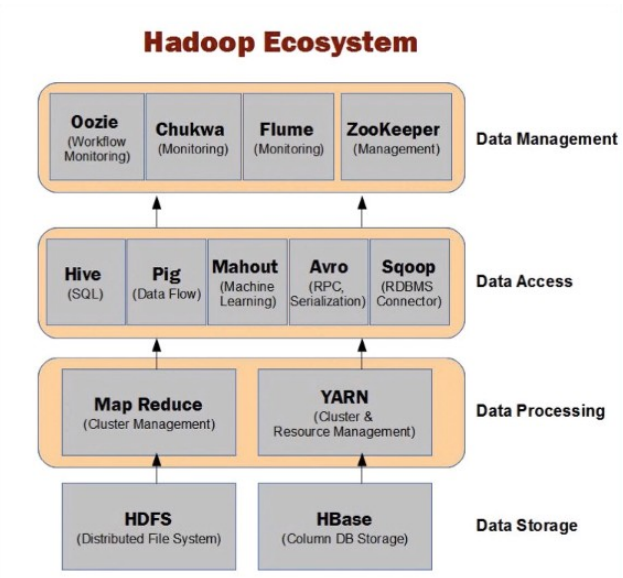
Desvantagens do Hadoop:

- Node Master Único (ponto central de falha)
- Processamento de Arquivos Pequenos
- Muito Processamento em Poucos Dados (melhor menos regras de negócio e grande volume de dados)

Ecossistema Hadoop



- Data Management:
  - Oozie (Workflow Monitoring)
  - Chukwa (Monnitoring)
  - Flume (Monnitoring)
  - ZooKeeper (Management)
- Data Access:
  - Hive (SQL)
  - Pig (Data Flow)
  - Mahout (Machine Learning)
  - Avro (RPC, Serialization)
  - Sqoop (RDBMS Connector)
- Data Processing:



    - Map Reduce (Cluster Management)
    - YARN (Cluster & Resource Management)
- Data Storage
    - HDFS (Distributed File System)
    - HBase (Column DB Storage)

Projetos Principais do Ecossistema Hadoop

Os principais projetos do ecossistema Hadoop são:

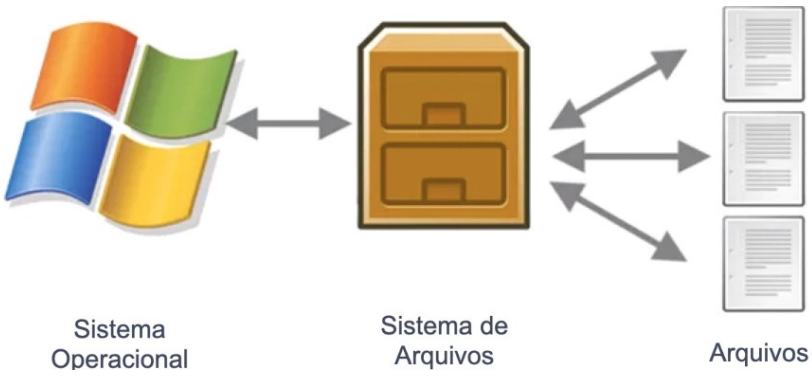
- Hadoop YARN → Gerenciador de recursos para o Hadoop
- Hadoop MapReduce → Módulo de programação ou processamento de dados
- Hadoop HDFS (Hadoop Distributed File System) → Sistema de arquivos distribuídos

Outros projetos importantes do Hadoop são:

- Zookeeper → Criado pela Yahoo, para coordenação de aplicações distribuídas de alto desempenho
- Hive → Criado pelo Facebook, fornece infraestrutura que permite utilizar linguagem SQL (HQL) em dados estruturados
- HBase → Criado pela Powerset (ou Power7), é um banco de dados NoSQL, que permite armazenamento de dados noSchema
- Pig → Linguagem de alto nível orientada a fluxo de dados e de execução para computação paralela. Não altera a configuração do Cluster Hadoop, pois é utilizada no modo cliente, oferecendo uma linguagem PigLatim (que cria Jobs para o MapReduce)
- Sqoop → ferramenta de transferência de dados para o Hadoop, que pode importar ou exportar dados entre uma base de dados e o HDFS, o Hive ou o Hbase.
- Mahout → módulo para se trabalhar com Machine Learning em ambiente distribuído
- Flume → permite trazer para o Hadoop dados das mais variadas fontes, inclusive dados de logs
- Oozie → sistema de fluxo de trabalho, de coordenação de Jobs do Hadoop.

Apache HDFS – Conceito e Importância

HDFS (Hadoop Distributed File System) é um sistema de arquivos criado para armazenamento de big data de forma distribuída, em um cluster de computadores.



Um sistema gerenciador de arquivos possui um conjunto de funcionalidades, tais como armazenamento, organização, nomeação, recuperação, compartilhamento, proteção e permissão de acesso aos arquivos.

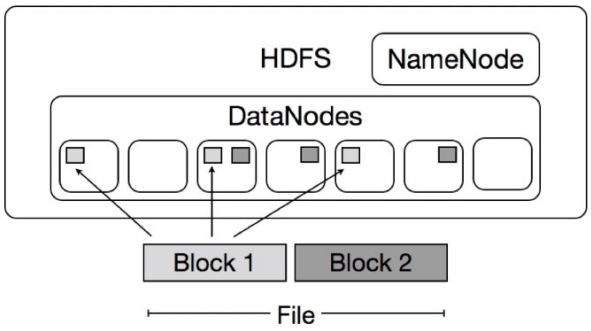
Os tipos de sistemas de arquivos são:

- ext2 – sistema de arquivos padrão Linux
- ext3 – sistema de arquivos ext2 melhorado
- reiserfs – sistema de arquivos do tipo Journaling
- msdos – sistema de arquivos FAT da Microsoft DOS
- vfat – sistema de arquivos FAT-32 do Microsoft Windows
- iso9660 – sistema de arquivos do CD-ROM
- nfs – Network File System. Usado para montar dispositivos em computadores remotos.
- swap – sistema de arquivos de troca utilizado para memória virtual.
- proc – uma janela especial dentro do Kernel do Linux. Utilizada pelos usuários, programas e utilitários para escrever ou ler parâmetros do Kernel. Geralmente montado no diretório “/proc”.

Além das funcionalidades de um sistema gerenciador de arquivos, um sistema de arquivos distribuído deve fornecer outras estruturas de controle imprescindíveis para o seu bom funcionamento. São elas:

- Tolerância a Falhas
- Integridade
- Segurança
- Desempenho
- Consistência

Hadoop Distributed File System



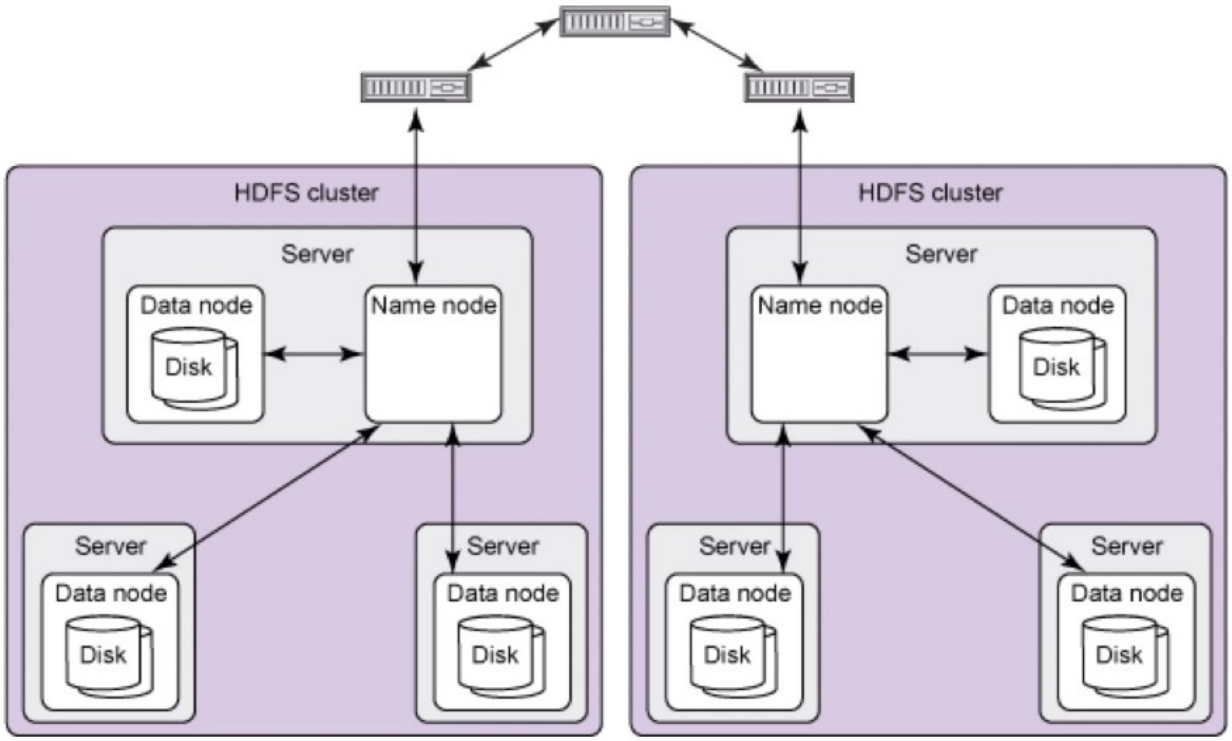
Outros Sistemas de Arquivos Distribuídos:

- NFS – Network File System
- GFS – Google File System
- GlusterFS – GNU Cluster File System

O HDFS foi criado para resolver “Big Problems” e por isso seu funcionamento e arquitetura são próprios para se trabalhar com grandes arquivos de dados e distribuir esses arquivos em blocos ao longo de um cluster de computadores, para que possam ser processados em paralelo.

Apache HDFS – Arquitetura

Arquitetura Master/Worker



NameNode é o componente central do HDFS (também chamado de Master). Assim, é recomendado que ele seja implementado em um nó exclusivo, de melhor desempenho do cluster (é o recomendado).

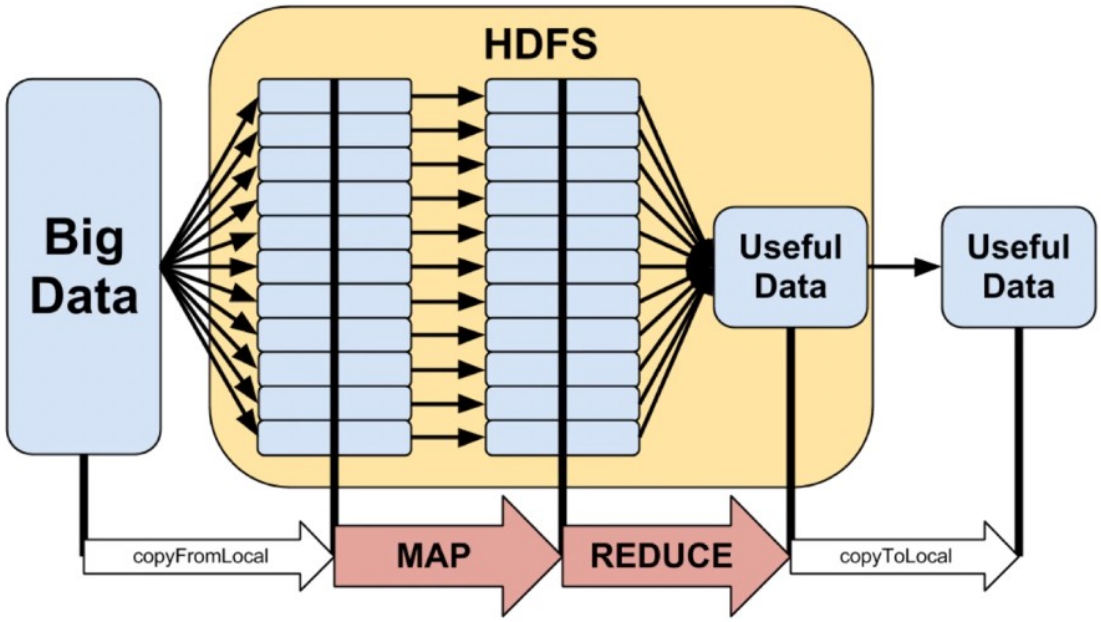
Estrutura de dados do NameNode (automaticamente inicializados):
• FsImage → arquivo responsável por armazenar informações estruturais do bloco, como mapeamento e namespace do diretório de arquivos, além da localização da réplica desses arquivos.
• EditLog → arquivo de log, responsável por armazenar todas as alterações ocorridas nos metadados dos arquivos

Os WorkerNodes são responsá veis pelo armazenamento físico dos dados. É onde o serviço DataNode roda.
Também nos WorkerNodes, tem-se o Task Tracker para o processamento de dados (MapReduce).

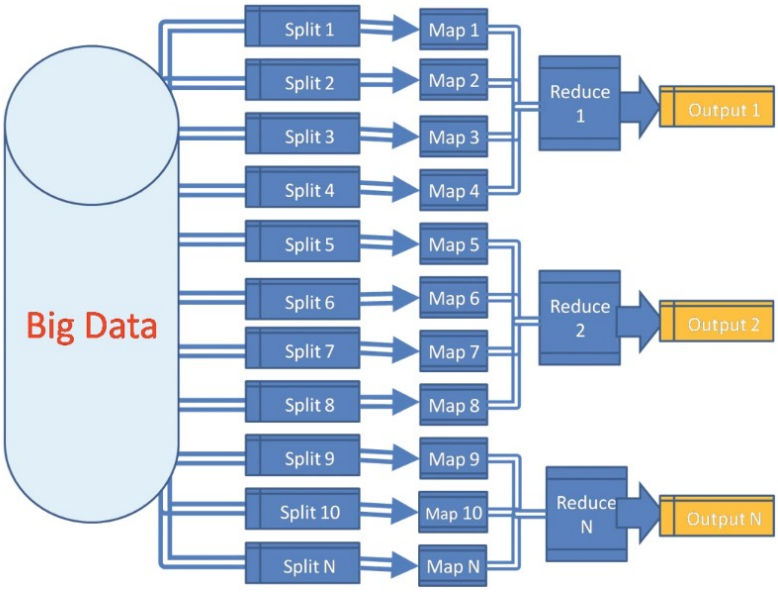
Replicação → Além de dividir os arquivos em blocos, o HDFS ainda replica os blocos, na tentativa de aumentar a segurança.

Definindo MapReduce

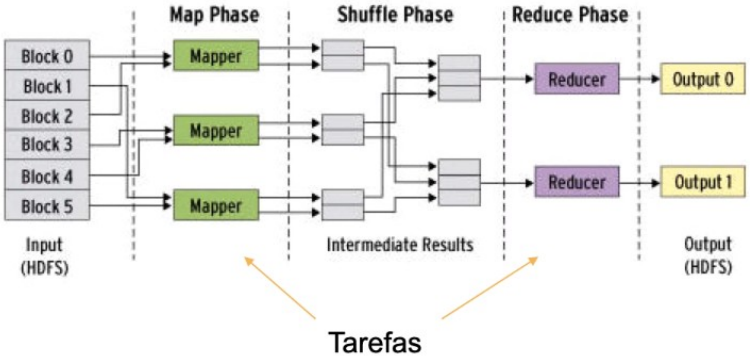
Para processamento de dados massivos, de forma paralela e distribuída, em um cluster de computadores.



Processamento Paralelo e Distribuído:

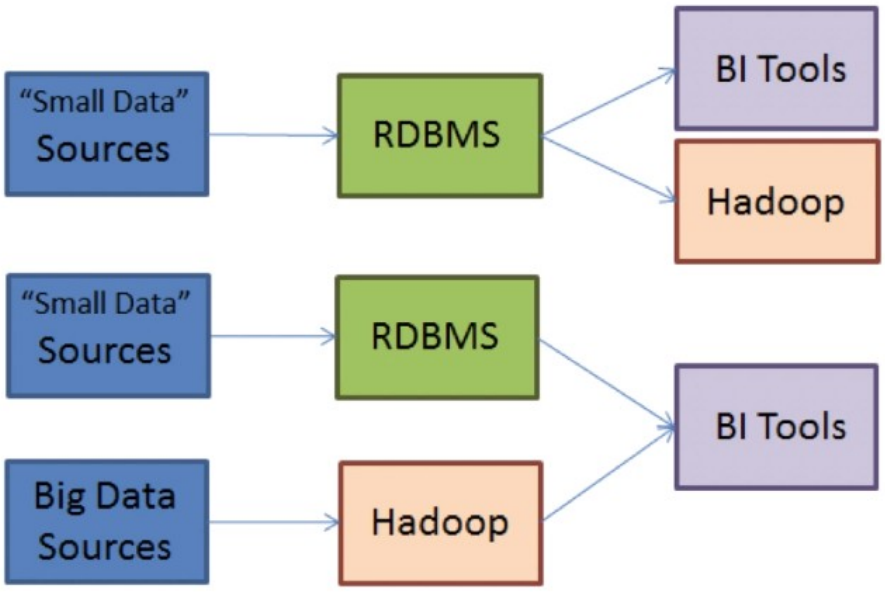


Map → Shuffle → Reduce:



Hadoop x Bancos de Dados Relacionais

- Bancos de Dados Relacionais ou RDBMS (Relational Database Management Systems) → Dados estruturados.
  - SGBD's → Sistemas que gerenciam um ou mais bancos de dados
  - Utilizam linguagem SQL
  - Uma das principais características de um banco relacional: ACID → Atomicidade, Consistência, Isolamento e Durabilidade
- NoSQL → Dados não estruturados podem ser armazenados em vários computadores de processamento e não requerem esquemas fixos. Geralmente, evitam operações de Join e, normalmente, funcionam bem com escalonamento horizontal.
- Hadoop → Grandes volumes de dados (estruturados ou não estruturados)
  - Hadoop não é um banco de dados! É um repositório de dados (um sistema de arquivos)



Hadoop processa dados em batch. Consequentemente, ele não deve ser usado para processar dados transacionais. Mas o Hadoop pode resolver muitos outros tipos de problemas relacionados ao Big Data.

6.2. e 6.3. Instalando o Ecossistema Hadoop

Introdução

Cloudera e Hortonworks → Principais distribuições comerciais do Hadoop

- Oferecem máquinas virtuais prontas para uso e totalmente gratuitas, com todo o ecossistema hadoop instalado.

Existem algumas limitações com essas distribuições:

- Somente para máquinas 64 bits
- Requerem computadores com no mínimo 8 GB de RAM

Hadoop não roda em máquinas Windows. Ele foi projetado para máquinas Unix, como o Linux.

Etapas para criação do ecossistema Hadoop:

- Etapa 1:
  - Criar uma máquina virtual
  - Instalar o sistema operacional Linux
  - Instalar utilitários (Java, ssh, ferramentas)
  - Instalar o MySQL
- Etapa 2:
  - Instalar o Hadoop
  - Configurar o HDFS e o MapReduce
  - Executar um job MapReduce no HDFS
  - Instalar e configurar o Zookeeper
  - Instalar e configurar o Hbase
  - Instalar e configurar o Hive
  - Instalar e configurar o Pig
  - Instalar e configurar o Sqoop
  - Instalar e configurar o Spark
  - Instalar e configurar o Flume

Abaixo você encontra o link para download da máquina virtual construída ao longo dos capítulos 2 e 3 deste curso e que será usada na sequência do treinamento.

https://drive.google.com/open?id=1eimXXmztLTXBmTOaTmpYUlPukzGgtjtU

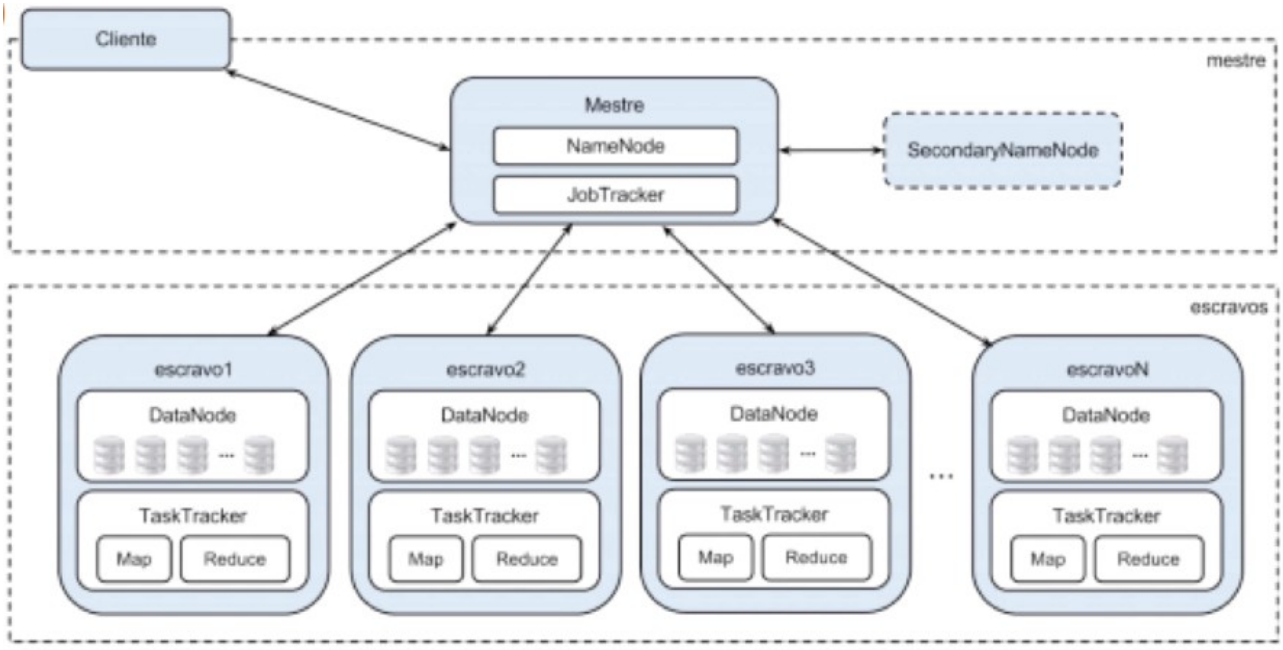
Por Que Cientistas de Dados Precisam Conhecer o Hadoop?

Diferentes pessoas usam diferentes ferramentas para diferentes propósitos.

1. Hadoop é open source
2. Hadoop oferece o framework mais completo para armazenamento e processamento de Big Data
3. A líder mundial em bancos de dados relacionais, a Oracle, oferece soluções de Big Data Analytics com Hadoop
4. A líder mundial em sistemas operacionais, a Microsoft, oferece soluções corporativas em nuvem, com Hadoop
5. O Hadoop é mantido pela Apache Foundation, mas recebe contribuição de empresas como Google, Yahoo e Facebook
6. Um Cientista de Dados deve conhecer bem o paradigma de processamento MapReduce
7. Hadoop normalmente aparece como um dos skills mais procurados em um Cientista de Dados
8. Por se tratar de uma tecnologia avançada, faltam profissionais de Hadoop no mercado
9. Hadoop é usado por algumas das maiores empresas do mundo
10. O Big Data ainda está na sua infância. Onde vamos armazenar todos esses dados?

Aprender ou não Hadoop é uma escolha sua. Mas com certeza este conhecimento será um grande diferencial na sua carreira e na sua compreensão sobre como armazenar e analisar Big Data.

Modos de Execução do Hadoop



Modos de Execução do Hadoop:

- Modo Local (Standalone)
    - Configuração padrão (não precisa alterar os arquivos de configuração)

    - Recomendado para fase de desenvolvimento
- Modo Pseudo-Distribuído (Pseudo-Distributed)
    - Configura-se para execução em cluster, mas tudo é executado numa máquina local (cluster de uma máquina só)
    - Permite simulação de um processamento em paralelo
    - Necessário editar os 3 arquivos de configuração
- Modo Totalmente Distribuído (Fully Distributed)
    - Usado para execução do Hadoop em cluster de computadores
    - Necessário editar os 3 arquivos de configuração

Para alterar entre essas configurações, é necessário a edição de 3 arquivos:

- core-site.xml
- hdfs-site.xml
- mapred-site.xml

Component	Property	Standalone	Pseudo-distributed	Fully distributed
Core	fs.default.name	file:/// (default)	hdfs://localhost/	hdfs://*namenode*/
HDFS	dfs.replication	N/A	1	3 (default)
MapReduce	mapred.job.tracker	local (default)	localhost:8021	*jobtracker*:8021

Instalação e Manuais

Alguns Softwares utilizados:

- Virtual Box
- SO Linux: Red Hat (pago) ou CentOS (gratuito)

Manual Completo de construção de uma Virtual Machine com o ecossistema Hadoop:

D:\Desenvolvedor\CienciaDeDados\Estudos\DSA-Cursos\FCD\06_EngenhariaDadosHadoop&Spark\Cap02&Cap03\2-Manual.pdf

Instalando e Configurando Máquina Virtual Cloudera

Acesse: www.cloudera.com
Downloads → Quickstart Vms
Selecione a plataforma → Virtual Box
Clique em Get It Now → Preencha o formulário e faça o download

Atenção: Só é possível utilizar essa VM com uma máquina de 64-bit

Apache Hadoop com Containers Docker

Acesse: www.docker.com
Clique em Sign In e crie uma conta
Acesse a documentação
Em Get Docker → Clique em Docker CE (Community Edition)

Após instalação, acesse o Docker Hub e baixe um container pronto (se desejar)

Quizz

O Hbase é um componente do ecosistema Hadoop, responsável pelo armazenamento de dados não estruturados.

