BIG DATA

AULA 2

Prof. Luis Henrique Alves Lourenço

CONVERSA INICIAL

Nesta aula, você se aprofundará nos principais conceitos que envolvem a tecnologia Hadoop. Vamos entender como funciona o ecossistema Hadoop e como cada um de seus principais componentes se relaciona. Veremos de maneira detalhada como funcionam o sistema de arquivos distribuídos Hadoop (HDFS) e a plataforma de processamento MapReduce. Esta aula tem como objetivo oferecer o domínio das ferramentas que envolvem os principais componentes do ecossistema Hadoop.

TEMA 1 – O ECOSSISTEMA HADOOP

A Hortonworks, uma das principais empresas vendedoras de soluções Hadoop, define que “Hadoop é uma plataforma de software *open source* para armazenamento e processamento distribuído de conjuntos de dados muito grandes em *cluster*s de computadores”. Ser uma plataforma de software significa ser um conjunto de software projetado execução em um grupo de computadores, como um *cluster* de servidores, em vez de em um único computador. Sendo uma plataforma de software *open source*, temos uma grande comunidade de usuários e desenvolvedores que podem auxiliar em seu desenvolvimento e correção de problemas.

O armazenamento distribuído é uma das principais características do Hadoop. Como vimos anteriormente, quando se tem um conjunto de dados muito grandes, pode ser bastante complicado armazenar tudo em um único dispositivo. Sendo assim, o Hadoop oferece uma solução de armazenamento de dados muito grandes em um *cluster* de servidores. Esse tipo de armazenamento tem as vantagens da escalabilidade horizontal, como menor custo de aumentar a capacidade, tolerância a falhas, entre outras.

O processamento distribuído é mais uma das principais características do Hadoop. Uma vez que os dados já estão armazenados de forma distribuída, o Hadoop oferece uma maneira de processar esses dados distribuídos pelo *cluster* aproveitando a capacidade de processamento de cada servidor paralelamente.

1.1 Um breve histórico

Originalmente, a criação do Hadoop ocorreu no Yahoo! durante o desenvolvimento de um novo motor de busca *open source* conhecido pelo nome de *Nutch*. Foi quando os desenvolvedores Doug Cutting e Tom White encontraram artigos publicados pela Google, em 2003 e 2004, que descreviam uma tecnologia de armazenamento distribuído, chamada GFS, e uma tecnologia de processamento distribuído, conhecida como *MapReduce*. Eles utilizaram esses artigos para criar sua própria plataforma de armazenamento e processamento distribuído. Tal plataforma foi nomeada de *Hadoop*, inspirado no nome do elefante amarelo de pelúcia do filho de Doug Cutting. Desde então, como um projeto *open source* que hoje faz parte da Apache Software Foundation, o Hadoop seguiu evoluindo em um complexo ecossistema.

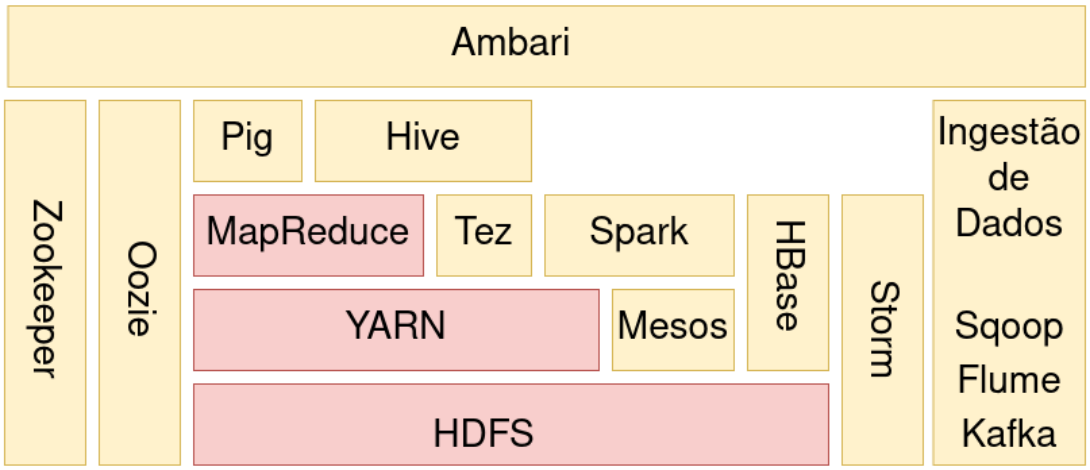
1.2 Componentes do ecossistema Hadoop

O ecossistema Hadoop é composto de diversos módulos intercambiáveis que interagem entre si. Tais componentes podem ser agrupados de diferentes maneiras. Vamos definir um núcleo principal de aplicações que inclui os três principais componentes básicos de todo o ecossistema e alguns componentes auxiliares que acrescentam funcionalidades específicas. Lembrando que todo e qualquer componente pode ser substituído por outros equivalentes.

- HDFS: é a base de toda a plataforma Hadoop. Seu nome é a sigla em inglês para sistema de arquivos distribuído Hadoop. É o componente que permite distribuir o armazenamento de grandes volumes de dados em um *cluster*, comportando-se como um imenso sistema de arquivos. Outra característica do HDFS é que ele mantém cópias redundantes dos dados. Então, se um dos servidores do *cluster* falhar, seus dados podem ser recuperados automaticamente.
- YARN: situa-se na camada logo acima do HDFS. Seu nome é a sigla em inglês para “apenas mais um negociador de recursos”. É o sistema responsável por gerenciar os recursos do *cluster*, ou seja, é o componente que decide qual servidor executará determinada tarefa. Ele também é o responsável pelo balanceamento de carga entre os servidores do *cluster*.
- MapReduce: uma vez que nós temos um sistema de arquivos e um alocador de recursos para o nosso *cluster*, podemos definir quais são as

tarefas que processarão nossos dados. MapReduce é o modelo de programação que permite o processamento dos dados utilizando o *cluster*. Ele é composto de mapeadores (*mappers)* e redutores (*reducers*), que são funções definidas pelo programador. Os mapeadores são utilizados para transformar os dados em paralelo utilizando o *cluster* de maneira eficiente. E os redutores servem para agregar os resultados. Tal modelo de programação parece muito simples, porém, ao mesmo tempo, é muito poderoso.