Em um ambiente Hadoop, nós temos um computador executando os processos mestres ou Masters, que são o Namenode, o secondary namenode e o Job tracker. Esses são processos de gerenciamento. O namenode gerencia o HDFS e o Jobtracker gerencia os trabalhos de MapReduce. O secondary namenode tem uma função similar a um backup, embora ele possa assumir outras funções de gerenciamento.

No modo pseudo-distribuído são aplicadas todas as configurações, semelhantes às necessárias para execução em um cluster, entretanto, toda a aplicação é processada em modo local, por isso o termo pseudo-distribuído ou também chamado “cluster” de uma máquina só.

Depois de instalar o Hadoop, nosso trabalho é editar os arquivos de configuração e definir em qual modo de execução usaremos o Hadoop.

O Flume é um componente do ecosistema Hadoop, responsável pela coleta de streaming de dados.

6.4. Planejando e Configurando um Cluster Hadoop

Introdução

O que será estudado no capítulo?

- Arquitetura de um Cluster Hadoop
- Topologia de Rede para o Cluster Hadoop
- Workflow
- Planejamento do Cluster
- Hardware Configuração de Rede do Cluster Hadoop
- Arquivos de Configuração
- Parâmetros de Configuração
- Como funciona o HDFS
- HDFS Writes
- HDFS Reads
- Importando Dados do MySQL para o HDFS

O Que é um Cluster?

Um cluster é um conjunto de computadores conectados que trabalham juntos para que, em muitos aspectos, possam ser vistos como um único sistema. Os clusters de computadores têm cada nó configurado para executar a mesma tarefa, controlada e programada por software.

Um node é o nome dado a cada um dos computadores de um cluster.

Principais Tipos de Cluster de Computadores

Principais Tipos de Cluster:

- Cluster de Alto Desempenho → Direcionado para aplicações muito exigentes, no que diz respeito a processamento (cluster Hadoop). Ex: sistemas de pesquisa científica (cálculos complexos)

    1 gigaflop corresponde a 1 bilhão de instruções por segundo

- Cluster de Alta Disponibilidade → O objetivo é que a aplicação não pare de funcionar. Ex: aplicações de missão crítica.
- Cluster para Balanceamento de Carga → Muito utilizado em servidores web e bancos de dados.
- Cluster Combo → cluster que possua todas as características citadas acima

Para uma aplicação de Big Data podemos configurar um cluster de alto desempenho e ao mesmo tempo alta disponibilidade, se for necessário processamento e análise de dados em tempo real, para um sistema de recomendação, por exemplo.

Aplicações de Cluster sde Computadores e Cloud Computing

Em geral, as principais aplicações de cluster de computadores são: aplicações de missão crítica, provedores de internet, internet banking, pesquisas científicas, etc.

As tecnologias de Clustering possibilitam a solução de diversos problemas que envolvem grande volume de processamento.

Cloud Computer → É utilizar o computador que alguém está disponibilizando, em qualquer lugar do planeta. A pioneira nesse seguimento foi a Amazon.

Arquitetura do Cluster Hadoop

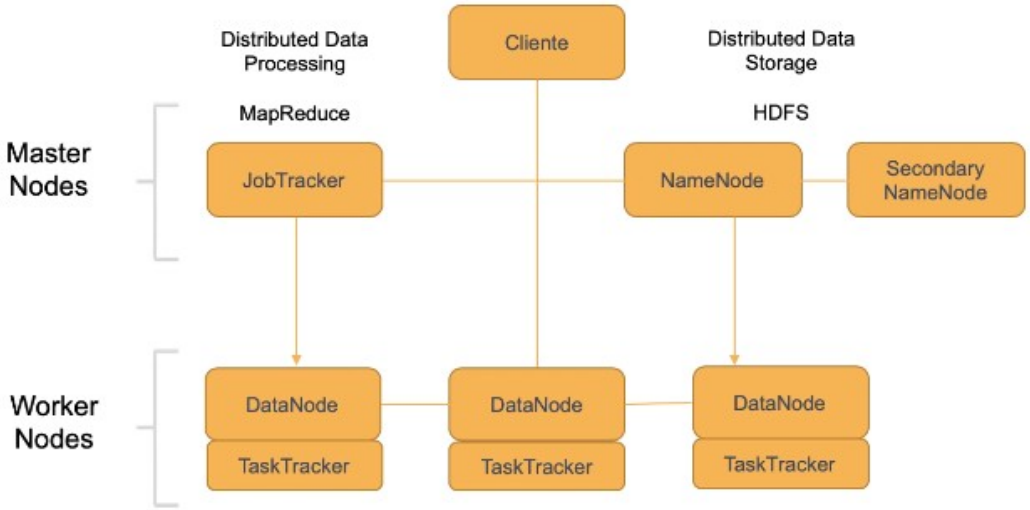
O que é um cluster hadoop?

Um Cluster Hadoop é um conjunto de máquinas com Hadoop instalado que é criado para armazenar e analisar grandes quantidades de dados, sejam eles estruturados ou não estruturados. Em um Cluster Hadoop, os dados são armazenados e processados ao longo de diversos computadores e tudo isso é feito de forma paralela.

Para armazenamento, utiliza-se o HDFS. Para processamento, o MapReduce.

Diagrama que resume um cluster hadoop:

- Cliente: tudo começa com uma requisição cliente, que vai solicitar o armazenamento, leitura ou processamento de dados. O cliente é uma operação, também conhecida como Job.
- Distributed Data Processing: Função do MapReduce, trata-se do processamento distribuído.
    - Master Nodes: JobTracker (MapReduce)
    - Worker Nodes: DataNode e TaskTracker
- Distributed Data Storage: Função do HDFS, trata-se do armazenamento distribuído.
    - Master Nodes: NameNode (serviço master ou principal) e Secondary NameNode
    - Worker Nodes: DataNode e TaskTracker



Funcionamento do Cluster Hadoop

- Passo 1: Requisição do Cliente
  - Cliente (Dados + Processamento) → Namenode (HDFS) / JobTracker (MapReduce)
- Passo 2: Dados são divididos em blocos e Jobs são divididos em partes menores (tarefas)
- Passo 3: Blocos e Tarefas são distribuídos pelo cluster
- Passo 4: Armazenamento/Processamento paralelo
  - JobTracker aciona os TaskTrackers
- Passo 5: Unir os resultados
  - Map (mapear os dados) e Reduce (redução)
- Passo 6: Resultado Final
  - Após a conclusão de cada tarefa de cada tasktracker, o jobtracker vai pegar cada resultado e juntar tudo isso numa entrega final ao cliente

Dados são gravados no DataNode (Armazena/Recupera Dados). Já os metadados (dados sobre os dados) são gravados no NameNode.

O TaskTracker executa Jobs de MapReduce.

Topologia de Rede para o Cluster Hadoop

Rack: estrutura física onde se armazena os computadores.

- Um cluster é um conjunto de computadores. Um rack é um conjunto de computadores dentro de um cluster.
- O Hadoop possui Rack Aware (consciência do rack) – conhece a estrutura de rede onde está instalado.

Switch: dispositivo para interconexão das máquinas.

A principal recomendação do Apache Hadoop é não colocar serviços master no mesmo rack. Isso porque, se um rack parar de funcionar, o cluster continua funcionando (desde que esse rack não possua o name node).

Mas e se o NameNode também der problema? Por isso devemos configurar um Secondary NameNode.

Secondary NameNode não é configuração de alta disponibilidade. Alta disponibilidade é quando se tem 2 máquinas como NameNode.

Workflow do Cluster Hadoop

Basicamente, todo o processo se resume em 4 etapas:

1. Os dados são divididos em blocos e distribuídos pelo cluster Hadoop
2. MapReduce analisa os dados baseado nos pares de chave-valor
3. Os resultados são colocados em blocos através do cluster Hadoop

4. Os resultados podem ser lidos do cluster

O objetivo do Cluster Hadoop, é o rápido processamento, em paralelo, de grandes quantidades de dados.

A configuração padrão do Hadoop, é ter 3 cópias de cada bloco de dados no cluster (o que pode ser modificado pelo parâmetro dfs.replication no arquivo de configuração hdfs-site.xml).

Workflow de Gravação de Dados no HDFS:

1. O cliente interage com o NameNode para obter a localidade onde o storage está disponível
2. O cliente então interage diretamente com o DataNode
3. O cliente envia os dados, que são divididos em pequenos pedaços de blocos
4. Após o dado ser completamente gravado pelo primeiro node, a replicação é feita para os demais nodes

Após todos os DataNodes terminarem a gravação do dado, o relatório de blocos envia um sinal ao cliente, que então comunica o NameNode. Os DataNodes também enviam o relatório de blocos ao NameNode. O NameNode utiliza o relatório de blocos para atualizar os Metadados.

A Função do NameNode no Processo de Gravação no HDFS

O NameNode é o controlador principal do HDFS, que mantém os metadados de todo o sistema de arquivos para o cluster.

Principais características do NameNode:

❑ Mantém o track de como cada bloco compõe um arquivo e a localização de cada bloco no cluster

❑ O NameNode não contém qualquer bloco de dados

❑ Direciona o cliente para os DataNodes e mantém o histórico de condições de cada DataNode

❑ Garante que cada bloco de dado atende aos critérios mínimos definidos pela política de replicação

O NameNode funciona da seguinte forma:

✓ Os DataNodes enviam sinais (heartbeats) para o NameNode a cada 3 segundos através de TCP Handshake.

✓ Cada décimo sinal é um relatório de bloco.

✓ O relatório de bloco permite que o NameNode crie os metadados e garanta que 3 cópias de cada bloco existam em nodes diferentes

✓ Se o DataNode fica sem conexão, o sinal não é enviado e o NameNode deixa de considerar aquele DataNode

✓ O NameNode então replica o bloco para outro DataNode, sempre mantendo 3 cópias de cada bloco.

Workflow de Leitura de Dados no HDFS:

1. Para recuperar um documento do HDFS, o cliente aciona o NameNode e solicita o endereço (bloco) onde o dado está armazenado.
2. O cliente então solicita ao DataNode o dado, com o endereço do bloco fornecido pelo NameNode. Tudo isso ocorre via protocolo TCP na porta 50010.

Planejamento do Cluster Hadoop

Fatores para Planejamento do Cluster Hadoop:

- Objetivo – Volume de dados x Alta disponibilidade
- Serviços – MapReduce (JobTracker, TaskTracker), HDFS (NameNode, DataNode), Storage (NFS, SAN)
- Layout – Pseudo-Distribuído para desenvolvimento e Totalmente Distribuído para produção Local / Nuvem

Hardware e Configuração de Rede do Cluster Hadoop

Worker:

Configuração	**Descrição**
Storage	Em um ambiente de intensivo i/o, recomenda-se 12 discos SATA 7200 RPM de 2 TB cada um, para balanceamento entre custo e performance. RAID não é recomendado em máquinas com serviços workers do Hadoop.
Memória	Nodes slaves requerem normalmente entre 24 e 48 GB de memória RAM. Memória não utilizada será consumida por outras aplicações Hadoop.
Processador	Processadores com clock médio e menos de 2 sockets são recomendados.
Rede	Cluster de tamanho considerável, tipicamente requer links de 1 GB para todos os nodes em um rack com 20 nodes.

Master:

Configuração	**Descrição**
Storage	Deve-se utilizar 2 servidores: um para o NameNode Principal e outro para o Secundário. O Master deve ter pelo menos 4 volumes de storage redundantes, seja local ou em rede.
Memória	64 GB de RAM suportam aproximadamente 100 milhões de arquivos.
Processador	16 ou 24 CPU’s para suportar o tráfego de mensagens.

Estas são apenas recomendações e que podem variar de acordo com os fatores para o planejamento do cluster: objetivo, serviços e layout.

A Instalação do Hadoop possui o mesmo processo na máquina Master ou Worker:

1. Instalar um servidor ssh
2. Criar um login ssh sem senha
3. Instalar o Java JDK

4. Instalar o Hadoop
5. Definir o diretório do Hadoop
6. Editar os arquivos de configuração do Hadoop
7. Criar diretório temporário
8. Iniciar o Hadoop
9. Acessar o Hadoop pelo browser e testar os serviços
10. Automatizar a inicialização do Hadoop

Single Node x Multi Node:

Cluster Single Node	Cluster Multi Node
Hadoop é instalado em um único servidor (node)	Hadoop é instalado em diversos nodes (entre algumas dezenas, até milhares)
Clusters Single Node são usados para processos triviais e operações simples de MapReduce e HDFS. Pode ser usado em ambiente de testes.	Clusters Multi Node são usados para computação complexa, incluindo processamento analítico.

Quando Usar e Quando Não Usar o HDFS?

Hadoop Distributed File System (HDFS) é um framework distribuído e extremamente tolerante a falha.

Foi concebido para processar grandes volumes de dados.

O conceito do HDFS é baseado no Unix.

O HDFS é similar a outros frameworks de arquivos distribuídos, mas com algumas diferenças:

- O HDFS possui um modelo chamado ”write-once-read-manytimes” (WORM), que significa: escreva uma vez e leia quantas vezes quiser.
- Eficiente controle de concorrência.
- Redireciona atividades (jobs) em caso de falhas.

Quando usar o HDFS?

- Grande quantidade de dados a serem armazenados
- Streams de dados constantes que requerem acesso
- Apenas equipamentos simples estão disponíveis

Quando NÃO usar o HDFS?

- Quantidade considerável de arquivos pequenos
- Composições variadas (muitos arquivos em formatos diferentes)
- Acesso de baixa latência aos dados

Customizando o HDFS

Cria os diretórios abaixo:

mkdir /opt/hadoop/dfs
mkdir /opt/hadoop/dfs/data
mkdir /opt/hadoop/dfs/namespace_logs

Editar o arquivo $HADOOP_HOME/etc/hadoop/hdfs-site.xml e adicionar as linhas:

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/opt/hadoop/dfs/namespace_logs</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/opt/hadoop/dfs/data</value>
</property>
```

Importando Dados do MySQL para o HDFS

Procedimento para configurar/alterar a senha do root:

- # Para o serviço
  sudo systemctl stop mysqld
- # Inicializa em modo de segurança
  sudo mysqld --skip-grant-tables --user=mysql &
- # Abre o shell
  mysql
- # Reset dos privilégios
  FLUSH PRIVILEGES;
- # Altera a senha do root
  ALTER USER 'root'@'localhost' IDENTIFIED BY 'dsahadoop';
- # Verifica as regras de senha
  SHOW VARIABLES LIKE 'validate_password%';
- # Ajusta a senha
  ALTER USER 'root'@'localhost' IDENTIFIED BY 'Dsahadoop@1';
- # Reinicia a máquina
- # Acessa o MySQL
  mysql -u root -p

Comandos Sqoop para listar bancos de dados no MySQL:

- sqoop list-databases --connect jdbc:mysql://localhost:3306/ --username root -P

- sqoop list-databases --connect jdbc:mysql://localhost:3306/?serverTimezone=UTC --username root -P

Comando Sqoop para importar dados do MySQL para o HDFS:

- sqoop import --connect jdbc:mysql://localhost:3306/testedb?serverTimezone=UTC --username root --password
- Dsahadoop@1 --table empregados --m 1

Quiz

Cluster (ou clustering) é, em poucas palavras, o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. Estas máquinas dividem entre si as atividades de processamento e executam este trabalho de maneira simultânea.

DataNode é o node responsável por armazenar/recuperar os dados.

Funções do NameNode: Direcionar o cliente para os DataNodes e manter o histórico de condições de cada DataNode; Manter o track de como cada bloco compõe um arquivo e a localização de cada bloco no cluster; Garantir que cada bloco de dado atende aos critérios mínimos definidos pela política de replicação.

Função do JobTracker: Dividir os jobs em tarefas e entregar aos Task Trackers.

O relatório de bloco permite que o NameNode crie os metadados e garanta que 3 cópias de cada bloco existam em nodes diferentes.

???

???

6.5. Usando MapReduce em Grandes Volumes de Dados

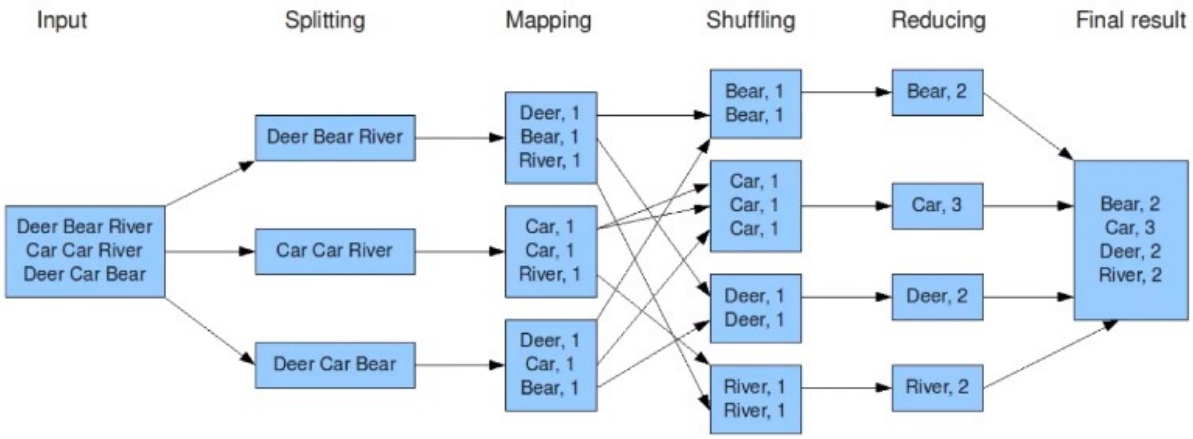
Introdução

O que vamos estudar neste capítulo?

• Computação Distribuída
• Funcionamento do MapReduce
• Processamento de Dados Armazenados no HDFS
• Processamento de Big Data
• Criação e Monitoramento de Jobs MapReduce
• Processamento de Jobs MapReduce em Nuvem, com o Serviço AWS da Amazon

O Que é MapReduce?

O que exatamente é processamento MapReduce?
Programação de computadores.



Computação Distribuída e Computação Paralela

Sistema de Processamento Distribuído e Paralelo é um sistema que interliga vários nós de processamento, ou seja, vários computadores individuais que não são, necessariamente, homogêneos, de maneira que um processo que necessite de um grande poder computacional possa ser subdividido em vários nós, conseguindo assim ganho de performance.

Uma tarefa qualquer pode ser dividida em várias subtarefas, que então podem ser executadas em paralelo.

Sistemas distribuiídos são usados em várias situações, onde grande processamento se faz necessário, como:

• Pesquisas científicas
• Previsões climáticas
• Descoberta de novas partículas
• Controle de epidemias
• Armazenamento e Processamento de Big Data

Sistemas Computacionais estão cada vez mais elaborados e complexos; e grande parte das máquinas estão interligadas por redes de computadores. Assim, favorecendo a computação distribuída.

Sistemas Distribuídos permitem:

- Maior poder de processamento
- Maior carga, maior número de usuários
- Melhor tempo de resposta
- Maior confiabilidade

A computação distribuída consiste na utilização de um conjunto de máquinas conectadas por uma rede de comunicação, atuando como um único sistema.

Principais Características da Computação Distribuída:

- Executa aplicações através de máquinas diferentes, como se estas fossem uma só.
- Tornou-se possível com a popularização das redes de computadores.
- As máquinas podem estar interligadas por redes intranets, internet, redes públicas e privadas.

Principais Vantagens da Computação Distribuída:

- Utiliza melhor o poder de processamento
- Apresenta melhor desempenho
- Permite compartilhar dados e recursos
- Pode apresentar maior confiabilidade
- Permite reutilizar serviços já disponíveis
- Permite processar grandes conjuntos de dados

Computação Distribuída é quando se tem vários computadores. Computação Paralela é a paralelização de tarefas que podem ser feitas, inclusive, dentro de um mesmo computador. Por exemplo, Programação Paralela em GPU.

Computação Distribuída – Cloud Computing

O Cloud Computing é uma boa alternativa para questões como:

- Será que uma empresa tem condições de montar um cluster de computadores?
- Criado um cluster, será que precisamos utilizá-lo o tempo todo? O que fazemos quando o cluster ficar ocioso?

Ao invés da empresa montar infraestrutura internamente, ela leva isso para um provedor de ambiente em nuvem.

Uma grande vantagem de se utilizar cloud computing é o conceito de Pay As You Go, ou pague conforme uso.

Com Cloud Computing tem-se uma série de possibilidades:

- Infraestrutura como um serviço (IaaS)
- Plataforma como um serviço (PaaS)
- Software como um serviço (SaaS)
- Big Data como um serviço (BDaaS)

O Modelo de Programação MapReduce

Como exatamente funciona o modelo MapReduce?

Dados de Entrada → Mapper/Mapeamento (onde o Cientista de Dados define o que será a chave e o que será o valor) → Shuffle (feito pelo frammework, como o Hadoop MapReduce) → Reduce/Redução (também definido pelo Cientista de Dados)

Mapper converte dados brutos em pares de chave/valor.

Feito o mapeamento, o Shuffle agrupa todos os pares de chave/valor e entrega para a etapa de redução. A redução, por exemplo, pode retornar as chaves e o total de suas ocorrências, reduzindo assim os dados à informação que você precisa.

Como o MapReduce Utiliza a Computação Distribuída

Etapas do processamento do MapReduce na computação distribuída (Workflow do MapReduce):

1. Agendamento
    - Os jobs são divididos em pedaços menores chamados tarefas
      As tarefas são agendadas pelo YARN
2. Localização de Tarefas
    - As tarefas são colocadas nos nodes que armazenam os segmentos de dados
      O código é movido para onde o dado está
3. Tratamento de Erros
    - Falhas são um comportamento esperado e no caso de falhas, as tarefas são automaticamente enviadas a outros nodes
4. Sincronização de Dados
    - Os dados são randomicamente agrupados e movidos entre os nodes
      Input e output são coordeandos pelo framework

Características do MapReduce

Algumas das principais características do MapReduce:

➢ Consegue trabalhar com grandes volumes de dados
➢ Funciona bem com o conceito WORM (Write Once and Read Many)
➢ Permite paralelismo
➢ As operações são realizadas próximas dos dados

- Hardware e storage de baixo custo podem ser usados
- O runtime fica responsável por dividir e mover os dados para as operações

Processo de Recuperação a Falhas do MapReduce

Processo interno de recuperação no caso de uma falha ocorrer:

1. Os processos de Task enviam sinais ao TaskTracker
2. Os TaskTrackers enviam sinais ao JobTracker
3. Tasks que levam mais de 10 min para responder ou emitem uma exceção, são finalizados pelo TaskTracker
4. O TaskTracker reporta as Tasks com erro para o JobTracker
5. O JobTracker reagenda as Tasks que falharam em TaskTrackers diferentes
6. Se uma Task falhar mais de 4 vezes, o job inteiro falha

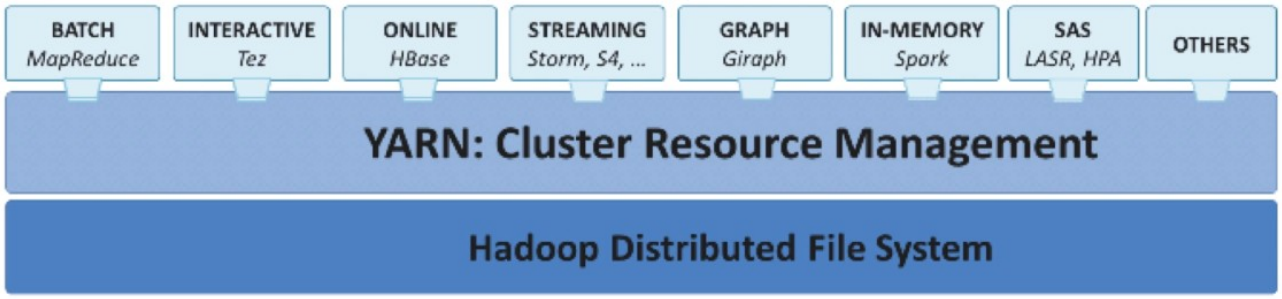
YARN

Apache YARN – "Yet Another Resource Negotiator" é a camada de gerenciamento de recursos do Hadoop.

O YARN foi introduzido no Hadoop 2.x e permite diferentes mecanismos de processamento de dados, como processamento de grafos, processamento interativo, processamento de fluxo e processamento em lote para executar e processar dados armazenados no HDFS.

Além do gerenciamento de recursos, o Yarn também é usado para agendamento de tarefas (jobs). O YARN amplia o poder do Hadoop para outras tecnologias, para que possam aproveitar as vantagens do HDFS (sistema de armazenamento mais confiável e popular do planeta) e do cluster de baixo custo.

O Apache YARN também é considerado como o sistema operacional de dados do Hadoop. A arquitetura do YARN fornece uma plataforma de processamento de dados de uso geral que não se limita apenas ao MapReduce.



Laboratório 2 – Trabalhando com MapReduce

Para execução do programa, no terminal digite:
python AvaliaFilme.py hdfs:///mapred/u.data -r hadoop

```
# AvalaFilme.py:
```

```
# Fase 1 –Mapeamento
# A palavra reservada yielddefine qual das colunas será a chave (nesse caso a coluna rating, pois queremos saber o total de filmes em cada rating,
# que vai de 1 a 5). Cada rating é mapeado e identificado com o valor 1, registrando a ocorrência do rating. Esse código é definido pelo
# Cientista de Dados.

# Fase 2 –Shuffle e Sort
# Essa fase é processada automaticamente pelo framework MapReduce, que então agrupa os ratings e identifica quantas ocorrências cada rating
# obteve ao longo do arquivo.

# Fase 3 –Redução
# Também definida pelo Cientista de Dados, esta fase aplica o cálculo matemático (no caso soma, com a função sum())
# e retorna o resultado: total de filmes com rating 1, total de filmes com rating 2, etc...