Figura 1 – Pilha de componentes do ecossistema Hadoop



Esses são os três componentes principais em todo o ecossistema Hadoop, porém, eles podem ser substituídos por outros equivalentes. Além disso, vale destacar que, inicialmente, o YARN era parte integrante do MapReduce. No entanto, eles foram separados para que outras tecnologias pudessem substituir o MapReduce. Também são considerados como parte do ecossistema Hadoop os seguintes componentes:

- Pig – situa-se em uma camada acima do MapReduce. É uma API de alto nível que permite a criação de scripts semelhantes ao SQL para encadear consultas e obter respostas complexas sem a necessidade de escrever códigos Python ou Java, como é o caso das funções que devem ser definidas no MapReduce. O que ocorre, neste caso, é que os *scripts* escritos para o Pig são convertidos em código, que é executado utilizando o MapReduce.
- Hive – também se situa acima do MapReduce e se comporta de forma semelhante ao Pig. No entanto, se parece muito mais com um banco de

dados SQL. Ele recebe consultas SQL e busca de forma distribuída pelos dados que estão no sistema de arquivos de modo muito parecido com um banco de dados SQL, mesmo que na realidade não se trate de um banco de dados relacional.

- Ambari – encontra-se na camada mais alta, uma vez que consiste em uma interface de usuário (UI). Por meio dele, é possível visualizar os recursos do *cluster*, verificar o que está executando no *cluster*, quais sistemas estão utilizando quanto recurso. Ainda, permite executar consultas por meio do Hive, importar bancos de dados para o Hive, executar consultas utilizando o Pig, e muito mais. Cloudera e MapR são interfaces semelhantes que podem substituir o Ambari.
- Mesos – não é propriamente parte do Hadoop, mas pode ser incluído como alternativa ao YARN. Soluciona problemas muito semelhantes com algumas vantagens e desvantagens.
- Spark – encontra-se no mesmo nível do MapReduce, logo acima do YARN, ou do Mesos. Da mesma forma que o MapReduce, necessita de programação em Python, Java ou Scala. No entanto, é muito mais eficiente e poderoso que o MapReduce. Como veremos com mais detalhes nas próximas aulas, Spark é capaz de manipular consultas SQL, efetuar operações de aprendizagem de máquina, manipular fluxos de dados em tempo real (*streaming*), e muito mais.
- Tez – semelhante ao Spark, utiliza grafos acíclicos dirigidos para gerar planos mais otimizados para executar consultas. Geralmente, é utilizado em conjunto com o Hive como modo de torná-lo mais eficiente.
- HBase – situa-se diretamente acima do HDFS e tem por objetivo expor os dados do sistema de arquivos para plataformas transacionais. Consiste em um banco de dados baseado em colunas. É uma forma muito eficiente de expor os dados armazenados no HDFS, como resultados de processamentos para outros sistemas.
- Storm – projetado para realizar o processamento de fluxos em tempo real (*streaming*). Muito útil para processar fluxos de dados vindos de sensores ou *web logs* em tempo real. Spark tem a capacidade de processar fluxos de dados de forma semelhante, portanto é um substituto equivalente.
- Oozie – agenda fluxos de tarefas no *cluster*. Muito útil quando é necessário realizar um trabalho que envolve várias tarefas em sistemas

diferentes. Por exemplo, você pode utilizá-lo para carregar dados por meio do Hive e então realizar alguma tarefa sobre esses dados utilizando o Pig ou o Spark e expor os resultados utilizando o HBase.

- Zookeper – coordenador de *cluster*. É utilizado por muitos componentes para gerenciar os servidores do *cluster*, mantendo sua confiabilidade e desempenho.
- Sqoop, Flume e Kafka – são aplicações para ingestão de dados, ou seja, são utilizados para obter dados de fontes externas para o HDFS. Cada um tem sua especificidade. Sqoop é utilizado para vincular o Hadoop a um banco de dados relacional. Flumes é utilizado para transportar *web logs*. Muito utilizado em conjunto com componentes de processamento de fluxo de dados em tempo real. E o Kafka é um injetor de dados de propósito geral utilizado para coletar dados de diversas fontes e conectá-las com diversos componentes do Hadoop.
- MySQL, Cassandra e MongoDB – armazenamento de dados externo. Podem ser substitutos do HBase.
- Drill, Hue, Phoenix, Presto e Zeppelin – são módulos de consulta SQL para acessar dados no HDFS. Podem trabalhar em conjunto com Hive e HBase.

TEMA 2 – HDFS

O sistema de arquivos HDFS é uma das bases mais importantes de todo o ecossistema Hadoop. Como acabamos de ver, ele é responsável por gerenciar o armazenamento de dados em um *cluster*. Dessa forma, o sistema de arquivos utiliza toda a capacidade de um *cluster* como se fosse um sistema de arquivos em um único servidor. Ele foi projetado para armazenar grandes volumes de dados de maneira eficiente, sendo capaz de evitar a perda de dados.

Para isso, o HDFS quebra os dados em blocos de 128MB por padrão e os distribui pelo *cluster*. Dessa forma, o volume dos dados não se limita à capacidade de um único servidor, sendo possível armazenar dados maiores do que a capacidade de um servidor inteiro. É com base nessa divisão dos dados que se torna possível o processamento distribuído pelo *cluster*. Uma vez que os dados não estão concentrados em um único servidor, partes diferentes dos dados podem ser processadas paralelamente por diferentes servidores.

Além disso, o *cluster* de dados não armazena apenas uma cópia de cada bloco. Ao armazenar mais de uma cópia de cada bloco, a disponibilidade dos dados não é afetada na ocorrência da falha de um servidor, pois nenhum bloco é perdido, uma vez que outros servidores têm uma cópia de cada bloco que estava armazenado no servidor defeituoso. Com isso, podemos dizer que o HDFS é um sistema de arquivos altamente tolerante a falhas. O acesso aos dados feito paralelamente em vários servidores simultaneamente faz com que leituras e escritas no sistema de arquivos sejam bastante eficientes. É importante lembrar que, dessa forma, não é necessário adquirir equipamento especial que permita que a solução tenha alta disponibilidade. Com o HDFS, é possível obter alta disponibilidade utilizando computadores comuns.