# o arquivo mrjob foi instalado durante a configuração do ambiente
# o MRJob permite que se programe em python, para então utilizar o MapReduce
from mrjob.job import MRJob

# a classe MRAvaliaFilme faz uma instância ao MRJob
class MRAvaliaFilme(MRJob):
    # metodo mapper
    def mapper(self, key, line):
        (userID, movieID, rating, timestamp) = line.split('\t') #\t trata-se do caracter tab
        # yield e uma palavra reservada que permite realizar uma iteracao em uma sequencia de valores,
        # sem ter que carregar tudo na memoria do computador
        yield rating, 1 # retorna a variavel rating e o numero 1 (par chave/valor)

    # metodo reducer
    def reducer(self, rating, occurences):
        yield rating, sum(occurences)

# cria o programa main e executa a classe MRAvaliaFilme
if __name__ == '__main__':
    MRAvaliaFilme.run()
```

Programação e Execução do Job MapReduce para Gerar Média de Amigos em Rede Social por Idade

Para execução do programa, no terminal digite:

jps → verifica se os serviços estão em execução
hdfs dfs -rm mapred/u.data → para remoção do arquivo do job anterior
hdfs dfs -put Datasets/amigos_facebook.csv /mapred → copiando arquivo para o hdfs
python Analytics/AmigosIdade.py hdfs:///mapred/amigos_facebook.csv -r hadoop → executando programa no terminal

```
from mrjob.job import MRJob

class MRAmigosPorIdade(MRJob):

    def mapper(self, _, line): # o _ serve para nao ter que colocar cada coluna como entrada (permite receber n atributos
        (ID, nome, idade, numAmigos) = line.split(',')
        yield idade, float(numAmigos) # converte para float para poder calcular a media no reducer

    def reducer(self, idade, numAmigos):
        total = 0
        numElementos = 0
        for x in numAmigos:
            total += x
            numElementos += 1

        yield idade, total / numElementos


if __name__ == '__main__':
    MRAmigosPorIdade.run()
```

Data Mining com MapReduce em Dados Não Estruturados

Para execução do programa, no terminal digite:

hdfs dfs -put Datasets/OrgulhoePreconceito.txt /mapred

python Analytics/MR-DataMining-???.py hdfs:///mapred/ OrgulhoePreconceito.txt -r hadoop

```
##  MR-DataMining-1.py ##

# Data Mining de palavras:

from mrjob.job import MRJob

class MRDataMining(MRJob):

    def mapper(self, _, line):
        palavras = line.split() # o split utiliza o espaco como padrao para separacao
        for palavra in palavras:
            yield palavra.lower(), 1 #  converte para minusculo e traz valor 1 para contabilidade no reduce

    def reducer(self, key, values):
        yield key, sum(values)


if __name__ == '__main__':
    MRDataMining.run()
```

```
## MR-DataMining-2.py ##

# Limpeza com Expressoes Regulares – Separando palavras das pontuacoes

from mrjob.job import MRJob
import re # expressoes regulares

REGEXP_PALAVRA = re.compile(r"[\w']+")  # retorne tudo que for w (palavra)

class MRDataMining(MRJob):

    def mapper(self, _, line):
        palavras = REGEXP_PALAVRA.findall(line)
        for palavra in palavras:
            yield palavra.lower(), 1

    def reducer(self, key, values):
        yield key, sum(values)


if __name__ == '__main__':
    MRDataMining.run()
```

```
## MR-DataMining-3.py ##

# Jobs Aninhados – Separa palavras das pontuacoes e ordena por quantidade de aparicoes no texto

from mrjob.job import MRJob
from mrjob.step import MRStep # Cria dentro de um único job vários passos de mapeamento e reducao
import re

REGEXP_PALAVRA = re.compile(r"[\w']+")

class MRDataMining(MRJob):

    def steps(self):
        return [
            MRStep(mapper = self.mapper_get_words, reducer = self.reducer_count_words),
            MRStep(mapper = self.mapper_make_counts_key, reducer = self.reducer_output_words)
        ]

    def mapper_get_words(self, _, line):
        palavras = REGEXP_PALAVRA.findall(line)
        for palavra in palavras:
            yield palavra.lower(), 1

    def reducer_count_words(self, palavra, values):
        yield palavra, sum(values)
```

```
    def mapper_make_counts_key(self, palavra, count): # conta quantas vezes cada chave aparece
        yield '%04d'%int(count), palavra  # '%04d'%int(count) para entregar na mascara de 4 numeros inteiros

    def reducer_output_words(self, count, palavras):
        for palavra in palavras:
            yield count, palavra


if __name__ == '__main__':
    MRDataMining.run()
```

Laboratório 3 – Analisando Logs de Servidores Web no Cloudera

Iniciar a VM do Cloudera
Verifique se a Shared Folder está com a pasta correta dos dados utilizados no trabalho

No terminal da VM:

hdfs dfs -mkdir /mapred → criando o diretorio
hdfs dfs -ls → lista arquivos e diretorios

cd /media/ → entrar no diretorio media
ls → lista a pasta selecionada no Shared Folder
cd ~ → retorna para diretorio home

mkdir Datasets → cria pasta para os arquivos
mkdir Analytics → cria pasta para o resultado dos jobs

su → entrar como usuário root (que possui as devidas permissões)
em Password: digite cloudera (senha padrão)

cd /media/sf_Cap05/ → entre na pasta de trabalho
cd Datasets/
ls
cp web_server.log.zip /home/cloudera/Datasets/ → copia arquivo de uma pasta para outra
cd .. → volta para diretorio anterior
cd Analytics/
cp mapper.py reducer.py /home/cloudera/Analytics → copia 2 arquivos para outra pasta

exit → sair do usuário root
clear → limpar tela do terminal

sudo chown -R cloudera:cloudera Datasets/ → muda o proprietario de root para o usuario atual
sudo chown -R cloudera:cloudera Analytics/

cd Datasets/
ls -la → perceba que agora o proprietario é o cloudera
pwd → mostra o diretorio onde você se encontra no terminal
unzip web_server.log.zip → descompacta o arquivo de logs

hdfs dfs -put web_server.log /mapred→ copia o arquivo para o hdfs (cluster hadoop)
hdfs dfs -ls /mapred → para conferir o conteudo do diretorio no cluster

cd..
cd Analytics/ → entra no diretorio para editar os arquivos .py
gedit mapper.py → abre o arquivo no gedit

cd..
cd Analytics/
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-(tecle tab e escolha a versão que aparecer do arquivo .jar) **-mapper** mapper.py **-reducer** reducer.py **-file** mapper.py **-file** reducer.py **-input** /mapred/web_server.log **-output** /saida → executa um arquivo jar que possui uma série de classes java para execução do job mapreduce

clear
hdfs dfs -ls → mostrara um arquivo com o resultado do job (ex: part-00000)
hdfs dfs -cat /saida/part-00000 (nome do arquivo resultado do comando acima) → mostra conteudo do arquivo no terminal

Obs.: quando usamos o MRJob, ele já faz isso automaticamente

IMPORTANTE – Antes de executar, é necessário informar ao hadoop onde se encontra o interpretador da linguagem python (ou ocorrerá erro na execução do job)

which python → informa o local do interpretador python na vm

```
## mapper.py ##

# A linha abaixo esta ativa, mesmo com o "#" no inicio
# Ela informa o local onde se encontra o interpretador da linguagem python

#!/usr/bin/env python

import sys # pacote da linguagem python para manipular o sistema operacional

for line in sys.stdin: # percorre entrada dos dados
    data = line.strip().split(" ") #  divide o arquivo sempre que encontrar " " e divide a linha com strip
    if len(data) == 10: # se o comprimento for igual a 10, distrribui nas 10 variaveis
        ip_address, identity, username, datetime, timezone, method, path, proto, status, size = data
        print ip_address
```

```
## reducer.py ##

#!/usr/bin/env python

import sys

current_ip_address = None # inicializando as variaveis
current_ip_address_count = 0

for line in sys.stdin:
    new_ip_address = line.strip().split()
    if len(new_ip_address) != 1: # se a linha for 1, considera uma linha invalida
        continue

    if current_ip_address and current_ip_address != new_ip_address: # se for igual ao None (da inicializacao) e diferente de new_ip_address
        print "{0}\t{1}".format(current_ip_address, current_ip_address_count)
        current_ip_address_count = 0

    current_ip_address = new_ip_address
    current_ip_address_count += 1

if current_ip_address != None:
    print "{0}\t{1}".format(current_ip_address, current_ip_address_count)
```

Quiz

Considerando um fator de replicação igual a 3, para armazenar 100 GB de dados, um Cluster Hadoop precisaria de 300 GB de espaço em disco.

Se você executar um Job MapReduce a especificar um diretório existente no HDFS para armazenar a saída, o Job vai falhar.

O que ocorre no processo de MapReduce, entre as fases de mapeamento e redução, são processos de Shuffle e Sort.

A computação distribuída consiste na utilização de um conjunto de máquinas conectadas por uma rede de comunicação, atuando como um único sistema.

Quando um programa é desenvolvido em um ambiente distribuído ou multiprocessado, o programador precisa estar ciente de toda estrutura do sistema, especificando as referências de cada dispositivo, para que possa estabelecer uma comunicação e sincronização entre os processos.

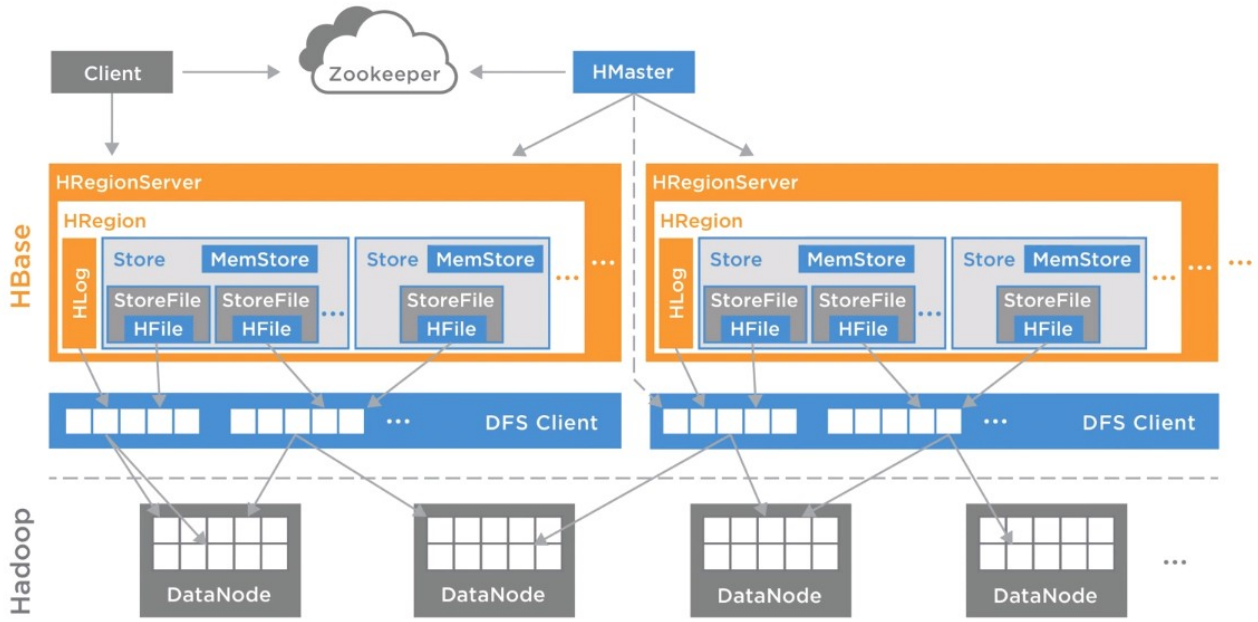
6.6. Armazenamento de Dados com HBase e Hive

Apache HBase

O Apache HBase é um banco de dados não relacional, orientado a colunas.

A vantagem de dividir as colunas em famílias é que você pode colocar cada família num servidor diferente, você pode particionar uma família, ou seja, permite fazer arranjos com as tabelas, o que não seria possível com bancos de dados relacionais.

O HBase roda sobre o HDFS.



DFS – Distributed File System

Para executar o HBase, abra o terminal e digite: hbase shell

Comando	**Descrição**
version	Versão do Hbase
status	Status do Hbase
help	Lista todos os comandos
table_help	Help
whoami	Usuário logado
status ‘simple’	Status simples
status ‘summary’	Status sumarizado
status ‘detailed’	Status detalhado

• Criar tabela: create ‘dsacademy’, {NAME=>’ALUNO’}, {NAME=>’INSTRUTOR’}

• Visualizar a estrutura da tabela: describe ‘dsacademy’

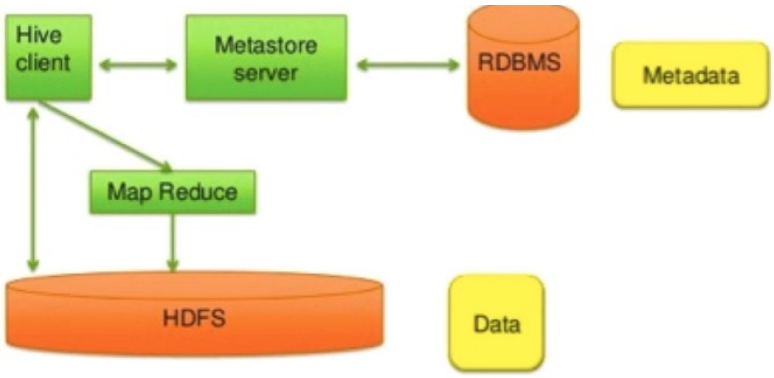
• Listando as tabelas: list

• Verificando se a tabela existe: exists ‘table1’

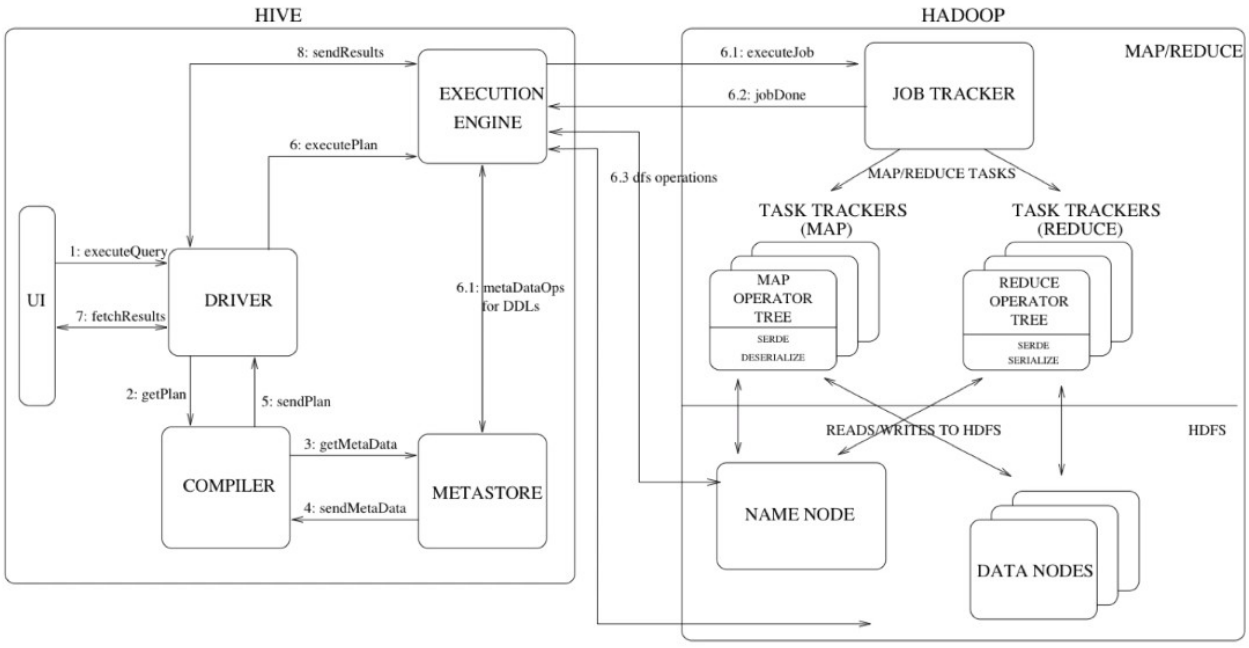
• Deletando uma tabela: disable ‘dsacademy’ drop ‘dsacademy’

Apache Hive

Permite interagir com os dados usando uma interface similar à de um banco de dados relacional. Para isso, utiliza-se o HQL (SQL).



O Apache Hive guarda os metadados em um banco de dados relacional, como MySQL, PostGree. Porém, os dados estão no HDFS.



Conhecendo o Apache HBase

HBase é mais um ”DataStore” do que um ”Database”.

HBase é um banco de dados distribuído, open-source, nãorelacional, inspirado no Google Big Table.

Principais características do HBase:

• Escalabilidade horizontal
• Processos consistentes de leitura/escrita (Hbase Read/Hbase Write)
• Particionamento automático
• Recuperação automática de falhas
• Java API para acesso aos dados

HBase x HDFS:

HBase	**HDFS**
HBase é um banco de dados NoSQL construído para trabalhar sobre o HDFS.	Sistema de arquivos distribuído para armazenamento de grandes conjuntos de dados.
Suporta consultas a grandes tabelas de dados.	Não suporta consultas a registros individuais de dados.
Baixa latência de acesso aos dados, mesmo em tabelas de bilhões de registros.	Alta latência e processamento em batch.
HBase armazena dados em formato key/value.	Armazena os dados em arquivos.

HBase x RDBMS:

HBase	**RDBMS**
Utiliza regiões.	Utiliza tabelas.
Suporta o filesystem HDFS.	Suporta filesystems FAT, NTFS, EXT, NFS.
Conceito de Write-Ahead Logs (WAL) para armazenar alterações nos dados	Conceito de commit logs para armazenar as alterações nos dados.
A coordenação dos processos é feita pelo Apache Zookeeper.	A coordenação dos processos é feita pelo sistema gerenciador de bancos de dados (Oracle, SQL Server, MySQL, etc...)
Linhas são identificadas unicamente pelas rowkeys.	Linhas são identificadas unicamente por chaves primárias.
Regiões podem ser particionadas.	Tabelas podem ser particionadas.
Conceito de linha, família de colunas, coluna e célula.	Conceito de linha, coluna e célula.
Suporta bilhões de registros.	Apresenta problemas de performance com bilhões de registros.

Afinal, Quando Usar e Quando Não Usar o HBase?

Quando Utilizar HBase?

- Dados não-estruturados ou semi-estruturados
- Alta escalabilidade
- Dados versionados
- Quando é necessário acesso baseado em chave
- Alto volume de dados devem ser armazenados
- Armazenamento de dados orientado a coluna

Quando Não Utilizar HBase?

- Poucas linhas devem ser armazenadas
- Não for necessário realizar consultas cruzadas (SQL Joins)
- Cluster com poucas máquinas

Normalização x Desnormalização de Dados

Outra importante diferença entre HBase e bancos relacionais é a normalização. O objetivo da normalização é evitar os problemas provocados por falhas no Projeto do Banco de Dados, bem como eliminar a "mistura de assuntos" e as correspondentes repetições desnecessárias de dados.

Uma Regra de Ouro que devemos observar quando do Projeto de um Banco de Dados baseado no Modelo Relacional é a de "não misturar assuntos em uma mesma Tabela". o HBase tem o conceito oposto. Como podemos criar famílias de colunas, devemos ter o menor número possível de colunas com dados representando assuntos diferentes na mesma tabela. O HBase possui um paradigma diferente e seu foco é armazenamento de Big Data.

RDBMS → Normalização (Maior número de tabelas)
HBase → Desnormalização (Menor número de tabelas)

Banco de Dados NoSQL

Bancos de Dados tradicionais RDBMS (Relational Database Management Systems) não foram projetados para tratar grandes quantidades de dados (Big Data). Bancos de Dados tradicionais foram projetados somente para tratar conjuntos de dados que possam ser armazenados em linhas e colunas e, portanto, possam ser consultados através do uso de queries utilizando linguagem SQL (Structured Query Language). Bancos de Dados relacionais não são capazes de tratar dados não-estruturados ou semiestruturados. Ou seja, Bancos de Dados relacionais simplesmente não possuem funcionalidades necessárias para atender os requisitos do Big Data, dados gerados em grande volume, alta variedade e alta velocidade.

Mas esta lacuna está sendo preenchida por Bancos de Dados NoSQL. Um banco de dados NoSQL é um banco de dados que não incorpora um modelo relacional baseado em tabelas e chaves primária/estrangeira, próprios de bancos de dados relacionais. Os bancos de dados NoSQL nasceram na era do Big Data e são apropriados para tratar dados não estruturados, que não possuem um relacionamento claro e bem definido. Bancos de Dados NoSQL oferecem uma arquitetura muito mais escalável e eficiente que os bancos relacionais e facilitam consultas no-sql de dados semiestruturados ou não-estruturados.

Principais Características de Bancos de Dados NoSQL:

• Ideais para soluções analíticas
• Modelos de dados flexíveis
• Escalabilidade
• Representação de dados sem esquemas

Aqui você encontra uma lista de Bancos de Dados NoSQL disponíveis: http://nosql-database.org/

Bancos de Dados NoSQL oferecem 4 categorias principais de bancos de dados:

- Graph databases (Neo4J, FlockDB, GraphDB, ArangoDB)
    - Esta categoria de Bancos de Dados NoSQL, geralmente é aderente a cenários de rede social on-line, onde os nós representam as entidades e os laços representam as interconexões entre elas. Desta forma, é possível atravessar o gráfico seguindo as relações. Esta categoria tem sido usada para lidar com problemas relacionados a sistemas de recomendação e listas de controle de acesso, fazendo uso de sua capacidade de lidar com dados altamente interligados.
- Document databases (MongoDB, CouchDB, RavenDB, Terrastore)
    - Esta categoria de Bancos de Dados NoSQL permite o armazenamento de milhões de documentos. Por exemplo, você pode armazenar detalhes sobre um empregado, junto com o currículo dele (como um documento) e então pesquisar sobre potenciais candidatos a uma vaga, usando um campo específico, como telefone ou conhecimento em uma tecnologia.
- Key-values stores (Oracle NoSQL DB, MemcacheDB, Redis, Amazon DynamoDB)
    - Nesta categoria, os dados são armazenados no formato key-value (chave-valor) e os valores (dados) são identificados pelas chaves. É possível armazenar bilhões de registros de forma eficiente e o processo de escrita é bem rápido. Os dados podem ser então pesquisados através das chaves associadas.
- Column family stores (HBase, Cassandra, Hypertable, Accumulo)
    - Também chamados bancos de dados orientados a coluna, os dados são organizados em grupos de colunas e tanto o armazenamento, quando as pesquisas de dados são baseadas em chaves. HBase é um banco de dados NoSQL orientado a famílias de colunas. Mais a frente veremos o que é esse conceito e como criar famílias de colunas em tabelas do HBase.

Quiz

O Hbase foi inspirado no Google Big Table, sendo um dos projetos top 10 da Apache Foundation. O Hbase não é necessariamente um banco de dados, sendo considerado um datastore, pois ele não possui diversas características de um banco de dados convencional, como triggers, stored procedures, índices secundários, etc…

O Apache Hive é um sistema de armazenamento de dados para o Hadoop, que permite o resumo, consulta e análise de grandes volumes de dados usando o HiveQL (linguagem de consulta semelhante ao SQL). O Hive pode ser usado para explorar os dados interativamente ou para criar jobs de processamento em batch.

Características do HBase:
Hbase é um banco de dados NoSQL construído para trabalhar sobre o HDFS.
Baixa latência de acesso aos dados, mesmo em tabelas de bilhões de registros.
Armazenamento de dados em formato key/value.

Característica do HDFS:
Não suporta consultas a registros individuais de dados.

Os comados existentes no SQL tais como create table, select, describe, drop table, etc existem no Hive, mas tem diferenças tal como a clausula ROW FORMAT do create table.

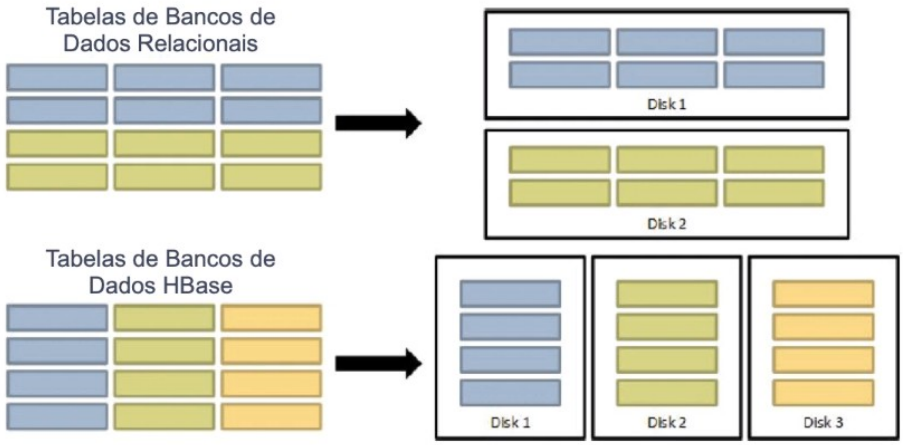
O Hive foi desenhado para executar queries em real time, com baixa latência! **(FALSO!!!)**

6.6. Armazenamento de Dados com HBase e Hive

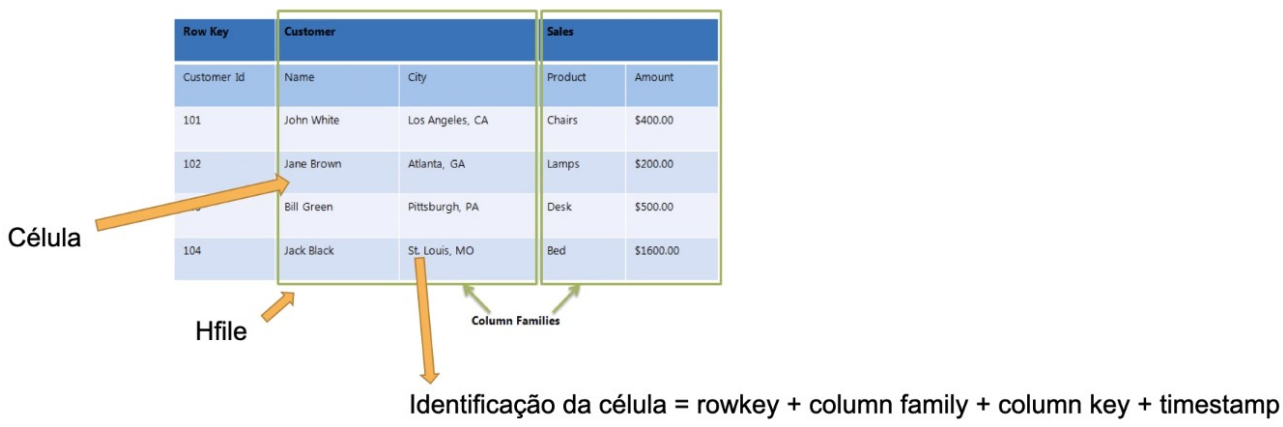
Apache HBase

RDBMS → Escalabilidade Vertical

HBase → Escalabilidade Horizontal – Tabelas divididas em partições e distribuídas no cluster



Timestamp é uma sequência de caracteres que identifica quando um evento ocorre e normalmente com dados de hora no nível de fração de segundos.



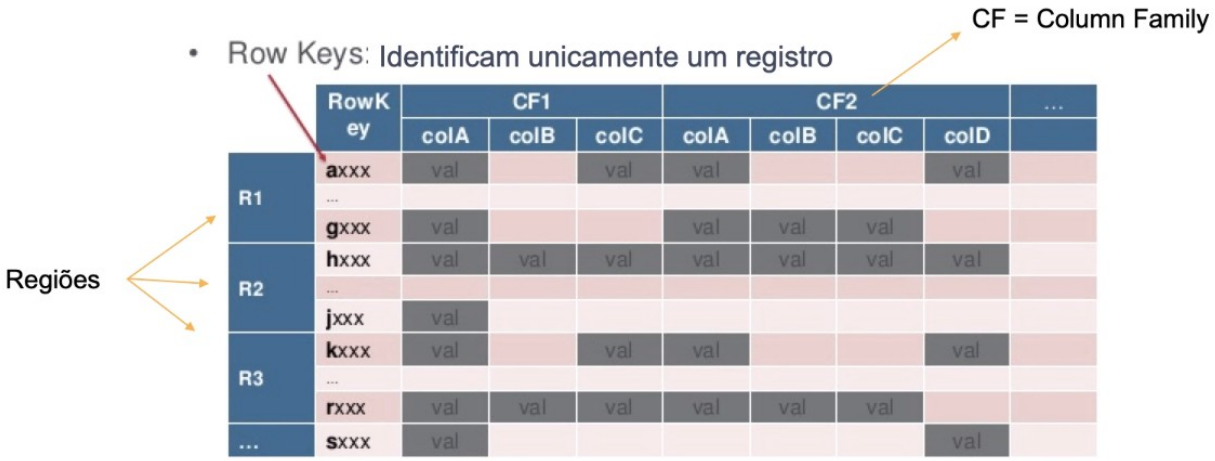
Identificação da célula = rowkey + column family + column key + timestamp

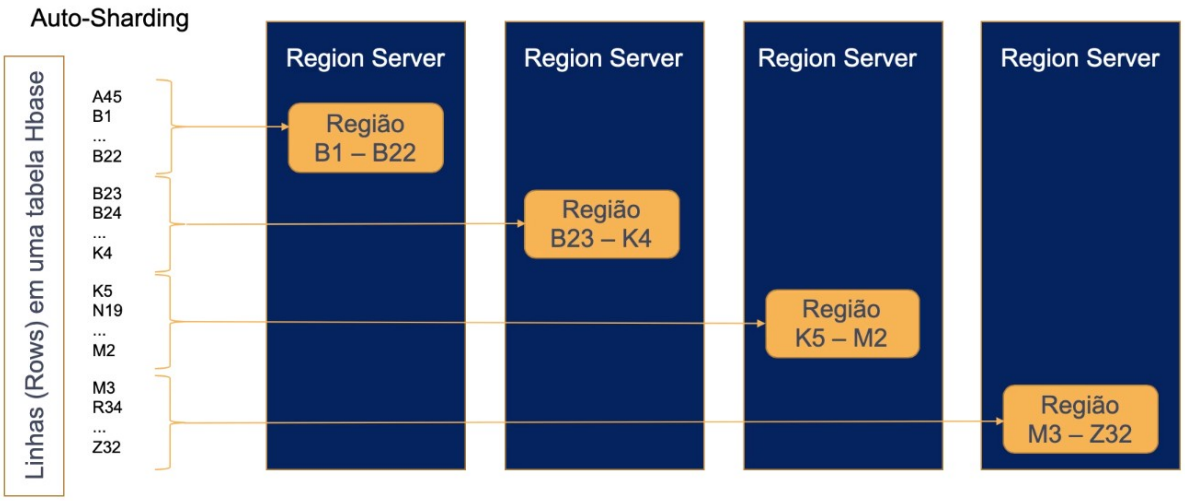
Row Keys: Identificam unicamente um registro

CF = Column Family

Regiões são partições da tabela.

Auto-Sharding é o processo de agrupar linhas em regiões.





Orientação a Linha x Orientação a Coluna

Bancos de Dados relacionais são orientados a linha (quanto mais tabelas, melhor):

Tabela Clientes

ID Cliente	Nome	Cidade
1	Bob	SP
2	Zico	RJ

Tabela Vendas

ID Cliente	ID Produto	Valor
1	1002	500
2	1008	700

O HBase é orientado a coluna (quanto menos tabelas, melhor):

Rowkey	Clientes		Vendas	
ID Cliente	Nome Cliente	Cidade	ID Produto	Valor
1	Bob	SP	1002	500
2	Zico	RJ	1008	700

Arquitetura HBase

Fisicamente, o HBase é composto por três tipos de servidores em uma arquitetura mestre/escravo. Servidores de Região servem os dados para leituras e gravações.

Ao acessar dados, os clientes se comunicam diretamente com os Region Servers. As operações de atribuição de região e DDL (criar e excluir tabelas) são tratadas pelo processo do HBase Master. O Zookeeper mantém o estado de cluster ativo.

O Hadoop DataNode armazena os dados que o Region Server está gerenciando. Todos os dados do HBase são armazenados em arquivos HDFS.

Os Region Servers são colocados com os DataNodes do HDFS, que permitem a localidade dos dados (colocando os dados perto de onde são necessários) para os dados servidos pelos Region Servers.

O NameNode mantém informações de metadados para todos os blocos de dados físicos que compõem os arquivos.

As tabelas HBase são divididas horizontalmente pelo intervalo de rowkeys em "Regiões".

Uma região contém todas as linhas da tabela entre a chave inicial e a chave final da região.

As regiões são atribuídas aos nós do cluster, chamados Region Servers. Um servidor de região pode atender cerca de 1.000 regiões.

Configurando o HBase em Modo Pseudo-Distribuído

Abra o terminal da VM
start-dfs.sh → inicializa o cluster HDFS
    * namenodes, datanodes e secondary namenodes

start-yarn.sh → inicializa o yarn para processamento de jobs mapreduce
    * resourcemanager, nodemanagers

hdfs dfs -ls / → verifica o que existe dentro do sistema de arquivos distribuídos

cd /opt/hbase/ → acessa pasta do hbase
ls -la → mostra conteudo da pasta
cd conf/ → pasta com os arquivos de configuração

hbase-env.sh e hbase-site.xml → são os 2 arquivos de parâmetros que necessitam de configuração

gedit hbase-env.sh → abre o editor
descomente as linhas abaixo:
    export HBASE_MANAGES_ZK=true
    export HBASE_REGIONSERVERS=$(HBASE_HOME)/conf/regionservers
    export HBASE_SSH_OPTS=”-p 22 -l hadoop” (essa linha deve ser incluída abaixo do HBASE_SSH_OPTS original)
salve o arquivo.

gedit hbase-site.xml → abre o editor
substitua a tag configuration por:

```
<configuration>
 <property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:9000/hbase</value>
 </property>
 <property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
 </property>
 <property>
  <name>hbase.zookeeper.quorum</name>
  <value>localhost</value>
</property>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>
 <property>
  <name>hbase.zookeeper.property.clientPort</name>
  <value>2181</value>
 </property>
 <property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/opt/zookeeper/data</value>
 </property>
 <property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>false</value>
 </property>
</configuration>
```

salve o arquivo.

start-hbase.sh → inicializa o hbase em modo pseudodistribuído
* HMaster, HRegionServer, HQuorumPeer

Importando Dados para o HBase com Pig

Parte 1 - Carregar dados no HDFS

Criar diretório

```
hdfs dfs -mkdir /user/dados
hdfs dfs -mkdir /user/dados/clientes
```

Carregar dados no HDFS

```
hdfs dfs -copyFromLocal clientes.txt /user/dados/clientes
```

Parte 2 - Criar tabela no HBase

```
# Abrir o shell
hbase shell

# Cria uma tabela com uma column family para receber os dados
create 'clientes', 'dados_clientes'
```

Parte 3 - Carregar dados com Pig

```
# Inicie o Job History Server
mr-jobhistory-daemon.sh start historyserver

# Abrir o shell do Pig
pig -x mapreduce (em caso de probemas, use pig -x local)

# Navegar até o diretório
cd /user/dados/clientes;

# Visualizar o conteúdo do arquivo
cat clientes.txt;

# Carregar dados do HDFS para o Pig
dados = LOAD 'clientes.txt' USING PigStorage(',') AS (
        id:chararray,
        nome:chararray,
        sobrenome:chararray,
        idade:int,
        funcao:chararray
);

# Testa os dados
dump dados;

# Usando Pig Store

STORE dados INTO 'hbase://clientes' USING org.apache.pig.backend.hadoop.hbase.HbaseStorage(
'dados_clientes:nome
 dados_clientes:sobrenome
 dados_clientes:idade
 dados_clientes:funcao'
);
```

```
# Parte 3 - Manipulando dados no HBase

# Abrir o shell
hbase shell
```

```
# Scan na tabela
scan 'clientes'
count 'clientes'
get 'clientes', '1100002', {COLUMN => 'dados_clientes:nome'}
put 'clientes', '1100002', 'dados_clientes:nome', 'Bob'
put 'clientes', '2100002', 'dados_clientes:nome', 'Zico'
put 'clientes', '2100002', 'dados_clientes:sobrenome', 'Galinho'
put 'clientes', '2100002', 'dados_clientes:idade', '50'
put 'clientes', '2100002', 'dados_clientes:funcao', 'Jogador'
delete 'clientes', '2100002', 'dados_clientes:funcao'
deleteall 'clientes', '2100002'
```

Laboratório 4 – Pipeline de Dados no HBase com API Java

?

```
// EnableTable.java:

package pipeop;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class EnableTable{

  public static void main(String args[]) throws IOException{

    Configuration conf = HBaseConfiguration.create();

    HBaseAdmin admin = new HBaseAdmin(conf);

    Boolean bool = admin.isTableEnabled("RH");
    System.out.println(bool);

    if(!bool){
      admin.enableTable("RH");
      System.out.println("A Tabela foi habilitada");
    }
    else
    {
            System.out.println("Tabela habilitada");
    }
  }
}
```

```
// Hbase_create.java:

package pipeop;
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.conf.Configuration;

public class Hbase_create {

  public static void main(String[] args) throws IOException {

    // Instanciando a classe de configuracao
    Configuration con = HBaseConfiguration.create();

    // Instanciando a classe HbaseAdmin
    HBaseAdmin admin = new HBaseAdmin(con);
```

```
    // Instanciando a table descriptor
    HTableDescriptor tableDescriptor = new HTableDescriptor(TableName.valueOf("RH"));

    // Adicionando familias de colunas
    tableDescriptor.addFamily(new HColumnDescriptor("pessoal"));
    tableDescriptor.addFamily(new HColumnDescriptor("profissional"));

    // Criando a tabela
    admin.createTable(tableDescriptor);
    System.out.println(" Tabela criada ");
  }
}
```

```
// Hbase_disable.java

package pipeop;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class Hbase_disable{

  public static void main(String args[]) throws  IOException{

    Configuration conf = HBaseConfiguration.create();

    HBaseAdmin admin = new HBaseAdmin(conf);

    Boolean bool = admin.isTableDisabled("RH");
    System.out.println(bool);

    if(!bool){
      admin.disableTable("RH");
      System.out.println("\nTabela desativada");
    }
    else
    {
            System.out.println("\nTabela desativada");
    }
  }
}
```

```
// Hbase_drop.java:

package pipeop;

import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class Hbase_drop {

  public static void main(String[] args) throws IOException {

    Configuration conf = HBaseConfiguration.create();

    HBaseAdmin admin = new HBaseAdmin(conf);

    admin.disableTable("RH");

    admin.deleteTable("RH");
    System.out.println("Tabela deletada");
  }
}
```

```
//Hbase_insert.java:

package pipeop;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
```

```
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

public class Hbase_insert{

  public static void main(String[] args) throws IOException {


    Configuration config = HBaseConfiguration.create();

    HTable hTable = new HTable(config, "RH");

    // Instanciando a classe put
    Put p = new Put(Bytes.toBytes("linha1"));

    // Adicionando valores
    p.add(Bytes.toBytes("pessoal"), Bytes.toBytes("nome"),Bytes.toBytes("Bob"));

    p.add(Bytes.toBytes("pessoal"), Bytes.toBytes("cidade"),Bytes.toBytes("Fortaleza"));

    p.add(Bytes.toBytes("profissional"),Bytes.toBytes("cargo"),Bytes.toBytes("Analista"));

    p.add(Bytes.toBytes("profissional"),Bytes.toBytes("salario"),Bytes.toBytes("25000"));

    // Salvando os dados na tabela
    hTable.put(p);
    System.out.println("Dados inseridos");

    // Encerrando a conexão
    hTable.close();
  }
}
```

```
// Hbase_read.java:

package pipeop;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;


public class Hbase_read{

  @SuppressWarnings("deprecation")
public static void main(String args[]) throws IOException{

    // Instanciando a classe
    Configuration config = HBaseConfiguration.create();

    // Instanciando a tabela no Hbase
    HTable table = new HTable(config, "RH");

    // Instanciando uma classe scan
    Scan scan = new Scan();

    // Scan nas colunas
    scan.addColumn(Bytes.toBytes("pessoal"), Bytes.toBytes("nome"));
    scan.addColumn(Bytes.toBytes("profissional"), Bytes.toBytes("cidade"));

    // Obtendo o resultado
    ResultScanner scanner = table.getScanner(scan);
try {

        for (Result result = scanner.next(); (result != null); result = scanner.next()) {
            for(KeyValue keyValue : result.list()) {
               System.out.println("Chave : " + keyValue.getKeyString() + " : Valor : " + Bytes.toString(keyValue.getValue()));
            }
          }
}
finally
```

```
{
      //Encerrando o scan
      scanner.close();
    }
    }
}
```

```
// ListTables.java:

package pipeop;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class ListTables {

   public static void main(String args[])throws  IOException{

      // Instanciando a classe
      Configuration conf = HBaseConfiguration.create();

      // Instanciando a classe HBaseAdmin

      HBaseAdmin admin = new HBaseAdmin(conf);

      // Ontendo todas as tabelas
      HTableDescriptor[] tableDescriptor = admin.listTables();

      // Imprimindo o nome das tabelas
      for (int i=0; i<tableDescriptor.length;i++ ){
         System.out.println(tableDescriptor[i].getNameAsString());
      }

   }
}
```

Introdução ao Apache Hive

O Hive é um framework para soluções de Data Warehousing, que executa no ambiente do Hadoop, construído inicialmente pelo time de desenvolvimento do Facebook em 2007.

Outro ponto que levou ao desenvolvimento do Hive foi o baixo desempenho das soluções de mercado para realizar operações de Full Scan em grandes volumes de dados.

O Apache Hive utiliza uma linguagem chamada Hive Query Language. Ela utiliza linguagem SQL em Jobs MapReduce, executados no cluster Hadoop.

O que o Hive não é?

• Um banco de dados relacional
• Um projeto para Online Transaction Processing (OLTP) / Similar a um Data Warehouse
• Uma solução para consultas em tempo real e atualizações em nível de linha

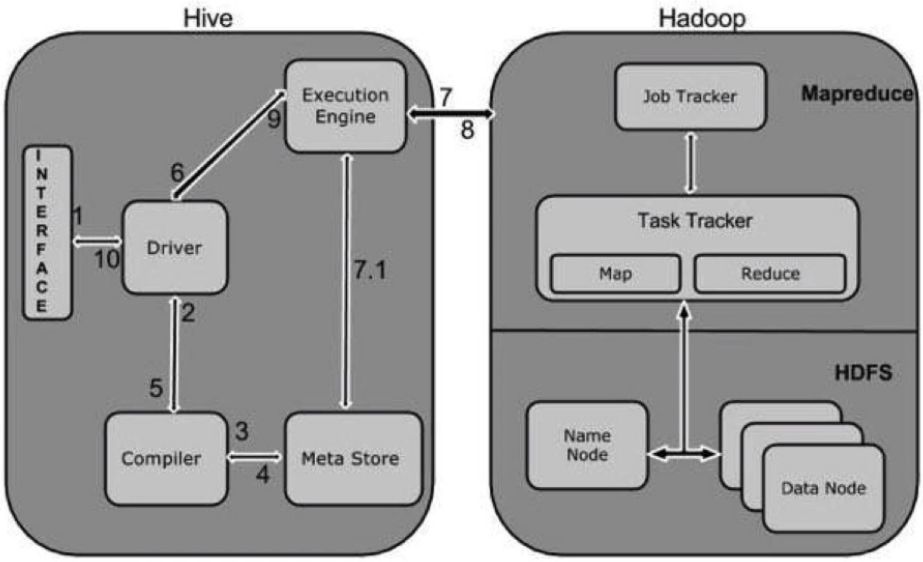
Usamos o Apache Hive quando precisarmos realizar consultas ou manipulações em grandes conjuntos de dados, tais como seleção de registros ou colunas, agregação, sumarização, contagem de elementos, filtros ou atualizações em massa. Essas tarefas não precisam ser feitas em tempo real e o que queremos é obter insights a partir de grandes conjuntos de dados, Big Data.

Arquitetura do Apache Hive

Para acessar os dados no Hive podem ser utilizadas a própria linha de comando, a interface do Apache Unbody, drivers JDBC e o Hive Thrift Pointer.

O Apache Hive não foi criado para executar queries em tempo real com baixa latência. Ele é indicado para grande volume de dados, em clusters de computadores. Muito utilizado para aplicações analíticas.

Para suportar esquemas e particionamento de tabelas, o Hive deve manter os metadados em um banco relacional.



Tipos de Dados do Apache Hive

- Tipos de Coluna (Int, Smallint, Tinyint, Bigint) (Char, Varchar) Dates e Timestamp
- Tipos Literais (Ponto Flutuante e Decimal)
- Tipo Nulo
- Tipos Complexos (Arrays, Maps, Structs)

Manipulação de Dados com Apache Hive

Inicie o cluster Hadoop
jps → mostra serviços em execução
start-dfs.sh → inicializa o hdfs
start-yarn.sh → gestão do mapreduce

hive → inicializa o hive

Comandos do Hive:

show databases; → mostra os bancos de dados
create database nomebanco; → cria novo banco de dados
use nomebanco; → utiliza o banco criado
show tables; → mostra as tabelas do banco de dados em uso

Cria tabela:

```
CREATE TABLE IF NOT EXISTS colaboradores (id int, nome String, cargo String, salario decimal)
COMMENT 'tabela de colaboradores'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```

```
ALTER TABLE colaboradores CHANGE salario salario Double;
ALTER TABLE colaboradores ADD COLUMNS (cidade String COMMENT 'Nome da Cidade');
```

describe colaboradores; → verifica a estrutura da tabela

create table temp_colab (texto String); → criação de tabela temporária

LOAD DATA LOCAL INPATH '/home/hadoop/colaboradores.csv' OVERWRITE INTO TABLE temp_colab; → carregando os dados na tabela

SELECT * FROM temp_colab;

INSERT overwrite table colaboradores → insert com select

```
SELECT
  regexp_extract(texto, '^(?:([^,]*),?){1}', 1) ID,
  regexp_extract(texto, '^(?:([^,]*),?){2}', 1) nome,
  regexp_extract(texto, '^(?:([^,]*),?){3}', 1) cargo,
  regexp_extract(texto, '^(?:([^,]*),?){4}', 1) salario,
  regexp_extract(texto, '^(?:([^,]*),?){5}', 1) cidade
FROM temp_colab;
```

→ Obs.: consultar documentação do hive para uso de expressões regulares

```
SELECT * FROM colaboradores;
SELECT * FROM colaboradores WHERE Id = 1002;
SELECT * FROM colaboradores WHERE salario >= 25000;
SELECT * FROM colaboradores WHERE salario > 10000 AND cidade = 'Natal';
```

6.8. Conectividade ETL (Extract, Transform, Load) com o Sistema Hadoop

O que é ETL?

ETL fornece a infraestrutura de integração através da realização de três importantes funções:
FONTE > Extrair > Transformar > Carregar > DESTINO

Qual é o Papel do ETL no Big Data?

A ideia é que se possa armazenar todos os dados de uma empresa, em um único repositório. E com a escalabilidade horizontal, esse repositório pode ir crescendo a medida em que se acrescentam outras máquinas. Aqui se encontra a figura do Data Lake.

Principais Ferramentas ETL

Principais Ferramentas ETL - Proprietárias

- Informatica Power Center
- IBM InfoSphere Data Stage
- Oracle Data Integrator (ODI)
- Microsoft – SQL Server Integration Services (SSIS)
- SAS – Data Integration Studio
- SAP – Business Object Integrator
- Pentaho Data Integration

Principais Ferramentas ETL - Open Source

- Dataiku Data Science Studio (DSS) Community Edition
- Talend Open Studio For Data Integration
- Jaspersoft ETL
- Jedox
- RapidMiner
- Apache Flume (tempo real)
- Apache NiFi (tempo real)
- Apache Sqoop (batch)

Preparando o Ambiente com Banco de Dados Oracle

```
# Oracle 19c - Instalação

1- Efetuar login como usuário root.

2- Atualizar o SO:
```

yum update -y

3- Editar o arquivos /etc/hosts e incluir o nome da máquina como FQDN com o ip da máquina virtual (não usar localhost).

4- Editar o arquivo /etc/sysctl.conf e incluir as linhas abaixo.

```
# Oracle
fs.file-max = 6815744
kernel.sem = 250 32000 100 128
kernel.shmmni = 4096
kernel.shmall = 1073741824
kernel.shmmax = 4398046511104
kernel.panic_on_oops = 1
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
net.ipv4.conf.all.rp_filter = 2
net.ipv4.conf.default.rp_filter = 2
fs.aio-max-nr = 1048576
net.ipv4.ip_local_port_range = 9000 65500
```

5- Efetivar as alterações do item 4: /sbin/sysctl -p

6- Adicionar as linhas abaixo para o arquivo /etc/security/limits.conf

```
# Oracle
oracle   soft   nofile    1024
oracle   hard   nofile    65536
oracle   soft   nproc    16384
oracle   hard   nproc    16384
oracle   soft   stack    10240
oracle   hard   stack    32768
oracle   hard   memlock    134217728
oracle   soft   memlock    134217728
```

7- Instalar os pacotes abaixo no SO.

yum install -y bc binutils compat-libcap1 compat-libstdc++-33 elfutils-libelf elfutils-libelf-devel fontconfig-devel glibc glibc-devel ksh libaio libaio-devel libdtrace-ctf-devel libXrender libXrender-devel libX11 libXau libXi libXtst libgcc librdmacm-devel libstdc++ libstdc++-devel libxcb make net-tools nfs-utils smartmontools sysstat unixODBC

8- Criar os grupos no SO.

```
groupadd -g 54321 oinstall
groupadd -g 54322 dba
groupadd -g 54323 oper
```

9- Adicionar usuário owner da instalação Oracle.

```
useradd -u 54321 -g oinstall -G dba,oper oracle
passwd oracle
```

10- Editar o arquivo /etc/selinux/config e adicionar a linha abaixo.