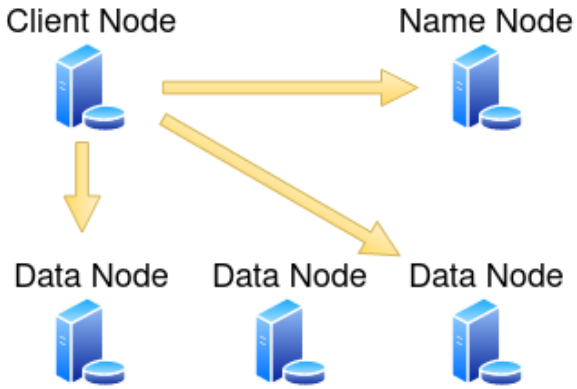
2.1 Arquitetura HDFS

A arquitetura do sistema de arquivos HDFS é dividida em dois tipos de estruturas principais: *Name Nodes* e *Data Nodes*. Um sistema HDFS normalmente tem apenas um *Name Node*. Ele é o responsável por gerenciar os locais onde se encontram cada um dos blocos de determinado arquivo. Além disso, o *Name Node* mantém um registro de edição que contém a informação a respeito dos arquivos criados, o que foi modificado, de forma que o *Name Node* seja capaz de rastrear quais blocos de quais arquivos cada *Data Node* tem. Os *Data Nodes* são os componentes que armazenam os blocos de dados em um servidor. Uma vez que a aplicação cliente já tiver consultado o *Name Node* para saber onde encontrar os pedaços dos dados necessários, ela realiza operações diretamente com os *Data Nodes*. Além disso, os *Data Nodes* trocam informações entre si para garantir a replicação e a integridade de cada bloco dos dados.

2.1.1 Leitura de arquivos

Para ler um arquivo, a aplicação cliente apenas consulta o *Name Node* para encontrar o local onde o arquivo está armazenado. Então, o *Name Node* informa quais são os *Data Nodes* mais eficientes para a aplicação cliente acessar. Assim, a aplicação cliente acessa diretamente esses *Data Nodes* para realizar a leitura do arquivo.

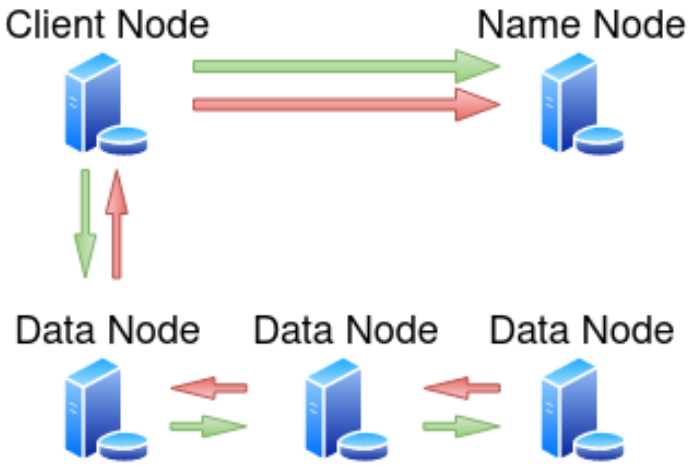
Figura 2 – Leitura de arquivos em HDFS



2.1.2 Escrita de arquivos

Para escrever um arquivo, o processo é um pouco mais complicado. A operação de criar um arquivo exige que a aplicação cliente consulte o *Name Node* para requerer a criação de um arquivo. Então, o *Name Node* registra a criação do arquivo e informa a aplicação cliente para realizar a criação do arquivo. A aplicação cliente se comunica com um único *Data Node,* que se comunica com os demais *Data Nodes* para distribuir as partes dos dados obtidos da aplicação cliente assim que eles estiverem distribuídos e replicados nos *Data Nodes*. O *Data Node* que recebeu a comunicação da aplicação cliente responde para que a aplicação cliente informe ao *Name Node* que o arquivo foi criado e o local onde está armazenado.

Figura 3 – Escrita de arquivos em HDFS



2.1.3 Disponibilidade

Uma preocupação importante é o que acontece quando o *Name Node* falha. Para que as informações contidas nele não se percam, os metadados do *Name Nodes* são replicados em forma de backup constantemente em outro servidor.

Também é possível utilizar um *Name Node* secundário, porém, esse segundo *Name Node* apenas mantém uma cópia de todos os dados do original, de forma que, se ele falhar, o secundário assume o seu lugar.

Outra forma de garantir que os dados do *Name Node* não se percam é utilizando uma federação HDFS, em que cada *Name Node* gerencia um conjunto específico de espaços. Essa solução é especialmente útil quando a quantidade de dados é tão grande que um único *Name Node* não é capaz de registrar todos os arquivos. Então, cada *Name Node* gerencia uma espécie de subdiretório, ou um *namespace*, na estrutura de arquivos HDFS. No entanto, para essa solução, se um dos *Name Nodes* falhar, ocorre a perda parcial dos dados em vez da perda de todos os dados.

Em casos em que se exige uma alta disponibilidade, o componente Zookeper pode auxiliar na recuperação de um *Name Node*.

TEMA 3 – MAPREDUCE

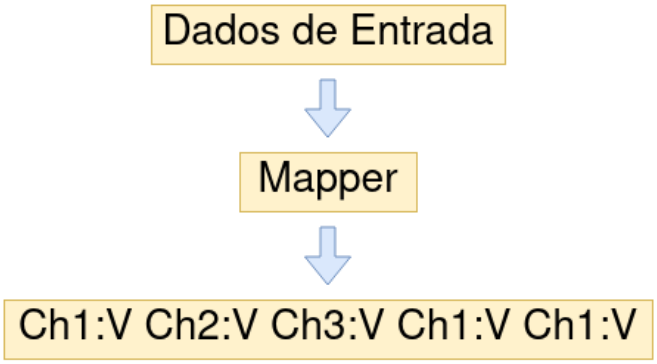
Como vimos, o MapReduce é um dos componentes fundamentais do ecossistema Hadoop. Ele foi desenvolvido como forma de processar os dados distribuídos pelo *cluster*. Como já definimos, o MapReduce transforma pedaços dos dados por meio das funções *Mapper* descritas pelo programador, e agrega os resultados pelas funções *Reduce*, também descritas pelo programador. O MapReduce é um modelo de processamento tolerante a falhas, pois uma aplicação mestre monitora a execução por todos os recursos do *cluster* de forma que, se um recurso falhar, outro pode assumir as suas tarefas automaticamente.