SELINUX=permissive

E então executar: setenforce Permissive

11- Desativar o firewall.

```
systemctl stop firewalld
systemctl disable firewalld
```

12- Criar os diretórios de instalação.

mkdir -p /u01/app/oracle/product/19.0.0/dbhome_1

```
mkdir -p /u02/oradata
chown -R oracle:oinstall /u01 /u02
chmod -R 775 /u01 /u02
```

13- Criar um diretório de scripts.

```
mkdir /home/oracle/scripts
```

14- Criar o arquivo de variáveis do usuário Oracle e incluir as linhas abaixo.

```
gedit /home/oracle/scripts/setEnv.sh

# Oracle
export TMP=/tmp
export TMPDIR=$TMP
export ORACLE_HOSTNAME=dataserver.localdomain
export ORACLE_UNQNAME=orcl
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=$ORACLE_BASE/product/19.0.0/dbhome_1
export ORA_INVENTORY=/u01/app/oraInventory
export ORACLE_SID=orcl
export DATA_DIR=/u02/oradata
export PATH=/usr/sbin:/usr/local/bin:$PATH
export PATH=$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:/lib:/usr/lib
export CLASSPATH=$ORACLE_HOME/jlib:$ORACLE_HOME/rdbms/jlib
```

15- Adicionamos às variáveis de ambiente ao arquivo e profile do usuário oracle.

```
echo ". /home/oracle/scripts/setEnv.sh" >> /home/oracle/.bash_profile
```

16- Efetuar login como usuário oracle e fazer download do Oracle 19c (2.8 GB)
https://www.oracle.com/database/technologies/oracle19c-linux-downloads.html

```
cd $ORACLE_HOME
unzip -oq /home/oracle/Downloads/LINUX.X64_193000_db_home.zip

./runInstaller
```

Obs: Se receber mensagem de erro durante a instalação, abra um terminal e adicione as linhas abaixo no arquivo /home/oracle/.bashrc e não esqueça de executar source .bashrc. Depois clique em Retry no instalador.

```
# Java JDK
export JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin
```

17 - Checar o status e iniciar o listener.

```
lsnrctl status
lsnrctl start
lsnrctl stop
```

18- Conectar no Banco de Dados e executar os comandos abaixo para registrar o banco no listener (use o ip da sua VM).

```
sqlplus / as sysdba
show parameter local_listener
alter system set local_listener='(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.122.1)(PORT=1539))';
ALTER SYSTEM REGISTER;
```

19- Criar o arquivo /u01/app/oracle/product/19.0.0/dbhome_1/network/admin/tnsnames.ora com este conteúdo:

```
orcl=
 (DESCRIPTION=
   (ADDRESS=(PROTOCOL=tcp)(HOST=dataserver.localdomain)(PORT=1539))
   (CONNECT_DATA=
     (SERVICE_NAME=orcl)))
```

20- Testar a conexão.

```
lsnrctl status → testar status da conexão (se há listener ou não)
```

lsnrctl start → inicializa o listener (serviço ouvinte – serviço de rede que abre uma porta no SO e fica ouvindo-a)
sqlplus / as sysdba → conecta ao banco de dados oracle

demais procedimentos no item 18

Carregando Dados no Oracle

########## Conectar no SO como usuário oracle ##########

Cria o schema no Banco de Dados e concede privilégios
sqlplus / as sysdba
create user aluno identified by dsahadoop;
grant connect, resource, unlimited tablespace to aluno;

Cria uma tabela
sqlplus aluno@orcl

```
CREATE TABLE cinema (
  ID   NUMBER PRIMARY KEY ,
  USER_ID       VARCHAR2(30),
  MOVIE_ID      VARCHAR2(30),
  RATING        DECIMAL,
  TIMESTAMP     VARCHAR2(256) );
```

Fazer download do arquivo para carga de dados.
http://files.grouplens.org/datasets/movielens/ml-20m.zip
unzip ml-20m.zip

Para carregar dados no Oracle, usamos o SQL*Loader. Este aplicativo requer um control file conforme abaixo:

```
mkdir ~/etl
cd ~/etl

gedit loader.dat

load data
INFILE '/home/oracle/Downloads/ml-20m/ratings.csv'
INTO TABLE cinema
APPEND
FIELDS TERMINATED BY ','
trailing nullcols
(id SEQUENCE (MAX,1),
 user_id CHAR(30),
 movie_id CHAR(30),
 rating   decimal external,
 timestamp  char(256))
```

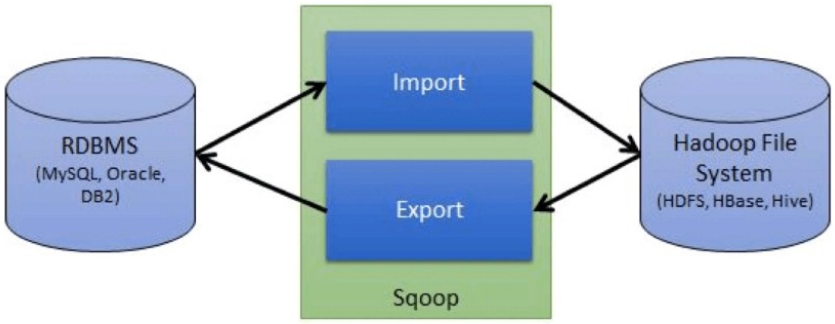
Executando o SQL*Loader
sqlldr userid=aluno/dsahadoop control=loader.dat log=loader.log

Verificando os dados
sqlplus aluno@orcl

```
select count(*) from cinema;
```

Sqoop (SQL to Hadoop)

O Sqoop é uma ferramenta simples, onde se tem import e export.



Na prática, ele cria um job mapreduce para mover dados entre bancos relacionais e o Hadoop.

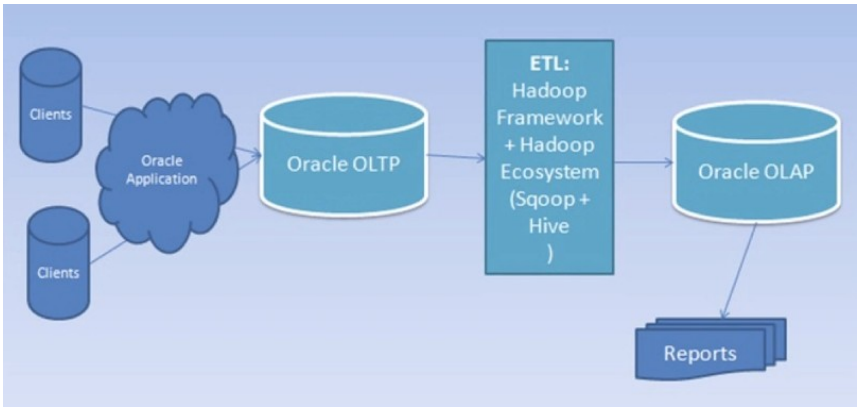
sqoop import (importa dados para o HDFS)
--connect jdbc:oracle:thin:aluno/dsahadoop @localhost:1521:orcl
--username aluno (define usuário)
--password dsahadoop (define senha)
--query "select user_id, movie_id from cinema where rating = 1 and \$CONDITIONS" (seleção de dados)
--target-dir /user/oracle/output (para onde vai os dados)

Principais Características do Sqoop:

- Import – Permite a importação de bancos de dados externos e enterprise data warehouses
- Transferência – Paraleliza a transferência de dados para melhorar performance e otimizar a utilização do sistema
- Cópia – Copia dados rapidamente de fontes externas para o Hadoop
- Aumento de Eficiência – Faz com que a análise de dados seja mais eficiente
- Diminuição de Carga – Evita cargas excessivas para sistemas externos

ETL Hadoop = Sqoop + Hive

É possível utilizar o sqoop junto com o hive, montando assim uma solução ETL completa para o ambiente com Apache Hadoop



Mini-Projeto 1 – Importando Dados de Banco de Dados Oracle para o HDFS

Sua empresa possui milhões de registros de avaliações de filmes e deseja usar esses dados para construir um sistema de recomendação de filmes para seus clientes. Os dados estão armazenados em um banco de dados relacional e a empresa possui um cluster Hadoop para armazenamento e processamento distribuídos. Seu trabalho é levar os dados da fonte para o HDFS para posterior análise.

```
# ETL Oracle/Hadoop com Sqoop

########## Conectar no SO como usuário hadoop ##########

# Como usuario hadoop, inicializar HDFS e Yarn
start-dfs.sh
start-yarn.sh


########## Conectar no SO como usuário oracle ##########

# Baixar o driver JDBC para o Sqoop
https://www.oracle.com/database/technologies/jdbc-ucp-122-downloads.html


# Descompactar o arquivo
tar -xvf ojdbc8-full.tar.gz


# Copiar o arquivo para o diretorio do sqoop
cp ojdbc8.jar /opt/sqoop/lib


# No usuario oracle, configurar as variáveis de ambiente para o Hadoop e Sqoop no arquivo ~/.bashrc

# Java JDK
export JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin

# Hadoop
export HADOOP_HOME=/opt/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

# Sqoop
export SQOOP_HOME=/opt/sqoop
export PATH=$PATH:$SQOOP_HOME/bin
export HCAT_HOME=/opt/sqoop/hcatalog
export ACCUMULO_HOME=/opt/sqoop/accumulo



# Como usuário hadoop, definir os privilégios com os comandos abaixo:s

hdfs dfs -chmod -R 777 /
chmod -R 777 /opt/hadoop/logs

# Como root:
groups oracle
usermod -a -G hadoop oracle


# Importação de Dados do Oracle para o HDFS

sqoop import --connect jdbc:oracle:thin:aluno/dsahadoop@dataserver.localdomain:1539/orcl --username aluno -password dsahadoop --query "select user_id, movie_id from cinema where rating = 1 and \$CONDITIONS" --target-dir /user/oracle/output -m 1

sqoop import -D mapreduce.map.memory.mb=1024 -D mapreduce.map.java.opts=-Xmx768m --connect jdbc:oracle:thin:aluno/dsahadoop@dataserver.localdomain:1539/orcl --username aluno -password dsahadoop --query "select user_id, movie_id from cinema where rating = 1 and \$CONDITIONS" --target-dir /user/oracle/output -m 1
```

```
# Checar o HDFS:
hdfs dfs -ls /user

# Comando usado para corrigir problemas com blocos corrompidos, caso ocorra
hdfs fsck / | egrep -v '^\.+$' | grep -v replica | grep -v Replica

# Para deixar o modo de segurança do Hadoop
hdfs dfsadmin -safemode leave
```

Quizz

O Sqoop é a principal ferramenta do ecosystema Hadoop e amplamente utilizada para a importação/exportação de dados do HDFS. E se combinado com o Hive, fornece uma robusta e gratuita solução de ETL.

O processo ETL é composto de 3 etapas. A primeira é a extração (extract), a segunda a transformação (transform) e por fim, a carga (load). Cada uma delas possui grande importância para o sucesso da transição dos dados dos sistemas de origem para o sistema destino. Ou seja, a função do ETL é movimentar os dados entre diferentes sistemas.

Sqoop é predominantemente operado e testado em Windows 10 Pro. (FALSO!!!)

Com o Sqoop podemos utilizar queries SQL para extrair dados de bancos de dados relacionais. O acesso ao banco de dados é feito via http e o comando Sqoop cria não um Job MapReduce. (FALSO!!!)

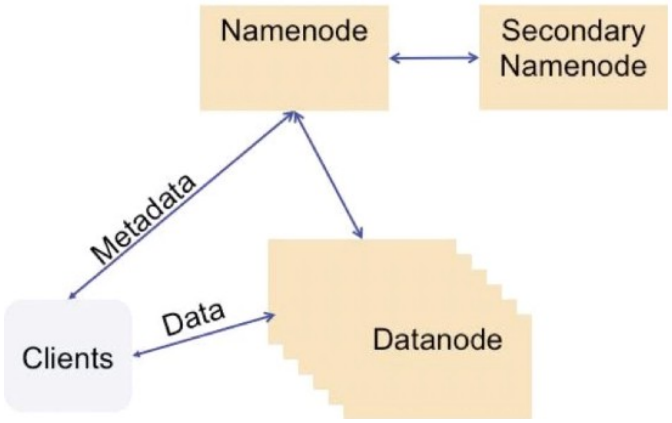
6.9. Administração e Manutenção do Hadoop

Desafios na Administração e Manutenção de um Cluster Hadoop

- Falta de gestão de configuração
- Baixa alocação de recursos
- Gargalos de rede
- Falta de métricas de monitoramento
- Medidas drásticas para resolver problemas simples
- Pontos únicos de falha
- Utilização dos valores default para os parâmetros
- Falta de profissionais qualificados

NameNode e Estrutura de Diretórios

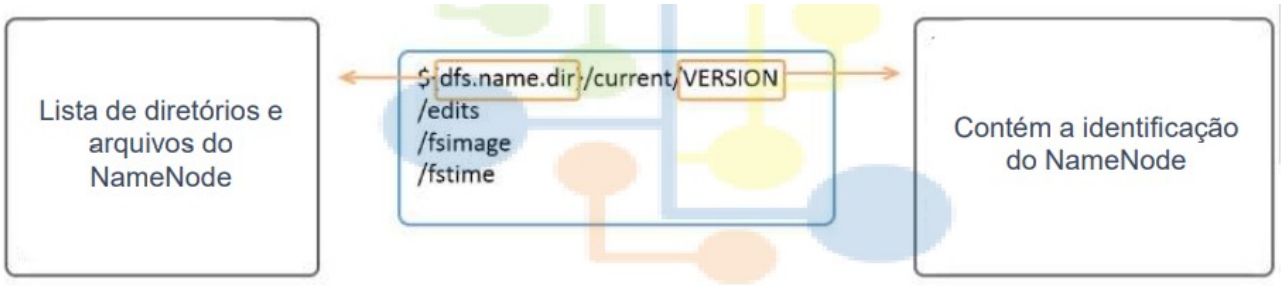
NameNode é o processo master do HDFS. Seu trabalho é o gerenciamento do cluster Hadoop. Ele não manipula os dados, mas os metadados (dados sobre os dados)



O endereçamento dos dados que se encontram em cada nó ao longo do cluster é mantido num arquivo do NameNode chamado fsimage. Esse arquivo é gravado em memória sempre que há inicialização do cluster.
Para uma melhor performance, as atualizações do cluster são gravadas em novos arquivos chamados de edit logs, para depois atualizar o fsimage.

Com o passar do tempo, o número de arquivos edit-log pode se tornar grande demais, sendo necessária uma atualização do fs-image. Essa é a função do SecondaryNameNode.

Estrutura de Diretórios do NameNode



Poucos arquivos grandes consomem menos memória.

A Importância do Secondary NameNode

O Secondary NameNode é o processo responsável por sincronizar os arquivos edit logs com a imagem do fsimage, para gerar um novo fsimage mais atualizado.

O principal objetivo do Secondary NameNode é ajudar a reconstruir o NameNode no caso desse vir a falhar!

DataNodes e Estrutura de Diretórios

Os DataNodes não precisam ser formatados (como o NameNode), uma vez que eles criam seus diretórios no storage automaticamente na inicialização.

A estrutura do DataNode é muito similar a do NameNode, com um arquivo VERSION com informações sobre o servidor e os arquivos binários de operação do serviço DataNode. Esse diretório é definido pelo parâmetro dfs.data.node.dir, nos arquivos de configuração do Hadoop.

Não se usa RAID de discos para fazer cópias de seguranças dos blocos.
O RAID é uma forma de se ter redundância de disco.

No Hadoop não há necessidade de redundância de disco, pois sua estrutura realiza 3 réplicas de cada bloco. Ou seja, um único bloco se encontra em 3 máquinas diferentes.

Tamanho padrão do bloco = 67108864 (64Mb) (versão anteriores)
Tamanho padrão do bloco = 134217728 (128Mb) (versão atuais)

Metadados do Sistema de Arquivos

Metadados → Informações gerais sobre o cluster e sobre os dados
Dados → Big Data

ATENÇÃO: Não tente modificar diretórios ou arquivos de metadados. Modificações podem causar interrupção do HDFS ou até mesmo a perda de dados de forma permanente. O Backup dos metadados é uma tarefa crítica em um cluster Hadoop.

fsimage x edit log

fsimage	Edit log
Representa uma imagem point-in-time dos metadados do file system	Contém uma série de arquivos, chamados segmentos
O arquivo é sequencial	Os segmentos representam todas as modificações feitas desde a data de criação do fsimage
Pode ser usado para obter o estado mais recente do file system quando o NameNode tiver problemas	Garante que nenhuma operação é perdida devido a uma falha do servidor

O fsimage não possui qualquer informação sobre os dados armazenados nos DataNodes.

NameNode

- VERSION
  - Layoutversion
  - namespaceID/clusterID/blockpoolIDstorageType
  - cTime
  - edits_start transaction ID-end transaction ID
  - edits_inprogress__start transaction ID
  - fsimage_end transaction ID
  - seen_txid
- in_use.lock

Parâmetro dfs.namenode.name.dir em hdfs-site.xml

DataNode

- BP-random integer-NameNode-IP address-creation time
- VERSION
  - storageType
  - blockpoolID
  - finalized/rbw
  - lazyPersist
  - dncp_block_verification.log
- in_use.lock

Lista de Parâmetros para configuração dos diretórios do NameNode e DataNode

dfs.namenode.name.dir dfs.namenode.edits.dir

dfs.namenode.checkpoint.period dfs.namenode.checkpoint.txns dfs.namenode.checkpoint.check.period dfs.namenode.num.checkpoints.retained dfs.namenode.num.extra.edits.retained dfs.namenode.edit.log.autoroll.multiplier.threshold dfs.namenode.edit.log.autoroll.check.interval.ms dfs.datanode.data.dir

Comandos de Configuração

Comando	**Descrição**
hdfs namenode	Inicializa o NameNode
hdfs dfsadmin -safemode enter hdfs dfsadmin -saveNamespace	Coloca o NameNode em modo de segurança e realiza um checkpoint
hdfs dfsadmin -rollEdits	Passa de um edit log para outro
hdfs dfsadmin -fetchImage	Obtém a última versão do fsimage (o que pode ser usado para criar um NameNode backup)

Procedimento de Checkpoint

Checkpoint é o processo pelo qual fsimage e edit log são combinados em um novo fsimage

Etapas do Checkpoint:

1. O NameNode Secundário solicita ao NameNode Master os arquivos
2. O NameNode Secundário solicita ao Master o fsimage e edit log (via HTTP GET)
3. O NameNode Secundário realiza operações no edit log e fsimage e gera um novo fsimage
4. Através de HTTP Post, o Secundário envia os arquivos de volta ao Master
5. O Master cria o novo fsimage e sobrepõe o antigo. O mesmo é feito com o edit log

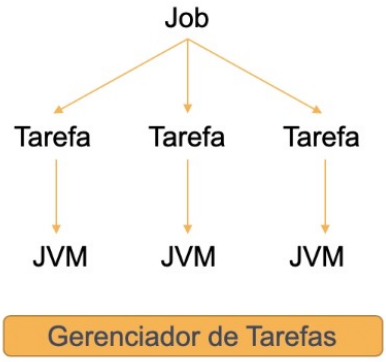
O processo de checkpoint é controlado por 2 parâmetros:

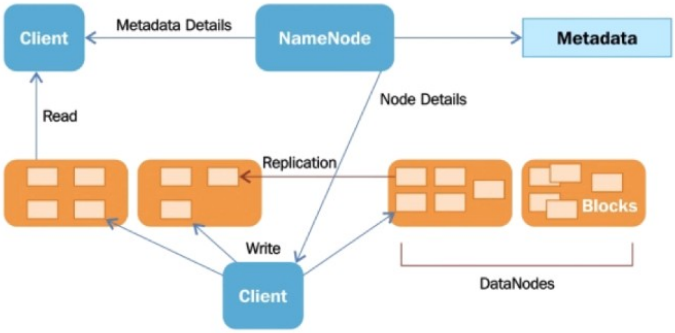
dfs.namenode.checkpoint.period	**dfs.namenode.checkpoint.txns**
O valor default é 1 hora, para garantir o próximo refresh entre 2 checkpoints consecutivos.	O valor default é 1 milhão e esta é a regra para definir o número de transações até o próximo checkpoint.

Compreender como funciona o checkpoint no HDFS pode fazer a diferença para ter um cluster eficiente.

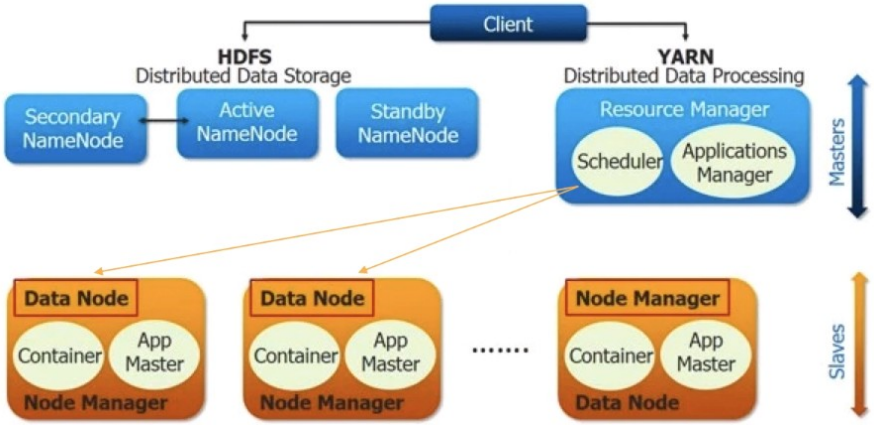
Procedimento de Recuperação à Falha do NameNode

O NameNode é o computador mais importante do cluster Hadoop.

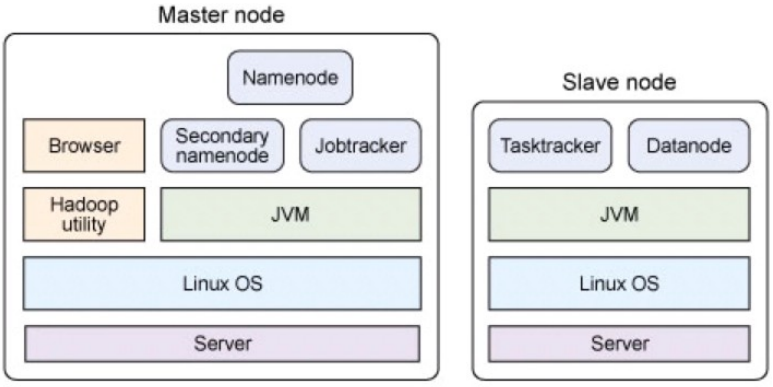




Processamento de Tarefas de Um Job MapReduce:



O sucesso e segurança do processo de análise de Big Data com Hadoop, depende do bom funcionamento do Node Master



No caso de não haver backup do NameNode e o servidor falhar, o risco de perda de dados pode ser reduzido, criando um nível de redundância do NameNode:

1. Faça uma cópia dos dados antes de promover o servidor a NameNode
2. Mude o endereço ip do novo servidor, para o endereço ip do servidor antigo
3. Garanta que o Hadoop esteja instalado e configurado de forma idêntica ao original
4. Não formate o NameNode

Modo de Segurança

Modo de Segurança (Safe Mode) é o modo apenas leitura do cluster HDFS, onde modificações não são permitidas no file system ou nos blocos.

Normalmente, o NameNode desativa modo de segurança automaticamente após concluir sua incialização. Se isso não ocorrer (por diversas razões), será necessário retirar do modo de segurança manualmente.

Em Safe Mode:

- Somente operações no file system, que acessam os metadados podem ser executadas
- Leitura de arquivos serão possíveis apenas se os blocos estiverem disponíveis nos DataNodes
- Modificações em arquivos não serão efetivadas

Para colocar o HDFS em Safe Mode, use o comando:

hdfs dsfadmin -safemode
hdfs dfsadmin -safemode leave (retira o HDFS de Safe Mode)

Backup

Backup é uma cópia de segurança dos dados.

No Hadoop o backup é composto de 2 partes:

• Backup dos metadados
• Backup dos dados

O HDFS é redundante a falhas por padrão, pois cada dado está dividido em 3 cópias espalhadas pelo cluster.

O Hadoop foi criado para executar em um único Datacenter

O Hadoop permite o armazenamento de dados compactados, tendo os formatos Parquet e ORC Files como os mais utilizados para isso.

Principais Estratégias de Backup do HDFS:

1. Soluções Proprietárias
    - Barracuda
2. Replicação do HDFS

    - Pode ser replicado em diferentes racks de computadores, em um mesmo datacenter.
3. Cópia dos Dados
    - Cópia dos dados de um cluster para outro
    - Pode ser feito com o utilitário Distcp
4. Dual Load
    - Gravação dos dados em 2 clusters simultaneamente

Apenas o Backup é suficiente?

Para garantir a segurança dos dados, além do backup, é necessário:

• Divisão dos dados em críticos e não críticos
• Formato do storage que será usado com o HDFS
• Monitoramento do Cluster
• Aplicação de Patches e Correções de Segurança

Uma vez feito o backup, tem-se o seu oposto: Recovery! O Recovery é retornar os dados do backup, caso necessário. Pode ser do tipo:

• Snapshots
• Replicação
• Recover Manual
• API

Outros comandos para identificar falhas no cluster:

Filesystem check	Filesystem Balancer	Relatório do cluster
hdfs fsck /	hdfs balancer	hdfs dfsadmin -report

A ferramenta usada para o backup de metadados e dados é o distcp (Distributed Copy).

Metadados dos demais produtos do ecossistema Hadoop, também devem ser incluídos no procedimento de Backup.

Solução de Problemas no Cluster Hadoop

O Hadoop não é um banco de dados e na condição de repositório de dados, ele possui muitas vantagens em termos de gerenciamento e segurança.

Hadoop é um grande ecossistema.