Vamos entender em detalhes como ocorre a execução por meio das principais funções do MapReduce.

3.1 *Mapper*

Mapper é a função em MapReduce que converte os dados ainda não processados em pares chave-valor relevantes para a solução.

Figura 4 – Fluxo de processamento *Mapper*



Dessa forma, estamos mapeando determinado conjunto de valores necessários para a nossa aplicação a uma chave específica, que pode ser algum elemento do conjunto de dados. Dessa forma, o objetivo do *Mapper* consiste em extrair e organizar os dados necessários para o processamento.

3.2 *Shuffle and Sort*

Logo após a execução do *Mapper*, uma etapa de processamento é executada automaticamente. Essa etapa é conhecida pelo nome em inglês *Shuffle and Sort* e significa algo como *embaralhar e ordenar*. O que ocorre neste processo é que as chaves dos pares chave-valor geradas pelo *Mapper* são verificadas e os valores dos pares que apresentam a mesma chave são agregados em uma única lista. Dessa forma, os valores aglutinados passam a ter uma única chave, agregando todos os valores associados a ela.

Esse tipo de operação pode ser muito poderoso quando estamos processando volumes de dados muito grandes, pois diferentes *Mappers* irão processar diferentes partes dos dados. Então, a operação de *Shuffle and Sort* é muito importante para agregar os dados processados por *Mappers* que estavam em servidores completamente diferentes pelo *cluster*, de forma que todos os dados correspondentes a uma mesma chave agora se encontram no mesmo lugar para serem processados juntos.

3.3 *Reducer*

Uma vez que o *Mapper* identificou os dados e o processo de *Shuffle and Sort* os organizou por chave, podemos realizar o processamento dos valores associados a cada chave. É neste ponto do processamento que são produzidas

as respostas que desejamos obter dos dados, ou seja, que informação iremos obter para cada conjunto de dados relacionados a uma chave específica. Cada par de chaves constitui uma tarefa a ser processada pelo servidor que estiver mais próximo dos dados.

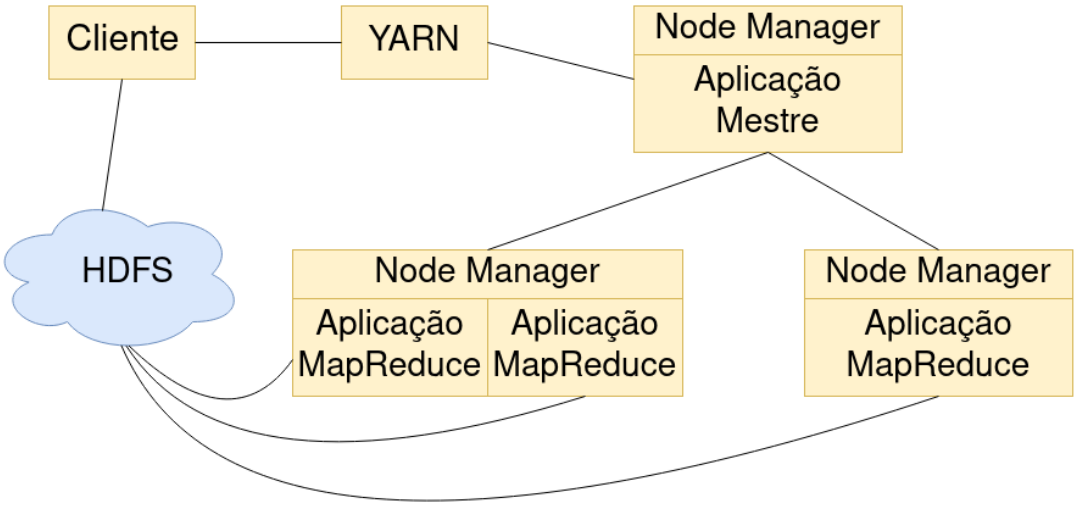
3.4 Distribuição das tarefas no MapReduce

Já entendemos quais são as tarefas executadas pelo MapReduce. No entanto, ainda não sabemos como elas são distribuídas para serem executadas no *cluster* e como são gerenciadas. Tudo começa com um cliente, que neste caso pode ser um terminal, um *prompt* de comando, ou qualquer interface que sirva para o usuário requisitar a execução de tarefas no *cluster*. O cliente requisita o agendamento da execução da aplicação ao gerenciador de recursos. Normalmente, utiliza-se o YARN como gerenciador de recursos. Logo em seguida, se houver a necessidade, requisita-se a entrada dos dados no HDFS para garantir que eles estejam disponíveis para os recursos de processamento realizarem suas tarefas.

Na sequência, o gerenciador de recursos inicia a aplicação mestre do MapReduce. Tanto a aplicação mestre quanto os recursos que processarão as tarefas são gerenciados por um *Node Manager*. O *Node Manager* é responsável por gerenciar quais tarefas cada recurso está processando, o estado de cada recurso, sua disponibilidade e outras informações importantes.

A aplicação mestre gerencia cada tarefa MapReduce a ser executada e divide tais tarefas entre os recursos de processamento do *cluster*. Cada recurso gerenciado pela aplicação mestre acessa o HDFS para obter os dados necessários para executar as tarefas que foram atribuídas pela aplicação mestre.

Figura 5 – Fluxo de processamento MapReduce



A aplicação mestre gerencia seis recursos de trabalho para que seja possível tratar e corrigir problemas imediatamente. Eventualmente, um recurso de processamento pode parar ou necessitar ser reinicializado por algum motivo. Neste caso, a aplicação mestre pode tentar reiniciar o recurso em outro servidor. Pode ocorrer também de a aplicação mestre ficar indisponível. Como no modelo de processamento MapReduce há apenas uma aplicação mestre, quando ela fica indisponível perdemos o gerenciamento sobre as tarefas. Assim, o gerenciador de recursos reinicia a aplicação mestre. Esse tipo de indisponibilidade pode acontecer se o servidor em que esses recursos estão processando parar por qualquer motivo, como um desligamento inesperado. Quando isso acontece, os recursos podem ser reiniciados em outro servidor. Mas o pior caso, certamente, é quando, por algum motivo, o gerenciador de recursos para. Para evitar que isso aconteça, sempre é possível habilitar a configuração de “Alta disponibilidade” para o *cluster* no Zookeeper.