Possíveis problemas na Administração do Hadoop:

- Erro humano: Erros humanos são uma das principais causas de problemas com o Hadoop
- Hardware: Falhas de disco ocorrem com frequência e nem sempre de uma vez. A degradação ao longo do tempo, pode levar à lentidão, antes de uma falha ocorrer
- Erros de configuração: É difícil configurar um ambiente Hadoop 100% otimizado e são muitos parâmetros possíveis. A utilização destes parâmetros depende de uma série de fatores por isso recomenda-se a realização maciça de testes antes de aplicar alterações em produção

- Excessiva Utilização de Recursos: Erros em tarefas devem ser investigados e resolvidos. Erros recorrentes, consomem recursos do servidor, degradando performance
- Identificação dos nodes: Configuração de rede dos nodes é um item crítico, pois a comunicação entre eles poderá gerar problemas de performance

Solução de Problemas em um Ambiente Hadoop:

- Sempre audite seu ambiente checando possíveis problemas. Verifique o histórico de operação do sistema
- Siga os eventos que levam a falhas
- Verifique dependências. Por exemplo: MapReduce precisa do HDFS funcionando corretamente.
- Todos os deamons precisam de recursos para operar corretamente. Tenha certeza que há espaço em disco suficiente
- Busque por padrões de falhas
- Sempre verifique os logs

Autenticação e Segurança no Hadoop

Riscos de Segurança no Armazenamento de Big Data:
• Perda de Dados
• Queda de Produtividade
• Redução do Valor Geral dos Dados
• Vazamento de Dados Confidenciais

A segurança no Hadoop é feita através de múltiplas camadas:

<table>
<tr><td colspan="2"></td><td rowspan="2">Camada de Transferência de Dados

• Autorização Based Roles
• Transferência Segura
• Proxy Users</td></tr>
<tr><td></td><td>Camada do SO

• Autenticação do SO
• User e Group Policies
• Permissão de Arquivos</td></tr>
<tr><td>Camada Hadoop

• Encriptação
• Autenticação
• Autorização</td><td colspan="2"></td></tr>
</table>

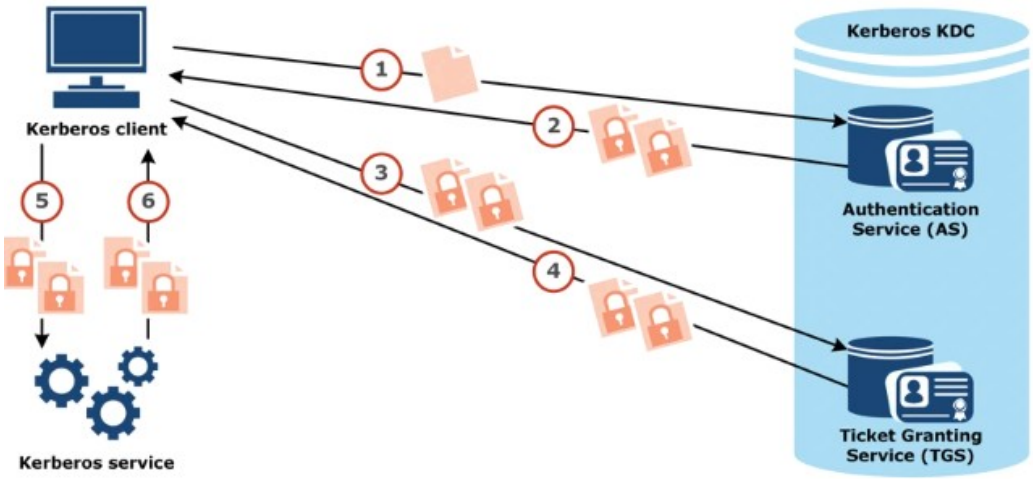
O Hadoop possui 2 métodos de Autenticação:

- Simples: O cliente precisa ter uma conta no sistema operacional Linux e utilizar o username para acesso ao HDFS e para executar Jobs MapReduce
- Kerberos: É utilizado um mecanismo de autenticação via token de acesso

Protocolo Kerberos

Kerberos é um protocolo de autenticação de redes, open-source e que vem sendo usado na autenticação do Hadoop desde a versão 0.20 do Framework.

Com o Kerberos, o usuário precisa obter um token de autenticação para poder acessar os dados no HDFS.



KDC – Centro de distribuição de chaves
Kerberos service – pode ser o HDFS, Microsoft, etc.

Configuração do Kerberos:

1. • Instalar os pacotes do Kerberos no sistema operacional
   • yum install krb5-server krb5-libs krb5-auth-dialog krb5-workstation
2. • Editar os arquivos de configuração
   • /etc/krb5.conf
   • /var/kerberos/krb5kdc/kdc.conf
3. • Copiar as versões atualizadas dos arquivos para cada node no cluster

Melhores Práticas de Monitoramento do Cluster Hadoop

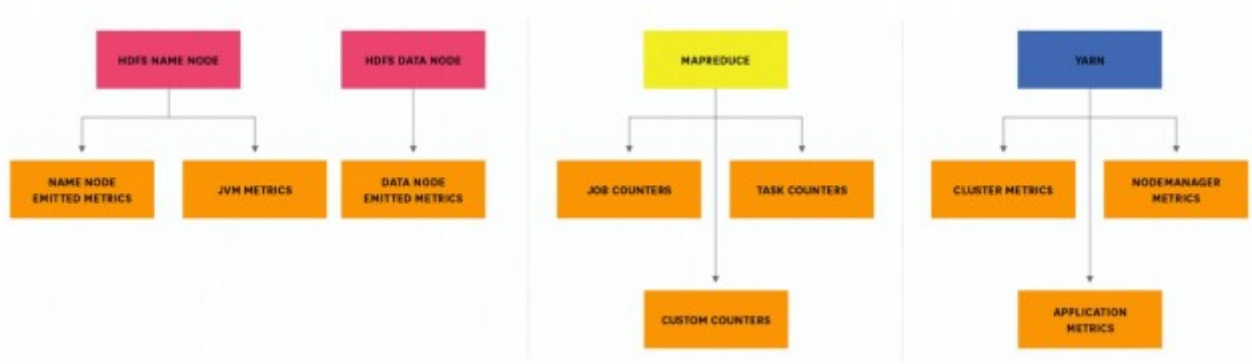
De uma perspectiva de operações, clusters Hadoop são incrivelmente resilientes em relação a falhas no sistema. O Hadoop foi projetado para ser tolerante a falhas e faz isso muito bem.

A falha de um DataNode pode não representar um problema, enquanto a falha no NameNode é algo crítico. Em outras palavras, indicadores de um único DataNode são menos importantes que os indicadores gerais de serviço de todo o cluster.

Quando configurado corretamente, um cluster Hadoop pode armazenar e processar quantidades massivas de dados, na casa de petabytes de dados. O monitoramento do cluster Hadoop normalmente é feito através do monitoramento de sub-componentes chave em 4 grandes categorias

Métricas de Monitoramento do Hadoop:

Cada um desses grupos se subdivide em grupos menores com diversas métricas que podem ser coletadas e monitoradas.



Os DataNodes se comunicam com o NameNode a cada 3 segundos, o que é configurável em um dos parâmetros nos arquivos de configuração. Essa comunicação, além de levar o estado do datanode, envia outras estatísticas que podem ser coletadas e usadas para monitorar o estado geral do cluster e do HDFS.

Diversas métricas podem ser coletadas e monitoradas (nesta tabela abaixo você encontra alguns exemplos de algumas das métricas que podem ser coletadas). Quando usamos o Apache Ambari, essas métricas são coletadas e apresentadas em um dashboards. Na seção de links úteis vc encontra a lista completa de todas as métricas de monitoramento do Hadoop.

Métricas HDFS

• NameNode metrics
• DataNode metrics

Métrica a ser coletada
CapacityRemaining CorruptBlocks / MissingBlocks VolumeFailuresTotal NumLiveDataNodes / NumDeadDataNodes FilesTotal TotalLoad BlockCapacity / BlocksTotal UnderReplicatedBlocks NumStaleDataNodes

Contadores MapReduce

• Job counters
• Task counters
• Custom counters
• File system counters

Métrica a ser coletada
MILLIS_MAPS/MILLIS_REDUCES NUM_FAILED_MAPS/NUM_FAILED_REDUCES REDUCE_INPUT_RECORDS SPILLED_RECORDS GC_TIME_MILLIS

NUM_FAILED_MAPS NUM_FAILED_REDUCES RACK_LOCAL_MAPS DATA_LOCAL_MAPS

Métricas YARN
• Cluster metrics
• Application metrics
• NodeManager metrics

Métrica a ser coletada
unhealthyNodes activeNodes lostNodes appsFailed totalMB / allocatedMB progress containersFailed

Métricas Zookeeper

Métrica a ser coletada
zk_followers zk_avg_latency zk_num_alive_connections

Como coletar as métricas?
• NameNode → coleta de métricas via API
• DataNode → coleta de métricas via API
• HDFS → coleta de métricas via JMX

Apache Ambari (Utilizado pelo: • Microsoft HDInsight • Hortonworks)
Cloudera Manager

Quizz

Solução de problemas em um ambiente Hadoop:
Busque por padrões de falhas.
Todos os deamons precisam de recursos para operar corretamente. Tenha certeza que há espaço em disco suficiente.
Verifique dependências. Por exemplo: MapReduce precisa do HDFS funcionando corretamente.

O NameNode é o processo master do HDFS. Todas as operações de manipulação de arquivos ou pastas do HDFS são tratadas pelo NameNode. Isso porque é ele quem contém as informações de quais nodes ao longo do cluster estão cada segmento de arquivo salvo no HDFS, bem como os diretórios existentes e suas permissões de acesso.

O Hadoop permite que se informe detalhes sobre a topologia da rede em que o cluster está configurado. Isso é muito útil para se obter melhor desempenho de execução de Jobs MapReduce. Essa funcionalidade é chamada de Rack Awareness.

Para colocar o HDFS em Safe Mode, use o comando:
hdfs dsfadmin -safemode

6.10. Hadoop Machine Learning com Apache Mahout

Conhecendo o Apache Mahout

O Apache Mahout é um conjunto de bibliotecas Java para Machine Learning, open source, que permitem realizar tarefas de classificação, clusterização, mineração de dados e busca por padrões, dentro do HDFS.

Principais Algoritmos de Machine Learning disponíveis no Apache Mahout:

• Algoritmos de Classificação
• Sistemas de Recomendação
• Clustering (aprendizagem não supervisionada)
• Redução de Dimensionalidade

Tipos de Algoritmos Suportados:

- Algoritmos Sequenciais (requerem que os dados estejam num único local de armazenamento)
  - Regressão Logística
  - Modelos Ocultos de Markov
  - Perceptrons de Multi-camadas
- Algoritmos Paralelos (podem ser utilizados em cluster de computadores)
  - Random Forest
  - Naive Bayes
  - K-Means (aprendizagem não supervisionada)

O Apache Mahout suporta tanto os algorítimos sequenciais como os paralelos. A diferença é, dependendo de como os dados estejam armazenados ou organizados, talvez não se tenha o benefício de usar computação paralela distribuida.

Apache Mahout x Outros Frameworks de Machine Learning

Então por que vamos estudar o Apache Mahout, se existem frameworks de Machine Learning mais fáceis e mais completos?

> Porque nenhum outro frammework de machine learning (R, scikit learn, TensorFlow, rapidminer, etc.) foi criado para manipular grandes conjuntos de dados.

Além disso, o mahout possui uma forte integração com o Spark, o que permite o processamento em paralelo de forma muito mais veloz.

Características do Apache Mahout:

- Os algoritmos do Mahout são escritos para funcionar sobre o Hadoop e dessa forma, eles funcionam em ambiente distribuído.
- O Framework Mahout está pronto para uso e permite realizar mineração de dados em grandes conjuntos de dados.
- O Mahout é eficiente na análise de grandes conjuntos de dados.

- Possui diversas implementações de Clustering, como: K-means, Fuzzy K-Means, Canopy e Mean-Shift.
- Suporta a execução do algoritmo de classificação Naive Bayes de forma distribuída
- Inclui bibliotecas para manipulação de vetores e matrizes.

Adobe, facebook, linkedin, tweeter, yahoo são exemplos de empresas que utilizam o Mahout.

Definindo Machine Learning

A palavra classificação faz parte das nossas vidas e do nosso dia a dia. Desde a época das aulas de português ou biologia ouvimos a palavra classificação o tempo todo. O conceito é simples: baseado em algumas características classificamos determinado objeto em uma categoria ou outra. Vejamos o conceito de Classificação nos algoritmos de Machine Learning.

Primeiro vamos definir Machine Learning. Machine Learning (ML) é uma área da Inteligência Artificial onde criamos algoritmos para ensinar a máquina a desempenhar determinadas tarefas. Um algoritmo de ML basicamente recebe um conjunto de dados de entrada e baseado nos padrões encontrados gera as saídas. Cada entrada desse conjunto de dados possui suas features (ou atributos) e ter um conjunto delas é o ponto inicial para qualquer algoritmo de ML. Feature é uma característica que descreve um objeto.

Qualquer atributo de um objeto pode ser tratado como feature, seja um número, um texto, uma data, um booleano etc. Como no objeto pessoa, temos vários atributos que o descreve, esses atributos são suas features. As features são as entradas dos algoritmos de ML, quanto mais detalhes o algoritmo tiver sobre uma entrada, mais facilmente achará padrões nos dados. Features ruins podem prejudicar o desempenho do algoritmo. Features boas são a chave para o sucesso de um algoritmo. Boa parte do trabalho em ML é conseguir trabalhar os dados e gerar boas features em cima deles, o que é conhecido como engenharia de features ou feature engineering. Existem diversas técnicas para gerar features, seja através do conhecimento da natureza dos dados ou da aplicação de matemática e estatística para criá-las sobre os dados. Tendo nossas features em mãos podemos aplicar diversos algoritmos de aprendizado nelas. Existem dois grandes grupos de algoritmos em ML, os de aprendizagem supervisionada e os de aprendizagem não supervisionada.

Aprendizagem Supervisionada

Quando você tem um conjunto de entradas que possuem as saídas que deseja prever em outros dados. Com conhecimento das entradas e saídas de um número suficiente de dados, os algoritmos desse grupo podem achar os padrões que relacionam as entradas com as saídas. Dessa forma, se tivermos novos dados apenas com as entradas, podemos prever as saídas com base nesses padrões previamente encontrados. São divididos em dois grupos: classificação e regressão.

Aprendizagem Não-Supervisionada

Quando você tem um conjunto de entradas sem as saídas que você deseja. Com base nas características desses dados podemos gerar um agrupamento ou processá-los a fim de gerar novas formas de expressar essas características. Dois grupos comuns de aprendizagem não supervisionada são: redução de dimensionalidade e clusterização.

Machine Learning – Algoritmos de Classificação

Vamos nos concentrar na Classificação, uma técnica de aprendizagem supervisionada e que compõe a maioria dos algoritmos suportados pelo Apache Mahout.

Classificação é a categorização de potenciais respostas e em ML queremos automatizar este processo. Temos classificação multiclasse, quando várias classes são possíveis no processo de classificação e temos a classificação binária, quando apenas duas classes são possíveis. Em Machine Learning, usamos um algoritmo que prevê, baseado em uma série de regras, a que classe um objeto ou indivíduo vai pertencer. Normalmente as classes são mutuamente exclusivas, ou seja, um indivíduo não pode pertencer a mais de uma classe ao mesmo tempo. O algoritmo de Machine Learning calcula a probabilidade de um indivíduo ou objeto pertencer a uma determinada classe.

Classificação é uma técnica de aprendizagem supervisionada. Nesta técnica, baseado em fatos históricos, o algoritmo é capaz de prever um valor desconhecido. Em aprendizagem supervisionada, nós já sabemos o possível output, ou seja, um objeto pode pertencer a uma ou outra categoria obrigatoriamente. Mas existem outras técnicas como a aprendizagem não supervisionada, em que não sabemos os possíveis outputs e nesse caso o algoritmo precisa realizar esta previsão.

Vejamos um simples exemplo: suponha que você tenha uma cesta de frutas e você quer classificá-las. Na sequência, faríamos as medidas de todas as frutas e colocaríamos em um banco de dados. Ao usar aprendizagem supervisionada, nós precisaríamos conhecer os possíveis outputs, por exemplo: uma maçã é vermelha e tem 4 centímetros de diâmetro.

Apresentamos todos os dados ao algoritmo e o treinamos. Ele então aprende sobre o que é uma maça e cada vez que encontrar algo parecido, ele classifica como maça.

Mas e se não tivéssemos os possíveis outuputs, ou seja, não sabemos o que é uma maça? Nesse caso, poderíamos usar aprendizagem não supervisionada e nosso algoritmo se encarrega de aprender sobre todas as características e fazer a classificação das frutas.

Outros exemplos de uso de algoritmos de classificação: Categorização de E-mails, Filtros de Spam, Detecção de Fraudes e Eligibilidade de Clientes

• Dataset de treino
• Dataset de teste
• Modelo

Processo de criação do modelo preditivo:

• Limpeza dos Dados
• Definição da variável target
• Definição das variáveis explanatórias
• Seleção do algoritmo
• Treinamento do modelo
• Teste do modelo
• Avaliação do modelo
• Otimização e ajuste do modelo
• Deploy

Outliers são valores extremos que não seguem o padrão esperado nos dados

Algoritmos suportados pelo Apache Mahout:

• Regressão Logística
• Naive Bayes
• Hidden Markov Model
• Random Forest
• Multi-Layer Perceptron (MLP)

Métricas de Ferramentas para Avaliar o Modelo de Classificação:

• Confusion Matrix
• Gráfico ROC
• AUC
• Entropy Matrix

Se o volume de dados for entre 1 e 10 milhões de registros, o Apache Mahout pode ser uma excelente opção

Modelo de Classificação com Naive Bayes

```
# Ciação do Modelo Preditivo com Naive Bayes – Arquivo NaiveBayes.sh

# Criar pastas no HDFS
hdfs dfs -mkdir /mahout
hdfs dfs -mkdir /mahout/input
hdfs dfs -mkdir /mahout/input/ham
hdfs dfs -mkdir /mahout/input/spam

# Copiando dados do filesystem local para o HDFS
hdfs dfs -copyFromLocal ham/* /mahout/input/ham
hdfs dfs -copyFromLocal spam/* /mahout/input/spam

# Converte os dados para uma sequence (obrigatório quando se trabalha com Mahout)
mahout seqdirectory -i /mahout/input -o /mahout/output/seqoutput

# Converte a sequence em vetores TF-IDF
mahout seq2sparse -i /mahout/output/seqoutput -o /mahout/output/sparseoutput

# Visualiza a saída
hdfs dfs -ls /mahout/output/sparseoutput

# Split dos dados em treino e teste
#               -i                              pasta com dados de entrada
#               --trainingOutput                dados de treino
#               --testOutput                    dados de teste
#               --randomSelectionPct percentual de divisão dos dados
#               --overwrite                     overwrite
#               --sequenceFiles                 input sequencial
#               --xm                            tipo de processamento.
mahout split -i /mahout/output/sparseoutput/tfidf-vectors --trainingOutput /mahout/nbTrain --testOutput /mahout/nbTest --randomSelectionPct 30 --
overwrite --sequenceFiles -xm sequencial

# Construção do Modelo Preditivo
#               -i              dados de treino
#               -li onde armazenar os labels
#               -o              onde armazenar o modelo
#               -ow             overwrite
#               -c              complementary
mahout trainnb -i /mahout/nbTrain -li /mahout/nbLabels -o /mahout/nbmodel -ow -c

# Teste do Modelo
#               -i  pasta com os dados de teste
#               -m              pasta do modelo
#               -l              labels
#               -ow             overwrite
#               -o              pasta com as previsões
```

```
#               -c              complementary
mahout testnb -i /mahout/nbTest -m /mahout/nbmodel -l /mahout/nbLabels -ow -o /mahout/nbpredictions -c
```

Clusterização com K-Means

```
# Criando um modelo preditivo de aprendizagem não-supervisionada – Arquivo Kmeans.sh

# Cria uma pasta no HDFS
hdfs dfs -mkdir /mahout/clustering
hdfs dfs -mkdir /mahout/clustering/data

# Copia os datasets para o HDFS
hdfs dfs -copyFromLocal news/* /mahout/clustering/data
hdfs dfs -cat /mahout/clustering/data/*

# Converte o dataset para objeto sequence
mahout seqdirectory -i /mahout/clustering/data -o /mahout/clustering/kmeansseq

# Converte a sequence para objetos TF-IDF vectors
mahout seq2sparse -i /mahout/clustering/kmeansseq -o /mahout/clustering/kmeanssparse

hdfs dfs -ls /mahout/clustering/kmeanssparse

# Construindo o modelo K-means
#               -i              diretório com arquivos de entrada
#               -c              diretório de destino para os centroids
#               -o              diretório de saída
#               -k              número de clusters
#               -ow             overwrite
#               -x              número de iterações
#               -dm             medida de distância
mahout kmeans -i /mahout/clustering/kmeanssparse/tfidf-vectors/ -c /mahout/clustering/kmeanscentroids -cl -o /mahout/clustering/kmeansclusters -
k 3 -ow -x 10 -dm org.apache.mahout.common.distance.CosineDistanceMeasure

# Visualiza os arquivos no HDFS
hdfs dfs -ls /mahout/clustering/kmeansclusters

# Dump dos clusters para um arquivo texto
mahout clusterdump -i /mahout/clustering/kmeansclusters/clusters-1-final -o clusterdump.txt -p /mahout/clustering/kmeansclusters/clusteredPoints/
-d /mahout/clustering/kmeanssparse/dictionary.file-0 -dt sequencefile -n 20 -b 100


# Visualiza os clusters.
cat clusterdump.txt
```

Quiz

O Apache Mahout é uma biblioteca machine learning, open source que tem como principais objetivos operar como uma máquina de recomendação, clustering e classificação.

Quando falamos de algoritmos de Machine Learning é importante destacar que nem todos serão capazes de executar sobre um ambiente de cluster, simplesmente porque não foram criados para processamento paralelo e distribuído.

O algoritmo Regressão Linear não é suportado pelo Apache Mahout.

Os algoritmos do Mahout são escritos para funcionar sobre o Hadoop e dessa forma, eles funcionam em ambiente distribuído.

O Apache Mahout Inclui bibliotecas para manipulação de vetores e matrizes.

6.11. Apache Hadoop e Apache Spark

Hadoop x Spark

- Hadoop:
  - Framework para desenvolvimento de aplicações distribuídas
  - HDFS Sistema de Arquivos Distribuído
  - MapReduce Processamento Distribuído
- Spark:
  - Framework para processamento de Big Data

Se Hadoop e Spark são produtos diferentes, com propósitos diferentes e operam de formas diferentes, por que são frequentemente comparados?

Comparamos o Apache Spark com o Apache MapReduce. Mas o Apache Spark não possui um sistema de armazenamento, podendo usar, por exemplo, o HDFS.

Embora alguns departamentos de TI podem se sentir compelidos a escolher entre Hadoop e Spark, o fato é que provavelmente, muitas empresas usarão os dois, por serem tecnologias complementares.

É possível usar um sem o outro!

Portanto, não existe escolha entre Hadoop e Spark e sim o objetivo do seu projeto. Os 2 frameworks são complementares e podem ser usados em conjunto.

Hadoop e Spark Juntos

- Hadoop e Spark fazem coisas diferentes
- Você pode usar um sem o outro
- O Spark é mais rápido
- Mas você pode não precisar da velocidade do Spark
- Mecanismos diferentes de recuperação a falhas

Spark é considerado o futuro do processamento distribuído no ecossistema Hadoop!

Com o Spark podemos realizar:

• Operações de ETL
• Análise Preditiva e Machine Learning
• Operações de Acesso a Dados com SQL
• Text Mining
• Processamento de Eventos em Tempo Real
• Aplicações Gráficas
• Reconhecimento de Padrões
• Sistemas de Recomendação

Embora seja escrito em Scala, o Spark suporta Scala, Java, python, R e SQL.

O Spark é normalmente utilizado com o HDFS, mas outros sistemas de arquivos ou sistemas de armazenamento podem ser usados, tais como:

• Sistema de arquivos local ou de rede (NFS)
• Amazon S3
• RDBMS
• NoSQL (Apache Cassandra, Hbase)
• Sistemas de Mensagens (Kafka)

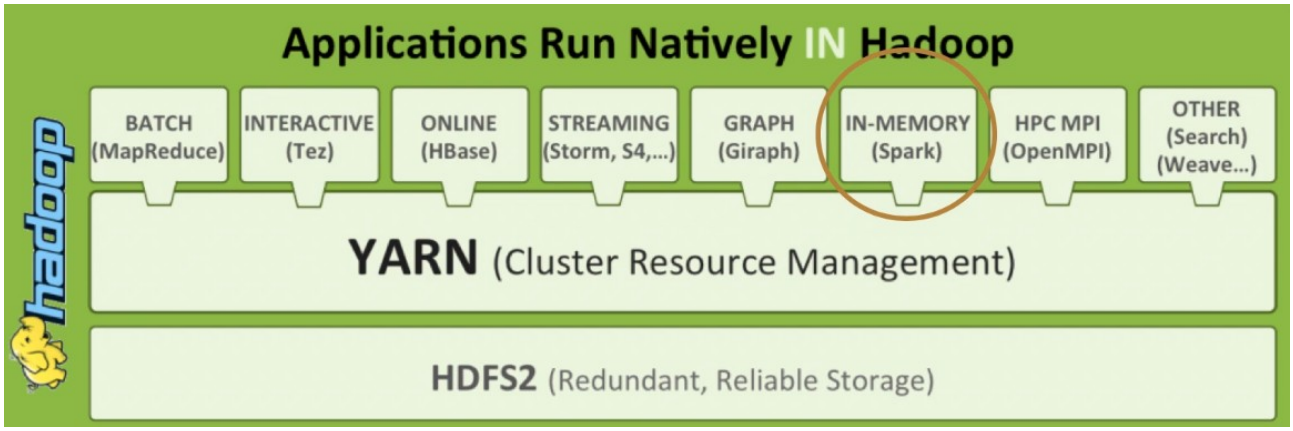
Como o Spark Funciona Sobre o HDFS?

Para ler arquivos do HDFS com Spark usamos:
textfile = sc.textFile("hdfs://mycluster/data/file.txt")

Para gravar arquivos no HDFS com Spark usamos:
myRDD.saveAsTextFile("hdfs://mycluster/data/output.txt")

Modos de Instalação do Spark

- Standalone
  - Spark é executado sem um agendador de tarefas externo
- YARN (Hadoop)
  - Faz o gerenciamento de recursos do cluster.
  - Pode ser usado como agendador de tarefas pelo Spark
- Mesos



Anatomia de Uma Aplicação Spark

Se eu posso construir meu processo de análise com Python ou R, por exemplo, por que usaria o Spark?

1 – Porque você precisa processar um grande volume de dados.
2 – Porque você quer usar uma das APIs prontas do Spark, como SQL ou Streaming, por exemplo!

As 4 APIs ou módulos principais, que compõem o ecossistema Spark:

- Spark SQL – permite aplicar linguagem SQL a grandes conjuntos de dados, de maneira distribuída em cluster.
- Spark Streaming – permite processar dados gerados em tempo real.
- MLlib (machine learning) – permite aplicar algorítimos de Machine Learning de maneira distribuída.
- GraphX (graph) – permite trabalhar com grafos computacionais, para computação paralela.

Temos ainda alguns módulos adicionais:

- Cassandra Spark Connector – conecta ao banco de dados Cassandra (noSql)
- SparkR – permite executar aplicações em R, diretamente com Spark

Arquitetura do Spark

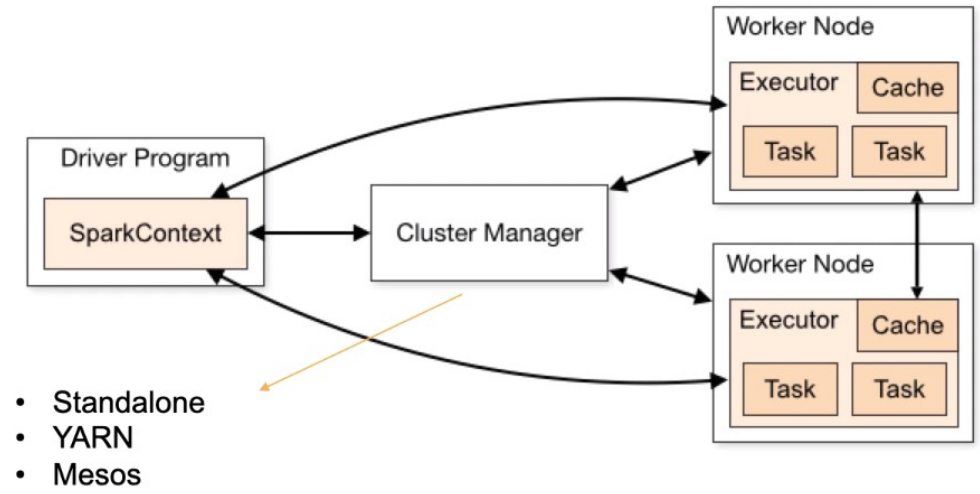
Basicamente, qualquer aplicação desenvolvida com Spark envolverá 3 componentes principais:

- API – API (Scala, Java, Python)
- Gestão de Recursos - Distributed Computing (Stand-alone, Mesos, YARN)
- Dados – Storage (HDFS, Other Formats)

Processamento de Uma Aplicação Spark

O cliente submete a aplicação Spark

A aplicação Spark é planejada, agendada, executada e monitorada



Outras Características do Spark

Outras Características do Spark:

• Suporta mais do que apenas as funções de Map e Reduce
• Otimiza o uso de operadores de grafos arbitrários
• Avaliação sob demanda de consultas de Big Data contribui com otimização do fluxo global do processamento de dados

• Fornece APIs concisas e consistentes em Scala, Java e Python
• Oferece shell interativo para Scala, Python e R. O shell ainda não está disponível em Java

RDDs e Dataframes

O Apache Spark oferece 2 estruturas de dados principais:

1. RDDs (Resilient Distributed Datasets)

   Dados não estruturados (oferece mais flexibilidade)

2. Dataframes (estruturas de dados mais recentes lançadas no apache 2.0)

   Dados estruturados ou semiestruturados

RDD é uma coleção de objetos distribuída e imutável. Cada conjunto de dados no RDD é dividido em partições lógicas, que podem ser computadas em diferentes nodes do cluster.

RDD é Conceito Central do Framework Spark!

RDD's são imutáveis!

Existem 2 formas de criar um RDD:

- Paralelizando uma coleção existente (função sc.parallelize)
- Referenciando um dataset externo (HDFS, RDBMS, NoSQL, S3)

No Hadoop MapReduce (que não possui o conceito de RDD), cada resultado intermediário é gravado em disco. Ou seja, imagine um algoritmo de ML que precisa realizar diversas iterações nos dados. O disco será usado com muita frequência, deixando o processo mais lento.

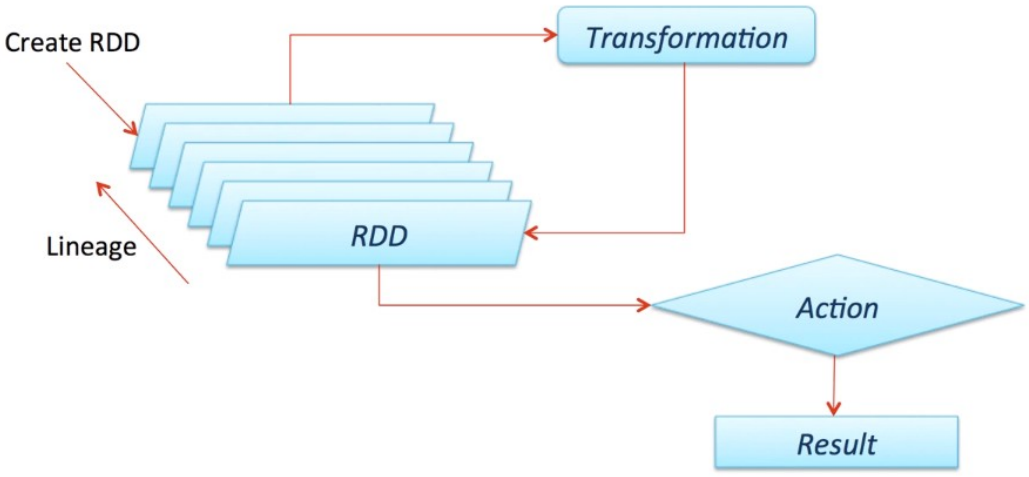
Com o conceito de RDD, o Spark armazena os resultados intermediários em memória, permitindo que operações iterativas que precisam acessar os dados diversas vezes, possam recorrer a memória do computador e não ao disco. Os dados serão gravados em disco apenas ao fim do processo ou se durante o processo, não houver memória disponível. Lembre que estamos falando aqui de cluster de computadores, com Terabytes de memória RAM quando se combina a memória de cada node do cluster.

Os RDD's podem ser particionados e persistidos em memória ou disco!

O RDD suporta dois tipos de operações:

Transformações (cria um novo RDD)	Ações (conclui uma série de transformações)
map() filter() flatMap() reduceByKey() aggregateByKey	reduce() collect() first() take() countByKey()

Lazy Evaluation:



Funções cache() X persist() e o fluxo de operações do Spark:

1. Criação do RDD: sc.parallelize() sc.textfile()
2. Transformação do RDD: Map, Flatmap, distinct, filter
3. Persistência em Memória: Cache Persist
4. Ações sobre o RDD: Resultado

Quando usamos RDDs?

- Você deseja transformações e ações de baixo nível e controle no seu conjunto de dados. Seus dados não são estruturados, como fluxos de mídia ou de texto;
- Você deseja manipular seus dados com construções de programação funcional;
- Você não se preocupa em impor um esquema, como formato colunar, ao processar ou acessar atributos de dados por nome ou coluna;
- Você pode renunciar a alguns benefícios de otimização e desempenho disponíveis com Dataframes e Datasets para dados estruturados e semiestruturados.

Dataframes

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

- Como um RDD, um Dataframe é uma coleção distribuída imutável de dados.
- Ao contrário de um RDD, os dados são organizados em colunas nomeadas, como uma tabela em um banco de dados relacional.
- Projetado para facilitar ainda mais o processamento de grandes conjuntos de dados, o Dataframe permite que os desenvolvedores imponham uma estrutura em uma coleção distribuída de dados, permitindo abstração de nível superior.

Quando usamos Dataframes?

- Se você deseja semântica rica, abstrações de alto nível e APIs específicas do domínio, use Dataframe ou Dataset.
- Se o seu processamento exigir expressões de alto nível, filtros, mapas, agregação, médias, soma, consultas SQL, acesso colunar e uso de funções lambda em dados semiestruturados, use Dataframe ou Dataset.
- Se você deseja unificação e simplificação de APIs nas bibliotecas Spark, use Dataframe ou Dataset.
- Se você é um usuário R, use Dataframes.
- Se você é um usuário Python, use Dataframes e recorra aos RDDs se precisar de mais controle.

Em resumo, a escolha de quando usar RDD ou Dataframe e/ou Dataset parece óbvia. Enquanto o primeiro oferece funcionalidade e controle de baixo nível, o último permite visualização e estrutura personalizadas, oferece operações específicas de alto nível e domínio, economiza espaço e é executado em velocidades superiores.

Apache Spark SQL

O Spark SQL, é parte integrante do framework Apache Spark, utilizado para processamento de dados estruturados, que permite executar consultas SQL no conjunto de dados do Spark. É possível realizar tarefas de ETL sobre os dados em diferentes formatos, como por exemplo JSON, arquivos csv, bancos relacionais e não relacionais. O Spark SQL permite que facilmente usemos SQL em nossas aplicações de análise de dados escritas em Python para Spark, por exemplo. Vejamos as principais funcionalidades do Spark SQL.

As versões mais recentes do Spark fornecem uma abstração de programação chamada de DataFrames, que pode agir como um motor de consultas SQL. DataFrames podem fornecer funções de alto nível, permitindo que o Spark compreenda melhor a estrutura dos dados, assim como o cálculo a ser executado. Esta informação adicional permite que o otimizador e o mecanismo de execução do Spark acelerem automaticamente as análises de Big Data.

Com a inclusão de uma API para fontes de dados, a biblioteca Spark SQL permite que a computação de informações sobre dados armazenados de forma estruturada seja facilitada e mais abrangente.

Com o servidor JDBC interno do Spark, a conexão com banco de dados relacionais, para consultar dados estruturados em tabelas e realizar análises de Big Data, pode ser feita com ferramentas de BI tradicionais.

O Spark SQL se integra com o Spark MLLib, o módulo de Machine Learning do Spark, servindo como fonte de dados para a construção de modelos preditivos. Ou seja, você pode usar o Spark SQL para extrair dados de Data Warehouses por exemplo, realizar alguns tratamentos e entregá-los para processamento de Machine Learning.

Componentes do Apache Spark SQL

Uma das principais vantagens do Spark SQL é que você não precisa reaprender nada. Podemos usar os mesmos conceitos de SQL que usamos em bancos de dados relacionais, extrair dados para o Spark e então nos beneficiarmos do processamento paralelo e distribuído fornecido pelo framework Spark através de clusters de computadores. O Spark SQL é uma biblioteca poderosa que Cientistas e Analistas de Dados, podem utilizar para realizar análise de dados em suas empresas ou clientes. Vejamos os componentes do Spark SQL.

DataFrame

Um DataFrame é uma coleção de dados distribuídos e organizados em forma de colunas nomeadas. É baseado no conceito de estrutura de dados da linguagem R, dataframes do Pandas e similar a uma tabela de um banco de dados relacional.

Nas primeiras versões, o componente DataFrame era chamado de SchemaRDD. DataFrames podem ser transformados em RDDs. DataFrames podem ser criados a partir de diferentes fontes de dados e embora tenham métodos diferentes para manipulação, internamente o Spark trata um DataFrame como um RDD. Mas usar DataFrames facilita o trabalho de quem está construindo as aplicações de análise de dados.

DataFrames suportam operações: filter, join, groupby, agg e aninhamento de operações.

Spark Session

Criamos um Spark Session para acessar as funcionalidades do Spark SQL. Dataframes são criados a partir de Spark Sessions, que permitem registrar um Dataframes como tabelas temporárias e executar queries SQL (muito útil no processamento de streams de dados).

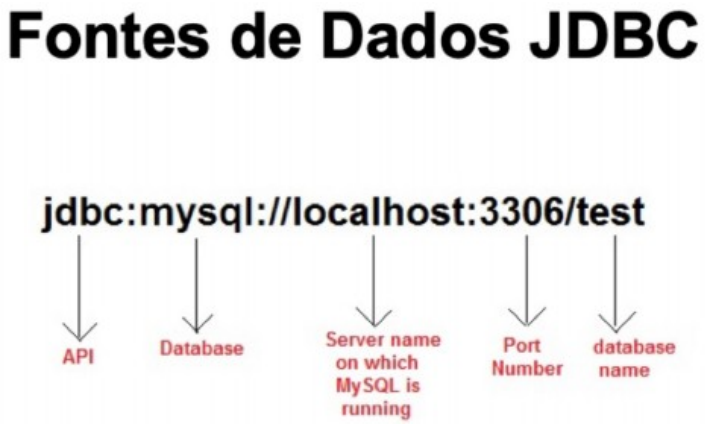
SQL Context

O Spark SQL fornece o componente SQLContext para encapsular todas as funcionalidades relacionais no Spark. É possível criar os SQLContext a partir do Spark Context. Existe também um componente HiveContext o qual fornece um conjunto maior de funcionalidades para o SQLContext. Este componente pode ser utilizado para escrever consultas utilizando o HiveQL e com isto, ler dados de tabelas Hive, a partir do Spark.

Fontes de Dados JDBC

Outras funcionalidades na biblioteca Spark SQL incluem fontes de dados que fazem uso de JDBC (Java Database Connection) como interface de integração. Interfaces de integração JDBC podem ser utilizadas para ler informações armazenadas em banco de dados relacionais. O JDBC é um conjunto de classes e interfaces (API) escritas em Java para execução e manipulação de resultados de instruções SQL para qualquer banco de dados relacional. Para cada Banco de dados há um driver JDBC. Apenas para que você tenha ideia da importância de se utilizar JDBC como conexão a bancos de dados, todos os principais bancos comerciais, como Oracle, SQL Server, DB2 e MySQL suportam conexão JDBC.

Para realizar uma conexão via JDBC, precisamos definir uma string de conexão, onde especificamos a API, o banco de dados, o nome do servidor, a porta e o nome do banco de dados que iremos conectar. Abaixo um exemplo de como seria essa string:



Tabelas Temporárias As tabelas temporárias são outro importante recurso do Spark SQL. Podemos usar operações SQL em tabelas temporárias e embora sejam estruturas simples, são muito poderosas. Uma query executada em uma tabela temporária retorna um outro Dataframe.

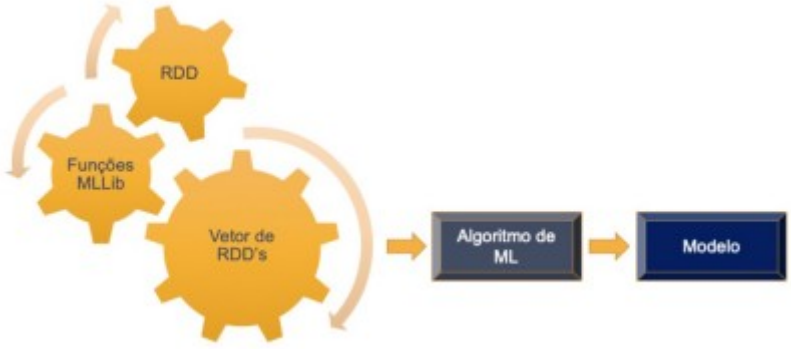
Apache Spark MLLib

O objetivo dos algoritmos de Machine Learning é tentar fazer previsões sobre conjuntos de dados, frequentemente otimizando uma função matemática que melhor descreve o relacionamento entre os dados. Existem diversos tipos de problemas que podem ser resolvidos com Machine Learning, tais como classificação, regressão e clusterização, cada qual com diferentes técnicas. O Spark MLLib é a biblioteca do Spark responsável pelo aprendizado de máquina. Vejamos o que é o MLLib.

O módulo MLLib do Spark contém as funções que implementam Machine Learning e foi criado para ser executado em paralelo, através de um cluster, assim como os demais módulos do Spark. Os algoritmos de Machine Learning do MLLib podem ser executados por todas linguagens de programação suportadas pelo Spark, bem como este módulo é intercambiável com os outros módulos do Framework Spark. O conceito por trás do MLLib é simples. Podemos invocar os algoritmos de Machine Learning e aplicar os modelos nos RDD's. O MLLib introduz mais alguns tipos de dados como vetores e label points, que são funções que aplicamos ao conjunto de dados.

Como já vimos, os dados no Spark são representados por RDD's, que são objetos que armazenam o conjunto de dados e podem ser particionados e distribuídos em paralelo através de um cluster. Aplicamos as funções do MLLib aos RDD's, como por exemplo, funções para feature extraction, que nos permitem converter valores de strings por representações numéricas (mais apropriadas para os algoritmos de Machine Learning). Ao aplicar uma função ao RDD, temos como retorno um vetor

de RDD’s. Nós então aplicamos os algoritmos de Machine Learning a esses vetores e criamos nosso modelo preditivo. O MLLib possui ainda funções de avaliação do modelo preditivo.



Principais Funcionalidades do Apache Spark MLLib

O MLLib possui também diversas funções para o trabalho de pré-processamento dos dados. Muitas dessas funções são encontradas no pacote mllib.feature.

Funcionalidades (Feature Extraction)	Funções (importadas a partir do mllib.feature)
TF-IDF (Team Frequency – Inverse Document Frequency)	HasingTF() e IDF()
Escala	StandardScaler()
Normalização	Normalizer()
Word2Vec	Word2Vec()
Estatística	ColStats(), corr(), chiSqTest(), mean(), stdev(), sample()

Funções TF-IDF são usadas para tratamento de textos e geram vetores a partir de documentos de texto (como páginas web por exemplo). São computadas 2 estatísticas em cada termo em cada documento. O TF (Term Frequency) é o número de vezes que um termo ocorre no documento e o IDF (Inverse Document Frequency) é o número de vezes não frequentes (por isso o nome inverso) que um termo aparece em um documento, normalmente representado pelo Corpus (conjunto de dados de texto). Se multiplicarmos esses 2 índices (TF x IDF) teremos quão relevante é um termo em um documento específico. Normalmente processamos um texto com o pacote NLTK da linguagem Python antes de aplicarmos essas funções com MLLib. O NLTK é bem mais completo para atividades de mineração de texto, enquanto o MLLib é indicado para processamento paralelo e distribuído em cluster, por exemplo para tarefas de Processamento de Linguagem Natural ou Sistemas de Recomendação.

Muitos algoritmos de Machine Learning precisam que os dados estejam na mesma magnitude, ou seja, na mesma escala. Os algoritmos esperam por isso e vão funcionar corretamente se você, Cientista de Dados, entregar isso a eles. Um exemplo de escala é colocar os dados com a mesma média e desvio padrão igual a 1. Normalização dos dados é outra tarefa importante durante o pré-processamento e em MLLib, usamos a função Normalizer().

Word2Vec é um algoritmo baseado em redes neurais para textos, que pode ser usado com diversos outros algoritmos de análise. Similar ao TF/IDF.

Podemos também executar diversas operações e testes estatísticos com MLLib, com a vantagem de poder realizar o processamento em grandes conjuntos de dados.

Algoritmos de Machine Learning Suportados Pelo Apache Spark MLLib

O MLLib não suporta todos os algoritmos de Machine Learning. Alguns algoritmos não foram construídos para execução em paralelo e de forma distribuída e por isso não foram implementados no MLLib. Porém muitos outros algoritmos podem ser usados, sejam algoritmos de aprendizagem supervisionada ou aprendizagem não supervisionada.

Vejamos a descrição dos principais algoritmos de Machine Learning suportados pelo MLLib.

A classificação é uma família de algoritmos de aprendizagem supervisionada que designam valores de entrada como pertencendo a uma de várias classes pré-definidas. Alguns casos de uso comum de classificação incluem: detecção de fraude com cartão de crédito e detecção de spam. Os dados de classificação são rotulados, por exemplo, como spam / nãospam ou fraude / não-fraude. O algoritmo de Machine Learning atribui um rótulo ou classe para novos dados, classificando esses novos dados com base em características pré-determinadas.

As árvores de decisão criam um modelo que prevê a classe ou o rótulo com base em várias características de entrada. As árvores de decisão funcionam avaliando uma expressão contendo uma característica em cada nó e selecionando uma alternativa para o próximo nó com base na resposta.

No clustering, o algoritmo cria grupos de objetos em categorias, analisando semelhanças entre exemplos de entrada. Esses grupos são chamados de clusters (não confundir com cluster de computador, ok? O termo é igual, mas são coisas diferentes). Utilizações de clustering incluem: resultados da pesquisa de agrupamento, agrupamento de clientes, detecção de anomalias e categorização de texto. Clustering utiliza algoritmos não supervisionados, ou seja, não são expostos as possíveis saída.

Algoritmos de filtragem colaborativa recomendam itens (esta é a parte do filtro) com base na preferência de informações de muitos usuários (esta é a parte colaborativa). A abordagem de filtragem colaborativa baseia-se na semelhança; as pessoas que gostavam de itens semelhantes no passado vão gostar itens semelhantes no futuro. O objetivo de um algoritmo de filtragem colaborativa é tomar preferências de usuários, e criar um modelo que pode ser usado para recomendações ou previsões.

Uma das características importantes sobre MLLib é que ele contém apenas algoritmos de Machine Learning que podem ser executados em paralelo através de um cluster. Alguns algoritmos clássicos não estão incluídos no MLLib, pois não foram criados para processamento em paralelo. Em compensação, o MLLib permite criar modelos com alguns importantes algoritmos como Árvores de Decisão e K-Means. Os algoritmos do MLLib são otimizados para computação em clusters e com grandes conjuntos de dados. Se o seu objetivo for aplicar Machine Learning em conjuntos de dados pequenos ou médios, com certeza o MLLib não será a melhor opção para isso e nesse caso podemos usar o Scikit-Learn ou mesmo o TensorFlow.

O MLLib oferece duas bibliotecas para a construção dos modelos:

spark.mllib → API original construída para trabalhar com RDD's
spark.ml → Nova API construída para funcionar também com Dataframes e SparkSQL

Apache Spark MLLib x Apache Mahout

Apache Mahout é o framework de Machine Learning do Hadoop e MLLib é o framework de Machine Learning do Spark. Eles são concorrentes ou complementares? Qual é o melhor e quando cada um deve ser utilizado? É o que vamos responder agora.

A Fundação Apache tem introduzido muitos frameworks de Machine Learning e um dos mais utilizados em ambientes de larga escala é o Apache Mahout. Já existem muitas empresas usando o Mahout para criar sistemas de recomendação ou construir modelos preditivos sobre grandes conjuntos de dados. A Amazon talvez seja um dos exemplos mais emblemáticos. A empresa utiliza o Mahout em seus sistemas de recomendação e segundo a empresa obteve um crescimento de 35% nas vendas desde a implementação do framework.

O Mahout demonstrou ter algumas funcionalidades que o Spark MLLib ainda se quer implementou. Mas o Mahout tem um pequeno problema. Ele é executado sobre o Hadoop MapReduce, o que restringe e muito sua performance. Algoritmos de Machine Learning geralmente utilizam muitas iterações, o que pode tornar o Mahout lento. Em contraste, o MLLib foi construído sobre o Spark, que é muito mais veloz que o Hadoop MapReduce.

A principal diferença entre Mahout e MLLib recai sobre os frameworks em que eles são executados, Hadoop MapReduce ou Apache Spark. A questão é que o Spark é muito mais veloz e por conta disso, o MLLib processa os algoritmos de Machine Learning de forma muito mais rápida. Entretanto, o MLLib não implementa alguns dos algoritmos implementados no Mahout. As atualizações do Mahout também são menos frequentes que as atualizações do MLLib. A equipe do Mahout está desenvolvendo uma nova biblioteca para álgebra linear chamada Samsara, que promete mudar completamente o funcionamento do Mahout. Os dois frameworks suportam processamento paralelo e distribuído, suportam linguagem Java e Python, permitem processar grandes conjuntos de dados e são open-source.

Algumas pesquisas recentes indicaram que em um único job MapReduce, o Spark é significantemente mais veloz que o Mahout.

Ok, então qual framework de Machine Learning devo utilizar?

Se você estiver começando em Machine Learning para computação em larga escala, o Spark MLLib seria uma escolha mais segura. O Apache Mahout deve ser usado apenas em casos bem específicos com volumes de dados na casa de Petabytes e para utilizar algoritmos que não estejam implementados no MLLib. Já existem esforços para migrar o Apache Mahout para ser executado sobre o Spark, o que vai torná-lo muito parecido ao MLLib. Portanto, se tiver que criar modelos preditivos para grandes conjuntos de dados, sua melhor escolha seria o Spark MLLib.

Mas saiba que já existe um novo framework que promete superar a velocidade do MLLib e executar algoritmos de Machine Learning bem mais poderosos e voltados para Inteligência Artificial. É o H2O e esse é o site, h2o.ai. Mas isso é assunto para outro curso.

Apache Spark Streaming

Muito do processamento realizado em procedimentos analíticos são feitos em dados devidamente armazenados e disponíveis, seja em bancos de dados ou arquivos csv. Carregamos os dados, fazemos alguns filtros e aplicamos técnicas de análise de dados. Esse procedimento funciona bem para resolver alguns problemas de negócio. Mas e quando precisamos realizar análises que não podem esperar todo o processo de carga, filtro e manipulação dos dados? Por exemplo: detecção de fraudes com cartões de crédito. Faz sentido aguardar 5 horas, para processar dados e então obter uma visão sobre o que está acontecendo? Precisamos detectar a fraude no momento em que ela ocorre e para isso precisamos analisar dados em tempo real. Esta é a proposta do Spark Streaming.