TEMA 4 – ALTERNATIVAS AO MAPREDUCE

Escrever *Mappers* e *Reducers* em Java nem sempre pode ser a forma mais eficiente de se processar grandes dados em Hadoop. Como vamos ver com mais detalhes em breve, o gerenciamento do processamento distribuído foi separado do componente do MapReduce para que outros modelos de processamento distribuído pudessem ser implementados, substituindo, assim, o MapReduce. Uma das formas de estender e otimizar o MapReduce é por meio

do Pig. Mas também podemos substituí-lo utilizando o Tez ou, como vamos ver em outra aula, o Spark.

4.1 Pig

Pig implementa uma linguagem de script criada para simplificar a criação de *Mappers* e *Reducers* com uma sintaxe semelhante a SQL de modo extensível. Mais do que isso, Pig é uma plataforma para analisar grandes volumes de dados que consiste em uma linguagem de alto nível para expressar programas de análises de dado, com uma infraestrutura de avaliação de tais programas. É importante destacar que a estrutura dos programas Pig favorece uma paralelização substancial, o que por sua vez permite o processamento de conjuntos muito grandes de dados.

A camada de infraestrutura do Pig consiste em um compilador que produz sequências de programas MapReduce. A camada de linguagem do Pig consiste em uma linguagem textual conhecida como Pig Latin, que apresenta as seguintes propriedades:

- **Facilidade de programação –** é simples de realizar execuções paralelas e tarefas de análise de dados. Tarefas complexas que compreendem múltiplas transformações de dados inter-relacionais são codificadas explicitamente como sequências de fluxos de dados, tornando-os fáceis de escrever, entender e manter.
- **Oportunidades de otimização –** a forma como tarefas são codificadas permite que o sistema otimize suas execuções automaticamente, permitindo ao programador se concentrar na semântica e na lógica de programação, em vez de pensar na eficiência.
- **Extensibilidade –** usuários podem criar suas próprias funções para propósitos específicos.

Além disso, há a possibilidade de o Pig ser executado em conjunto com o Tez, ou o Spark, obtendo desempenho ainda melhor do que com o MapReduce. Como veremos em breve, o Tez e o Spark utilizam-se de modelos de processamento diferentes para realizar tarefas de forma distribuída sobre o HDFS.

A execução de comandos no Pig pode ser feita utilizando o interpretador Grunt pela linha de comando. Dessa forma, é possível executar comandos Pig

interativamente. Outra forma de processar dados por meio do Pig é executando arquivos de script Pig. Além disso, interfaces de usuário para Hadoop, como Ambari e Hue, também suportam o Pig.

Um dos principais conceitos do Pig são as relações. Uma relação é muito similar a uma tabela em bancos de dados relacionais. No Pig, dizemos que uma relação é uma *bag* de tuplas, ou seja, as tuplas em uma *bag* correspondem às linhas de uma coluna. Dessa forma, relações do Pig não exigem que cada tupla contenha o mesmo número de campos, ou que os campos de uma mesma posição (o equivalente a uma coluna) tenham todos o mesmo tipo. Os principais comandos da sintaxe do Pig incluem:

- **LOAD –** carrega dados do sistema de arquivos;
- **STORE –** armazena ou salva resultados no sistema de arquivos;
- **FILTER –** seleciona tuplas de uma relação baseada em certas condições;
- **DISTINCT –** remove tuplas duplicadas em uma relação;
- **FOREACH –** gera transformações de dados baseadas em colunas de dados;
- **JOIN –** realiza junções de duas ou mais relações baseadas em colunas em comum;
- **COGROUP e GROUP –** agrupa dados de uma ou mais relações;
- **CROSS –** realiza o produto cruzado de duas ou mais relações;
- **ORDER BY–** ordena relações baseadas em um ou mais campos;
- **LIMIT –** limita a quantidade de resultados da tupla;
- **UNION –** computa a união de duas ou mais relações;
- **SPLIT –** divide uma relação em duas ou mais relações.

4.2 Tez

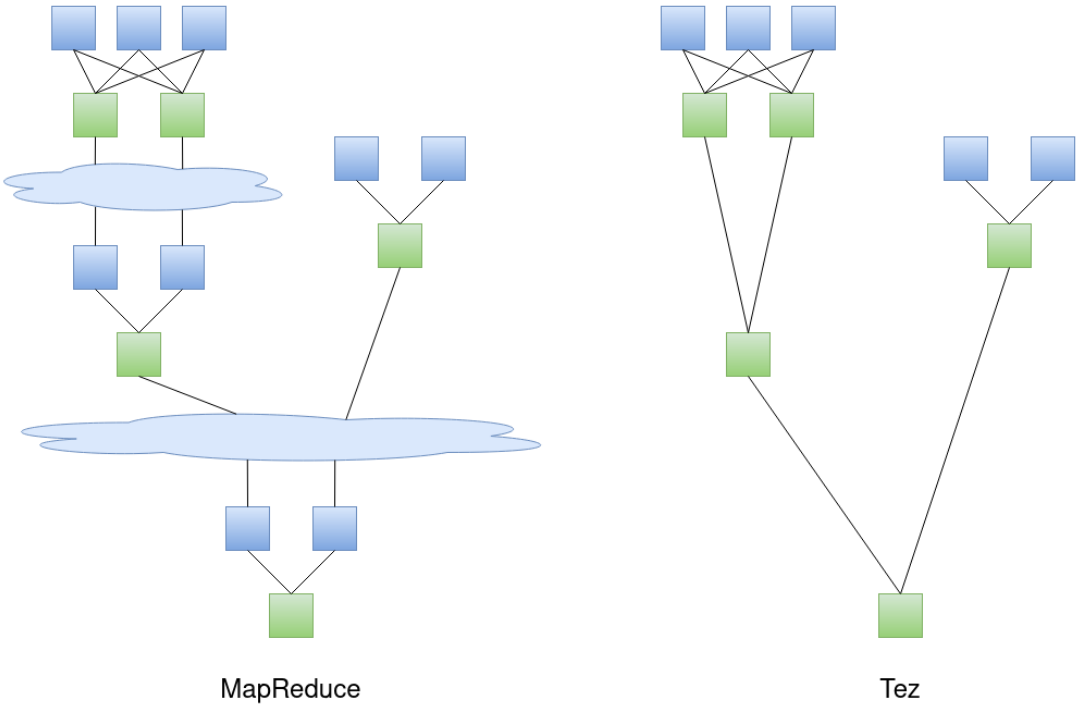
Tez é um componente baseado no gerenciador de recursos YARN que permite o uso de grafos cíclicos direcionados (ou DAG, em inglês) complexos para modelar o conjunto de tarefas que processarão os dados. Portanto, Tez é parte da infraestrutura de processamento que pode ser utilizada por componentes como Hive e Pig para otimizar seu desempenho.