A vida não acontece em batches (lotes), sendo na verdade um fluxo contínuo de acontecimentos. Muitos dos sistemas que desejamos monitorar e entender, acontecem como um fluxo contínuo de eventos - batimentos cardíacos, correntes oceânicas, métricas de máquinas, ou sinais de GPS. A lista, assim como os eventos, é essencialmente infinita. É natural, então, que possamos recolher e analisar informações a partir desses eventos, como um fluxo de dados. Mesmo análise de eventos esporádicos, como o tráfego de web sites pode se beneficiar de uma abordagem de streaming de dados.

Com Spark podemos manipular os dados de acordo com a forma como eles são gerados. Se tivermos um grande volume de dados por exemplo de transações comerciais de uma rede de varejo ao longo de um ano, podemos processar isso em batch. Carregamos os dados uma única vez, processamos e analisamos os dados. Mas podemos também, coletar dados à medida que eles são gerados, processar e analisar. O Spark suporta as duas abordagens.

Há muitas vantagens potenciais de manipulação de dados como fluxos, mas até recentemente esse era um trabalho difícil. Atualmente Streaming de dados e análises em tempo real estão se tornando cada vez mais o padrão do mercado. E por que existe agora uma explosão de interesse em streaming de dados? A resposta a essa pergunta é que as novas tecnologias agora estão disponíveis para lidar com streaming de dados em níveis de alto desempenho, grande escala e de forma muito fácil - o que está levando mais empresas a lidar com dados como um stream.

O Apache Spark Streaming é um sistema de processamento de streaming, tolerante a falhas e escalável, o que significa que rapidamente podemos aumentar a quantidade de nodes em um cluster e assim expandir sua capacidade.

Por ser parte do framework, o Spark Streaming se integra com o MLLib, o Spark SQL e o Graphx. A partir da versão 2.0, o Spark Streaming suporta streaming estruturado de dados com o Streaming DataFrame. O Spark Streaming pode receber dados de diversas fontes e produzir resultados que podem ser usados por diversas soluções do mercado.

O Spark Streaming funciona com o conceito de microbatching para simular análise de fluxo em tempo real. Isso significa que um programa em batch é executado em intervalos frequentes para processar todos os dados ao longo do streaming. Embora esta abordagem seja inadequada para aplicações de baixa latência (aquelas que realmente requerem "real, realtime"), é uma maneira inteligente de utilizar pequenos processos em lote (microbatching) para se aproximar de uma atividade em tempo real e funciona bem para muitas situações.

Quer dizer que o Spark Streaming não é ”real” real-time?

Sim, isso mesmo. Na prática, o que esse módulo faz é gerar micro-batches a partir do fluxo de dados capturados e com isso simular um processamento em tempo real. Não deixa de ser genial, mas na prática, o Spark Streaming não é o que chamamos de real, real-time. Mas como os micro-batches são velozes e processados em memória, temos a impressão de estarmos trabalhando com dados em tempo real. Isso será muito bom em algumas situações, mas não será em outras. Lembre-se: não existe tecnologia perfeita.

O Spark Streaming pode receber dados de diversas fontes e para cada uma dessas fontes haverá um receiver. Aqui as fontes de dados suportadas pelo Spark Streaming:
• Flat Files (à medida que são criados)
• TCP/IP
• Apache Flume
• Apache Kafka
• Amazon Kinesis
• Mídias Sociais (Facebook, Twitter, etc.)
• Bancos NoSQL • Bancos Relacionais

Uma importante vantagem de usar o Spark para Big Data Analytics é a possibilidade de combinar processamento em batch e processamento de streaming em um único sistema.

Streaming de dados é considerado uma das tecnologias mais promissoras e o Apache Streaming, além de ser fácil de utilizar, é totalmente integrado as demais bibliotecas do Spark, permitindo a criação de soluções bem robustas.

Com o Apache Streaming podemos: coletar e analisar dados direto da fonte e à medida que são gerados, transformar e sumarizar os dados, aplicar modelos de Machine Learning e fazer previsões em tempo real.

Processamento em Batch x Processamento de Stream

Existem basicamente 2 modos de processamento de dados:

- Batch – você inicia o processamento de um arquivo ou dataset finito, o Spark processa as tarefas configuradas e conclui o trabalho.
- Streaming – você processa um stream de dados contínuo; a execução não pára até que haja algum erro ou você termine a aplicação manualmente.

Usamos Processamento em Batch no Apache Spark para:
• Análise exploratória de dados
• Gerar Data Warehouses sobre grandes conjuntos de dados, estilo OLAP
• Treinar um modelo de aprendizado de máquina sobre grandes conjuntos de dados
• Outras tarefas analíticas que antes eram feitas com Hadoop MapReduce

Usamos Processamento de Stream no Apache Spark para:
• Monitoramento de serviços
• Processamento de eventos em tempo real para alimentação de dashboards
• Processamento dados de cliques e eventos em web sites
• Processamento de dados de sensores de Internet das Coisas
• Processamento de dados vindos de serviços como: Twitter, Kafka, Flume, AWS Kinesis

Apache Spark GraphX

O GraphX é um dos componentes mais recentes do Spark, tendo como objetivo computação paralela de grafos.

Você quis dizer grafos ou gráficos?

Boa questão. Muitas pessoas confundem esses dois termos. Vamos explicar:

grafo (Graph)
gráfico (Graphic)

Um grafo é uma estrutura matemática usada para modelar relacionamento entre objetos. Um grafo é composto de vértices e arestas. Os grafos são sem dúvida uma das formas mais interessantes de representação de informação, pois diferentemente dos gráficos, que possuem tipicamente relacionamento linear, podemos visualizar nos grafos redes interconectadas, o que representa muito melhor a relação entre objetos.

O conceito de grafos é bastante abrangente - podemos representar muitas coisas do mundo (e suas relações) utilizando esse conceito. É isso que torna a teoria dos grafos um campo tão estudado: as pessoas se debruçam para estudar esse modelo matemático porque sabem que, se conseguirem desenvolver novos trabalhos em cima desses modelos abstratos, esses trabalhos serão aplicáveis a inúmeros problemas reais. Imagine que você conseguiu desenvolver um trabalho muito bom sobre, digamos, aumentar o fluxo entre dois nós de um grafo. Esse seu trabalho poderá ser aplicado desde redes de água (aumentando o fluxo de água) até redes de computadores (aumentando o fluxo de dados transmitidos).

A teoria dos grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Para tal são empregadas estruturas chamadas de grafos, formados por um conjunto não vazio de objetos denominados vértices e um subconjunto de pares não ordenados de vértices, chamados arestas. Para se ter uma ideia de quão importante é a Teoria dos Grafos, saiba que Google Maps e o Facebook o utilizam bastante em seus produtos. Aqui na DSA a Teoria dos Grafos é estudada em detalhes no curso Análise em Grafos Para Big Data, da Formação Inteligência Artificial.

Existem inclusive bancos de dados NoSql do tipo Graph Database, como o Neo4j e o OrientDB, além de soluções analíticas como o SAP Hana, que permite criar análises baseadas em grafos.

A análise de grafos é muito utilizada para Page Rank e filtros colaborativos, onde se busca relação entre diversos objetos. O Page Rank foi o algoritmo inicial usado nas buscas do Google.

Mas o que é o Spark GraphX? Spark GraphX é um framework para processamento de grafos de forma paralela e distribuída através de um cluster. O GraphX estende o conceito dos RDDs, criando os Resilient Distributed Property Graphs.

Ou seja, um RDD para processamento dos elementos de um grafo, como o vértice ou as arestas. Há diversas maneiras de armazenarmos grafos em computadores. A estrutura de dados usada dependerá tanto da estrutura do grafo quanto do algoritmo usado para manipulá-lo. Teoricamente, podemos dividir entre estruturas do tipo lista e do tipo matriz, mas em aplicações reais, a melhor estrutura é uma combinação de ambas.

O GraphX, por ser um componente novo do ecossistema Spark, está disponível apenas em linguagem Scala por enquanto. Criar uma aplicação analítica com o GraphX, passa pelo aprendizado da teoria dos grafos e computação paralela de grafos, elementos normalmente utilizados em Inteligência Artificial.

Deploy do Spark em um Cluster Hadoop com YARN

Submetendo um Job com Deploy em Cluster Mode:

```
# Modo Client
spark-shell --master yarn --deploy-mode client


# Modo Cluster

spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    /opt/spark/examples/jars/spark-examples*.jar \
    10
```

Quizz

Os 3 modos de execução do Spark são o Standalone, YARN e Mesos.

O Spark é um framework para processamento de Big Data construído com foco em velocidade, facilidade de uso e análises sofisticadas. Oferece APIs de alto nível em Java, Scala, Python e mais recentemente em R, bem como um conjunto de bibliotecas que o tornam capaz de trabalhar de forma integrada, em uma mesma aplicação, com SQL, streaming e análises complexas, para lidar com uma grande variedade de situações de processamento de dados.

Diferente das plataformas tradicionais, o Hadoop é capaz de armazenar qualquer tipo de dado no seu formato nativo e realizar, em escala, uma variedade de análises e transformações sobre esses dados.

O Spark armazenará a maior quantidade possível de dados na memória e, em seguida, irá persistí-los em disco.

Mini-Projeto 1 Importando Dados do Banco de Dados Oracle para o HDFS

A Oracle é a líder mundial em banco de dados e uma das gigantes de tecnologia. Bancos de dados Oracle podem ser encontrados em Data Warehouses ou sistemas ERP como SAP, PeopleSoft e JDEdwards. Esses sistemas podem conter bancos de dados com muitos milhões de registros.

Por esta razão, trouxemos para este projeto, todo o processo passo a passo de como importar dados do Oracle para o HDFS utilizando o Sqoop, uma ferramenta ETL gratuita e um dos componentes do ecosistema Hadoop.

O Sqoop permite conectar via JDBC ao banco de dados Oracle, executar uma query, extrair dados e carregar no HDFS, para posterior processamento analítico, através de um cluster.

A Oracle tem investido bastante no Hadoop e sua solução de Big Data, chamada Big Data Appliance, é totalmente baseada no Hadoop. A Oracle fornece ainda conectores e ferramentas de análise com linguagem R, que permitem manipulação de dados no Hadoop

```
# Cria o schema no Banco de Dados e concede privilégios

create user aluno identified by dsacademy;

grant connect, resource, unlimited tablespace to aluno;

# Cria uma tabela

CREATE TABLE cinema (
  ID   NUMBER PRIMARY KEY ,
  USER_ID       VARCHAR2(30),
  MOVIE_ID      VARCHAR2(30),
  RATING        DECIMAL EXTERNAL,
  TIMESTAMP     VARCHAR2(256)
);

# Para carregar dados no Oracle, usamos o SQL*Loader. Este aplicativo requer um control file conforme abaixo

load data
INFILE 'ml-20m/ratings.csv'
INTO TABLE cinema
APPEND
FIELDS TERMINATED BY ','
trailing nullcols
(id SEQUENCE (MAX,1),
 user_id CHAR(30),
 movie_id CHAR(30),
 rating   decinal external,
 timestamp  char(256))

# Executando o SQL*Loader
sqlldr userid=aluno/dsacademy control=loader.dat log=loader.log

# Baixar o driver JDBC
http://www.oracle.com/technetwork/database/features/jdbc/default-2280470.html

# Copiar o arquivo para o diretorio do sqoop
sudo cp ojdbc7.jar /opt/sqoop/lib

# Como usuario aluno, inicializar HDFS e Yarn
start-dfs.sh
start-yarn.sh

# No usuario oracle, configurar as variaveis de ambiente
/home/oracle/.bashrc

# Importação de Dados do Oracle para o HDFS
sqoop import --connect jdbc:oracle:thin:aluno/dsacademy@localhost:1521:orcl --username aluno -password dsacademy --table cinema --columns
"user_id, movie_id" --where "1" -m 1

sqoop import --connect jdbc:oracle:thin:aluno/dsacademy@localhost:1521:orcl --username aluno -password dsacademy --query "select user_id,
movie_id from cinema where rating = 1 and \$CONDITIONS" --target-dir /user/oracle/output -m 1


# Comando usado para corrigir problemas com blocos corrompidos, caso ocorra
hdfs fsck / | egrep -v '^\.+$' | grep -v replica | grep -v Replica

# Para deixar o modo de segurança do Hadoop
hdfs dfsadmin -safemode leave

# Incializar o Oracle

# Linha de comando - incializa o Listener
lsnrctl start

# SQLplus - inicaliza o banco de dados
sqlplus / as SYSDBA
startup
```

Mini-Projeto 2 Prevendo a Ocorrência de Doenças Cardíacas

O que é uma doença cardíaca?

Doença cardíaca é um termo geral para designar diversas condições médicas crônicas ou agudas que afetam um ou mais componentes do coração. Entre os pulmões existe uma cavidade conhecida como mediastino. Este é o lugar onde o coração está posicionado – partindo do centro do corpo humano, dois terços para a esquerda. O coração é um órgão muscular do tamanho de um punho, que bombeia o sangue através da rede de artérias e veias chamada sistema cardiovascular.

O coração tem quatro câmaras:

- Átrio direito: recebe o sangue das veias e bombeia para o ventrículo direito
- Ventrículo direito: recebe o sangue do átrio direito e bombeia para os pulmões, onde ele é carregado com oxigênio
- Átrio esquerdo: recebe sangue oxigenado dos pulmões e bombeia para o ventrículo esquerdo
- Ventrículo esquerdo: bombeia o sangue oxigenado para o resto do corpo. As contrações do ventrículo esquerdo criam a nossa pressão arterial.

As artérias coronárias correm ao longo da superfície do coração e fornecem sangue rico em oxigênio ao músculo cardíaco. Uma teia de tecido nervoso também atravessa o coração, conduzindo os sinais neurológicos complexos que regem a contração e relaxamento. Essa teia que envolve o coração é um saco chamado pericárdio.

A doença cardíaca ocorre quando uma dessas estruturas não está funcionando corretamente. Algumas doenças cardíacas comuns: Angina instável e estável, Arritmia cardíaca, Artrose, Aterosclerose (doença cardíaca coronária), Arteriosclerose, Cardiomiopatia, Cardiopatia congênita, Doença arterial periférica, entre outras.

Diagnóstico e exames

Alguns exames podem ser feitos para diagnosticar ou acompanhar doenças cardíacas. Veja:

- Eletrocardiograma
- Ecocardiograma
- Teste ergométrico
- Cateterismo cardíaco
- Holter 24 horas
- Monitor cardíaco portátil
- Machine Learning

O que? Machine Learning? Sim, isso mesmo.

Cada vez modelos preditivos vem sendo usados para prever a ocorrência de doenças cardíacas, usando algoritmos de classificação. A partir de dados coletados de pacientes, é cada vez mais comum a utilização de Machine Learning para prever a ocorrência de doenças em geral, incluindo doenças cardíacas.

O objetivo deste projeto é servir como um estudo de caso para investigar todos os passos necessários na criação de um modelo preditivo doenças cardíacas. Usaremos diversas das ferramentas do ecosistema Hadoop, que estudamos ao longo do curso.

O dataset pacientes.csv (que você encontra junto com os demais arquivos do projeto) contém as seguintes colunas, com dados coletados de pacientes de um hospital que atende idosos:

Coluna	Descrição
ID	ID único para cada registro
Idade	Idade do paciente
Sexo	Sexo: 0 – Feminino 1 – Masculino
Pressão Sanguínea	Pressão sanguínea medida
Colesterol	Colesterol medido
Açúcar no Sangue	Nível de açúcar no coração: 0 – Nível de açúcar <= 120 mg/dl 1 - Nível de açúcar > 120 mg/dl
ECG	Resultados ECG: 0 – Normal 1- Alguma anomalia 2- Anomalia presente
Batimento cardíaco (Valor Máximo)	Batimento cardíaco medido
Doença	Indica se o paciente tem doença cardíaca: 0 – Não 1 – Sim

→ A variável Doença é a variável target e todas as demais são as variáveis preditoras.

Dataset (sample)

ID	Idade	Sexo	Pressão Sanguínea	Colesterol	Açúcar no Sangue	ECG	Batimentos Cardíacos	Doença
1001	63	1	145	233	1	2	150	0
1002	67	1	160	286	0	2	108	1
1003	69	1	145	235	1	2	129	0
1004	68	1	120	229	0	1	110	0

Seu trabalho, como Cientista de Dados, é criar um modelo preditivo que utilize as variáveis preditoras e seja capaz de prever (a partir de novos conjuntos de dados) se um paciente pode ou não desenvolver doenças do coração.

Solução:

Um grande hospital ou uma rede de serviços de saúde pode ser capaz de coletar grandes quantidades de dados sobre seus pacientes e um cluster Hadoop pode ser a solução ideal para armazenar e processar esse “Big Data da Saúde”. Nossa solução, portanto, vai utilizar um cluster Hadoop e as ferramentas:

Hive – como os dados estão em formato estruturado, usaremos o Hive para armazenar os dados no HDFS e realizar consultas interativas através da linguagem HQL.

Pig – será usado para transformação e pré-processamento nos dados.
Mahout – será usado para construção do modelo preditivo.

A solução contempla 5 etapas:

• Etapa 1 - Carregando o dataset no Hive e visualizando os dados com SQL
• Etapa 2 - Analise Exploratória e Pré-processamento nos dados com Pig
• Etapa 3 - Transformação de Dados com o Pig
• Etapa 4 - Criação do Modelo Preditivo de Classificação
• Etapa 5 - Otimização do Modelo Preditivo de Classificação

Na etapa 1, carregamos os dados em uma tabela criada com o Hive. O Hive é a solução ideal para dados estruturados e permite utilizar a HQL, uma variação da linguagem SQL, que nos permite consultar os dados de forma rápida e eficiente.

Na etapa 2, usaremos o Pig para compreender como os dados se relacionam e realizar análises estatísticas.

Na etapa 3, o Pig será usado para transformar os dados de forma a facilitar o trabalho de construção do modelo preditivo.

As etapas 4 e 5 são a criação do modelo preditivo com algoritmo Random Forest. Inicialmente criamos um modelo e avaliamos a Confusion Matrix e na sequência otimizamos o modelo aumentando o número de árvores de decisão.

```
# Projeto - Prevendo Doencas Cardiacas

# Etapa 1 - Carregando o dataset no Hive e visualizando os dados com SQL

CREATE DATABASE usecase location '/user/cloudera/projeto';

CREATE TABLE pacientes (ID INT, IDADE INT, SEXO INT, PRESSAO_SANGUINEA INT, COLESTEROL INT, ACUCAR_SANGUE INT,
ECG INT, BATIMENTOS INT, DOENCA INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH 'pacientes.csv' OVERWRITE INTO TABLE pacientes;

SELECT count(*) FROM pacientes;

SELECT doenca, count(*), avg(idade), avg(pressao_sanguinea), avg(colesterol), avg(acucar_sangue), avg(batimentos) FROM pacientes GROUP
BY doenca;



# Etapa 2 - Analise Exploratoria e pre-processamento nos dados com Pig

dadosPacientes = LOAD 'pacientes.csv' USING PigStorage(',') AS ( ID:int, Idade:int, Sexo:int, PressaoSanguinea:int, Colesterol:int,
AcucarSangue:int, ECG:int, Batimentos:int, Doenca:int);

REGISTER datafu-1.2.0.jar;

DEFINE Quantile datafu.pig.stats.Quantile('0.0','0.25','0.5','0.75','1.0');

diseaseGroup = GROUP dadosPacientes BY Doenca;

quanData = FOREACH diseaseGroup GENERATE group, Quantile(dadosPacientes.Idade) as Age, Quantile(dadosPacientes.PressaoSanguinea) as
BP, Quantile(dadosPacientes.Colesterol) as Colesterol, Quantile(dadosPacientes.AcucarSangue) as AcucarSangue;

DUMP quanData;


# Etapa 3 - Transformação de Dados com o Pig

ageRange = FOREACH dadosPacientes GENERATE ID, CEIL(Age/10) as AgeRange;
bpRange = FOREACH dadosPacientes GENERATE ID, CEIL(BloodPressure/25) as bpRange;
chRange = FOREACH dadosPacientes GENERATE ID, CEIL(Cholesterol/25) as chRange;
hrRange = FOREACH dadosPacientes GENERATE ID, CEIL(MaxHeartRate/25) as hrRange;
enhancedData = JOIN dadosPacientes by ID, ageRange by ID, bpRange by ID, hrRange by ID;
```

```
describe enhancedData;

predictionData = FOREACH enhancedData GENERATE dadosPacientes::Sexo, dadosPacientes::AcucarSangue, patientData::ECG,
ageRange::AgeRange, bpRange::bpRange, hrRange::hrRange, dadosPacientes::Doenca;

STORE predictionData INTO 'enhancedHeartDisease' USING PigStorage(',');



# Etapa 4 - Criação do Modelo Preditivo de Classificação

# Cria a pasta no HDFS
hdfs dfs -mkdir /projeto

# Copia o arquivo gerado pela transformação com o Pig para o HDFS
hdfs dfs -copyFromLocal enhancedHeartDisease/* /projeto

# Cria um descritor para os dados
mahout describe -p /projeto/part-r-00000 -f /projeto/desc -d 6 N L

# Divide os dados em treino e teste
mahout splitDataset --input /projeto/part-r-00000 --output /projeto/splitdata --trainingPercentage 0.7 --probePercentage 0.3

# Constrói o modelo RandomForest com uma árvore
mahout buildforest -d /projeto/splitdata/trainingSet/* -ds /projeto/desc -sl 3 -p -t 1 -o /projeto/model

# Testa o modelo
mahout testforest -i /projeto/splitdata/probeSet -ds /projeto/desc -m /projeto/model -a -mr -o /projeto/predictions


--> Visualizar a Confusion Matrix

# Etapa 5 - Otimização do Modelo Preditivo de Classificação

# Construir o modelo com 25 árvores, a fim de aumentar a acurácia
mahout buildforest -d /projeto/splitdata/trainingSet/* -ds /projeto/desc -sl 3 -p -t 25 -o /projeto/model

# Testa o modelo
mahout testforest -i /projeto/splitdata/probeSet -ds /projeto/desc -m /projeto/model -a -mr -o /projeto/predictions


--> Aumentando o número de árvoresm aumentamos a acurácia do modelo.
```

Mini-Projeto 3 Design de um Job MapReduce para Gastos Totais por Cliente

Você já parou para pensar quantas vendas são realizadas por dia em grandes empresas como, por exemplo, Amazon ou WalMart? Empresas que faturam bilhões vendendo os mais variados produtos para um grande número de clientes.

E se você fosse contratado para um projeto em uma dessas empresas e seu primeiro trabalho fosse calcular o total de vendas por cliente? Tarefa aparentemente simples. Sua primeira abordagem talvez fosse buscar o banco de dados transacional com as informações de vendas, cruzar os dados com o cadastro de clientes e obter o valor total gasto por cliente. Mas quantos clientes uma empresa como a Amazon possui? E se a solicitação fosse para gerar o total gasto por cliente nos últimos 5 anos, de modo a criar uma campanha personalizada para os clientes que tiveram os maiores gastos ao longo dos anos? Após alguma pesquisa, você poderia obter um dataset no seguinte formato:

Código do cliente → Valor gasto em uma única compra

128 → 899.90
1029 → 349.12
128 → 45.76

Sua pesquisa identificou que todos os registros dos últimos 5 anos geram um dataset com apenas duas colunas, mas 200 milhões de registros. Definitivamente esse não é um trabalho para um banco de dados relacional. Você precisa de uma ferramenta que possa rapidamente processar os dados e retornar apenas um valor total por cliente. Você então decide criar um job de mapeamento e redução. Com poucas linhas de código e usando linguagem Python, você consegue gerar o resultado esperado. Mas ainda tem um problema. Como processar esse job da forma mais rápida possível? Spark/Hadoop é a solução ideal.

Esse é um exemplo claro de projeto de Big Data. Um grande volume de dados e tudo que você precisa é extrair uma simples informação, que poderá fazer toda diferença na estratégia da empresa.

Seu trabalho agora é criar o Job de MapReduce e executá-lo com Spark e Hadoop.

Solução:

Para a solução desse projeto, são necessárias 3 etapas:

1- Instalação do Hadoop e Spark. Você pode utilizar o ambiente virtual criado ao longo do curso ou as máquinas virtuais Cloudera e Hortonworks.

2- Criar um código que faça a leitura do dataset e então para cada código de cliente, busque todos os gastos efetuados e faça a soma. Ao final, deve ser apresentado o resultado.

3- Executar o job via linha de comando usando o Spark-submit

```
# # #  Arquivo: gastos-cliente.py

from pyspark import SparkConf, SparkContext

# Define o Spark Context, pois o job será executado via linha de comando com o spark-submit
conf = SparkConf().setMaster("local").setAppName("GastosPorCliente")
sc = SparkContext(conf = conf)

# Função de mapeamento que separa cada um dos campos no dataset
def MapCliente(line):
    campos = line.split(',')
    return (int(campos[0]), float(campos[2]))

# Leitura do dataset a partir do HDFS
input = sc.textFile("hdfs://clientes/gastos-cliente.csv")
mappedInput = input.map(MapCliente)

# Operação de redução por chave para calcular o total gasto por cliente
totalPorCliente = mappedInput.reduceByKey(lambda x, y: x + y)

# Imprime o resultado
resultados = totalPorCliente.collect();
for resultado in resultados:
    print(resultado)
```

```
# Criar uma pasta no HDFS
hdfs dfs -mkdir /clientes

# Copiar o dataset para o HDFS
hdfs dfs -put gastos-cliente.csv /clientes

# Confirmar que o arquivo foi copiado
hdfs dfs -ls /clientes
```

```
# Executa o job na linha de comando
spark-submit gastos-cliente.py
```