4.2.1 Processamento distribuído baseado em Grafos Acíclicos Direcionados

Um dos problemas de desempenho que o MapReduce apresenta é a interdependência de uma grande quantidade de tarefas executadas de forma sequencial. Arquiteturas de processamento baseadas em grafos cíclicos direcionados, por outro lado, minimizam a dependência entre as tarefas, permitindo que algoritmos distribuídos possam ser executados com ainda mais eficiência, uma vez que são capazes de otimizar o fluxo de dados e de recursos de processamento.

Como podemos observar na figura 6, a cada MapReduce no fluxo de processamento, os dados são obtidos do HDFS, e após o processamento, os resultados são armazenados para que, dessa forma, possam ser utilizados pela próxima execução do MapReduce. O modelo de processamento por grafos direcionados, no entanto, detecta as dependências de dados entre os ciclos de processamento e otimiza o gerenciamento dos recursos.

Figura 6 – Comparação do modelo de processamento entre MapReduce e Grafos Acíclicos Dirigidos



4.2.2 Obtenção de dados distribuídos

Aplicações que utilizam o Tez podem determinar a quantidade de tarefas de leitura dos dados externos necessários. Para isso, existe um inicializador de entrada (*Input Initializer*) específico para cada vértice realizando a leitura. Na terminologia aplicada ao Tez, um vértice representa um passo lógico de processamento, ou seja, uma transformação dos dados. Durante a inicialização de cada vértice, o *Input Initializer* é utilizado para determinar a quantidade de fragmentos de dados externos distribuídos pelo *cluster*. Se a opção de agrupamento estiver habilitada, algumas porções dos dados podem ser agrupadas para melhor balanceamento da paralelização do processamento. Dessa forma, uma quantidade ótima de dados é agrupada em cada vértice.

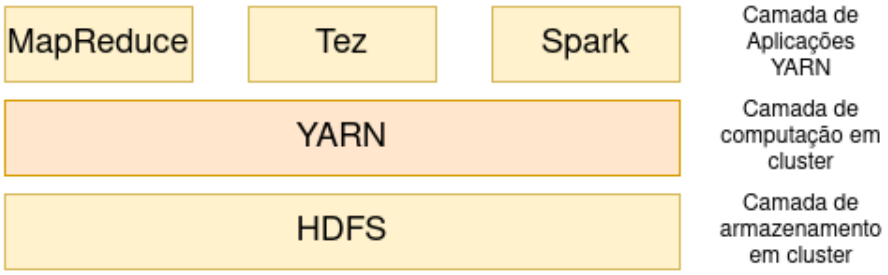
TEMA 5 – GERENCIAMENTO DE *CLUSTER*

É muito importante conhecer também os componentes Hadoop que fazem o gerenciamento e permitem que o *cluster* funcione sem que o programador precise se preocupar com a infraestrutura existente. Vamos entender o funcionamento do gerenciador de recursos YARN, o gerenciador de tarefas Oozie, e das ferramentas que auxiliam a manter a consistência do sistema, como o Zookeeper.

5.1 YARN

YARN é uma tecnologia que foi introduzida no Hadoop2 como forma de retirar do MapReduce a tarefa de gerenciar os recursos do *cluster*. A ideia é ter um gerenciador de recursos global e um gerenciador por aplicação. Isso permitiu a criação de alternativas ao MapReduce, como o Spark e o Tez, que funcionam baseadas no YARN.

Figura 7 – Aplicações YARN



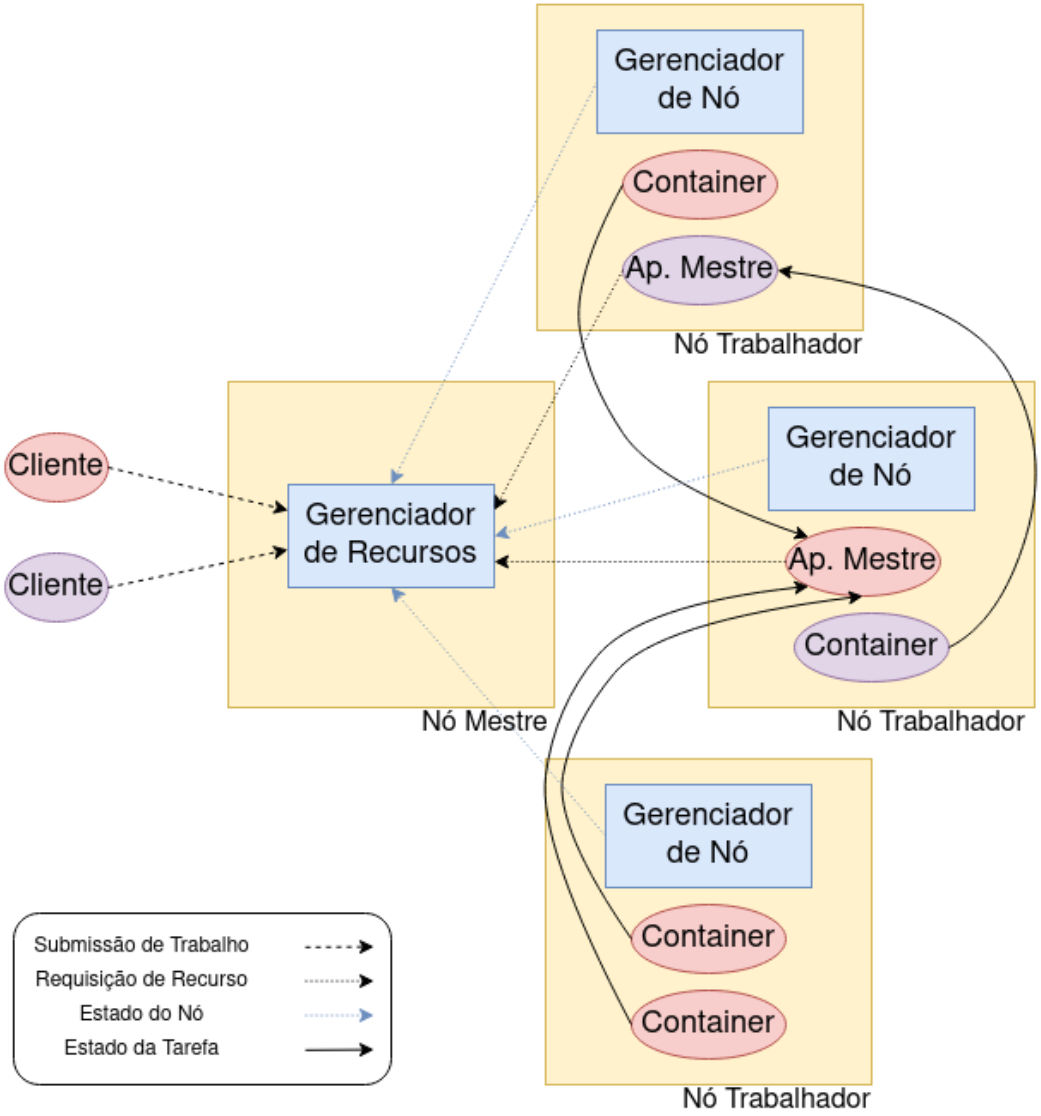
Dessa forma, o YARN foi desenvolvido para ser o componente responsável por distribuir e gerenciar o processamento pelo *cluster*, e, portanto, é altamente integrado ao HDFS. O YARN está relacionado ao gerenciamento dos recursos de processamento do *cluster* da mesma forma que o HDFS está relacionado ao gerenciamento dos recursos de armazenamento do *cluster*.

Para a terminologia utilizada em YARN, podemos dizer que cada computador (ou servidor) é chamado de *nó* (ou *Node*, em inglês). Um *cluster* é o conjunto de dois ou mais nós conectados por uma rede de alta velocidade. No entanto, é tecnicamente possível definir um *cluster* de um único nó, pois se trata de apenas um, normalmente utilizado para testes, e não é recomendado. Existem dois tipos de nó em um *cluster* Hadoop. Conceitualmente, um nó mestre é o ponto de comunicação com o programa cliente (aquele que requisita um trabalho de processamento). O nó mestre, então, faz a divisão do trabalho e envia tarefas para os nós trabalhadores (*worker nodes*).

5.1.1 Arquitetura YARN

O YARN tem um gerenciador de recursos (*ResourceManager*) global executado no nó mestre, e para cada nó trabalhador o YARN tem um gerenciador de nó. Além disso, outro conceito importante é o que define a ideia de *containers*. Um *container* é uma requisição para reservar recursos em um *cluster* YARN.

Figura 8 – Arquitetura YARN



Desta forma, o gerenciador de recursos tem dois componentes principais: o escalonador (*Scheduler*) e o gerenciador de aplicações (*ApplicationsManager*). O escalonador é responsável por alocar os recursos necessários para as aplicações em execução. Ele não monitora o estado das aplicações e não garante a reinicialização de tarefas com falhas, sejam estas em virtude da aplicação ou do hardware. Assim, podemos dizer que o escalonador agenda tarefas com base nas necessidades de recursos da aplicação. O gerenciador de aplicações é responsável por aceitar submissões de trabalho, negociar a criação do primeiro *container* para executar a aplicação mestre (*MasterApplication*) e fornecer o serviço de reinicialização do *container* da aplicação mestre em caso de falha. A aplicação mestre de cada trabalho tem a responsabilidade de

negociar com o escalonador os *containers* com os recursos apropriados para a execução de cada tarefa.

5.2 Zookeeper

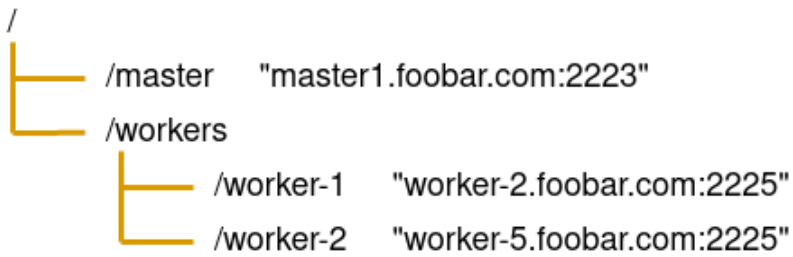
Zookeeper é um serviço *open source* distribuído de coordenação para aplicações distribuídas. Ele é capaz de manter dados de forma consistente por todo o *cluster*. Seu objetivo é livrar as aplicações da responsabilidade de implementar serviços de sincronização. Ele tem um conjunto simples de primitivas que as aplicações distribuídas podem construir para implementar serviços de alto nível para sincronização, manutenção de configuração, gerenciamento de grupos e nomeação. Projetado para ser fácil de programar, utiliza um modelo de dados baseado em uma estrutura de árvores de diretórios. É uma ferramenta que as aplicações podem utilizar para se recuperar de falhas parciais no *cluster*.

Em casos em que o nó mestre falha, o sistema elege um novo mestre, que recebe as informações de execução do nó que falhou. Em caso de falha de nós trabalhadores, o Zookeeper pode notificar a aplicação para que ela possa redistribuir a tarefa adequadamente.

5.2.1 API Zookeeper

O Zookeeper implementa um pequeno sistema de arquivos que permite garantir a consistência de suas informações. Dessa forma, cada “arquivo”, ou znode, na terminologia do Zookeeper, permite manter de forma consistente informações a respeito dos nós e das tarefas de uma aplicação. Os comandos definidos pela API do Zookeeper são: *create*, *delete*, *exists*, *setData*, *getData*, *getChildren* e *sync*.

Figura 9 – Sistema de arquivos Zookeeper



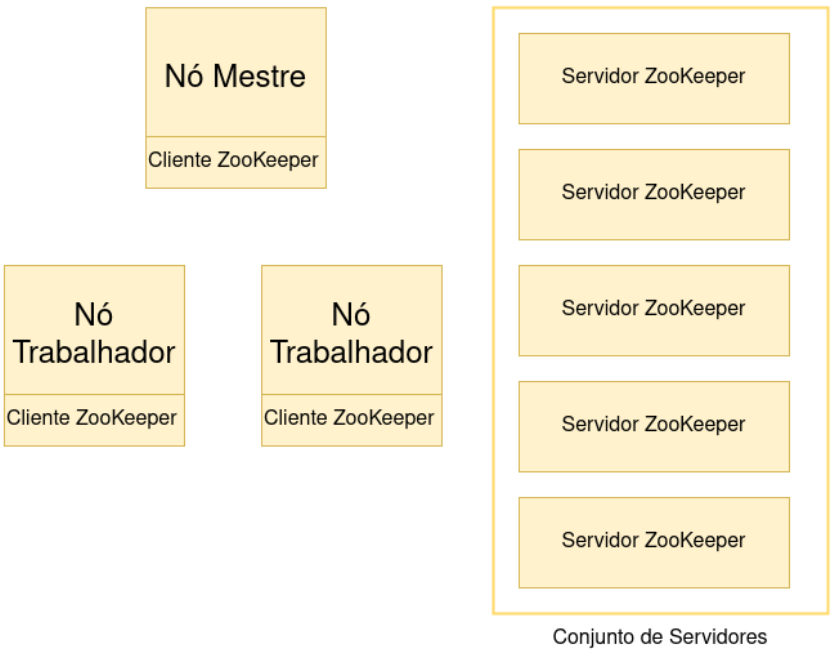
Os znodes do sistema de arquivos Zookeeper podem ser permanentes, como os que armazenam as tarefas dos nós trabalhadores. No entanto, podem ser efêmeros, como os que armazenam o estado de um nó. Por exemplo, quando um nó deixa de atualizar o seu registro no Zookeeper, ele deixa de existir no sistema de arquivos, e, portanto, o Zookeeper sabe que ele se tornou indisponível.

Além disso, um cliente pode se registrar para receber notificações sobre um znode. Assim, é possível uma aplicação se registrar para receber notificações do */master* e, caso o nó mestre deixe de atualizar o seu registro, a aplicação será notificada quando o registro do nó mestre perder a sua validade.

5.2.2 Arquitetura

Cada nó (mestre e trabalhadores) tem um cliente Zookeeper, que monitora as informações que devem ser armazenadas no sistema de arquivos, enquanto um conjunto de servidores Zookeeper gerenciam os clientes. A ideia aqui é não haver um único servidor, pois ele seria um ponto de falha. Dessa forma, os servidores replicam seus dados entre si para que mesmo que alguns deles falhem seja possível continuar operando normalmente. No entanto, é necessário definir um quórum mínimo de servidores para permitir a detecção de falhas no conjunto de servidores Zookeeper.

Figura 10 – Arquitetura Zookeeper



5.3 Oozie

Oozie é o componente que realiza a orquestração de trabalhos no Hadoop. Ele é muito útil quando se deseja implementar um fluxo de dados que utiliza vários componentes do Hadoop de forma encadeada e automática. O Oozie é um sistema que agenda e executa trabalhos no Hadoop. Dessa forma, é muito utilizado para implementar fluxos de trabalho (*workflows*) que encadeiam várias tarefas. Tais fluxos de trabalho são grafos acíclicos direcionados, ou seja, trabalhos sem relação de dependência podem executar em paralelo. A especificação dos fluxos de trabalho Oozie são implementados em notação XML.

Passos para definir um fluxo de trabalho Oozie

1. Garanta que cada ação funcione isoladamente;
2. Crie um diretório no HDFS para o fluxo de trabalho;
3. Crie o arquivo XML “workflow.xml” com a definição do fluxo de trabalho;
4. Crie o arquivo “job.properties” contendo as variáveis necessárias pelo arquivo XML.

Oozie permite a definição de coordenadores com os quais é possível configurar o agendamento dos fluxos de trabalho. Os coordenadores, assim como os fluxos de trabalho, também são definidos em arquivos XML, em que se pode configurar controles de execução, horário de início e frequência. Uma característica muito útil dos coordenadores é a capacidade de iniciar um fluxo de trabalho assim que determinado dado estiver disponível para uso.

Oozie é muito simples e pode apenas ser executado pela linha de comando. No entanto, algumas interfaces apresentam gerenciadores específicos para Oozie.

FINALIZANDO

Nesta aula, vimos os conceitos que envolvem a tecnologia de big data Hadoop. Conhecemos o ecossistema Hadoop, seus principais componentes e como eles se relacionam. Entendemos de forma mais aprofundada o funcionamento do sistema de arquivos distribuído Hadoop (HDFS). Como ele armazena os dados por meio de um *cluster* e como os dados são acessados. Vimos como utilizar o MapReduce para realizar o processamento distribuído dos dados armazenados no HDFS, como funciona a sua estratégia de

processamento, as suas vantagens e limitações. Para além do MapReduce, conhecemos tecnologias que o substituem no processamento distribuído de formas mais eficientes. Assim, vimos a tecnologia por trás do Pig e como ela permite realizar operações muito semelhantes a consultas SQL para acessar os dados armazenados no HDFS. Da mesma forma, conhecemos o modelo de processamento baseado em grafos acíclicos direcionados implementado pelo Tez e suas vantagens em comparação com o MapReduce.

Por fim, vimos algumas das tecnologias utilizadas para gerenciar o *cluster*. Entendemos o funcionamento do YARN, os seus principais componentes e como é feita a distribuição de tarefas pelos nós do *cluster*. Aprendemos a estrutura que permite ao Zookeeper monitorar os nós do *cluster* e suas tarefas de maneira que a consistência do sistema seja mantida mesmo em uma situação de falha parcial. Também conhecemos o orquestrador de trabalhos oozie e aprendemos a criar fluxos de trabalhos para encadear ações utilizando diversos componentes do ecossistema Hadoop.

REFERÊNCIAS

BENGFORT, B; KIM, J. **Analítica de dados com Hadoop:** uma introdução para cientistas de dados. São Paulo: Novatec, 2019.