



# Fundamentos de Big Data

UNIDADE 03

## Ecossistema Hadoop

Profissionais de Tecnologia da Informação (TI) que lidam com *big data* enfrentam desafios bem diferentes daqueles que trabalham em projetos locais (isto é, fazendo projetos para testar no próprio computador) devido ao volume, variedade e velocidade dos dados. Em primeiro lugar, a escala do *big data* requer a utilização de sistemas distribuídos e ferramentas especializadas, como o Hadoop, o Spark e o Databricks, que permitem processar e analisar grandes conjuntos de dados de forma eficiente. Essas ferramentas são essenciais para lidar com a quantidade massiva de dados que não podem ser processados em uma única máquina.

---

**Sistemas distribuídos** são sistemas de computação nos quais os componentes de *hardware* ou *software* estão localizados em diferentes

máquinas interligadas em uma rede, e cujas ações se comunicam e se coordenam por meio da passagem de mensagens. É como se fossem vários computadores que dividem o trabalho entre si.

- **Spark:** Apache Spark é uma plataforma de computação em *cluster rápida* e de código aberto, projetada para processamento de *big data*. Nele, podemos usar linguagens de programação como Scala, Python e R. Também inclui bibliotecas para análise de dados em massa, processamento de *stream*, *machine learning* e geração de grafos.
- **Hadoop:** Apache Hadoop é um *framework* de *software* de código aberto para armazenamento e processamento distribuído de grandes conjuntos de dados em *clusters* de computadores usando programação simples de modelos de programação em escala de dados.
- **Databricks:** Databricks é uma empresa de tecnologia que desenvolve uma plataforma de análise de dados baseada em Apache Spark. A plataforma unifica dados de várias fontes, oferece ferramentas para preparação e exploração de dados e facilita a construção e implantação de modelos de *machine learning*.

Além disso, a variedade de formatos de dados no *big data*, como texto, áudio, vídeo e dados estruturados (como aqueles de uma planilha) requer a aplicação de técnicas avançadas de processamento e análise, como mineração de dados e inteligência artificial. Os profissionais de TI precisam entender não apenas como extrair e transformar esses dados, mas também como interpretar e visualizar os resultados de maneira significativa para os usuários finais.

Por fim, a velocidade com que os dados são gerados e precisam ser processados no *big data* exige que os profissionais de TI estejam sempre atentos às últimas tecnologias e tendências, além de serem capazes de adaptar rapidamente suas estratégias e abordagens. Isso requer um alto nível de habilidade técnica e um profundo entendimento dos princípios do *big data*: portanto, trabalhar com *big data* significa escolher um trabalho desafiador e altamente especializado em TI.



## BUG DO DIA

Não é possível trabalhar adequadamente com *big data* executando as coisas em seu próprio computador. Por isso, precisamos de tecnologias especializadas nisso. A desvantagem é o alto nível de abstração: como são muitos servidores trabalhando em paralelo, perdemos um pouco da noção sobre o que está acontecendo "por baixo dos panos".

Então, a tecnologia de *big data* precisa de uma infraestrutura robusta para lidar com grandes volumes de dados de forma rápida e confiável. Isso inclui uma arquitetura redundante que suporte o processamento paralelo e distribuído. O **Hadoop** é uma das plataformas mais conhecidas para *big data*, utilizando o MapReduce e o Hadoop Distributed File System (HDFS). Vamos explorar os conceitos fundamentais do ecossistema Hadoop neste conteúdo.

Figura 1. Logo do Apache Hadoop



© Apache Software Foundation/Wikimedia Commons

### Infraestrutura de *big data*

Para garantir a alta disponibilidade dos sistemas de *big data* é essencial que os recursos de infraestrutura como servidores, armazenamento e rede sejam resilientes e redundantes (ou, em bom português: que o sistema inteiro não pare de funcionar se

algum servidor der algum problema). Isto evita interrupções nos negócios devido a falhas de *hardware*. A **escalabilidade** também é crucial, permitindo a adição de novos recursos conforme a demanda aumenta, especialmente devido ao crescimento contínuo no volume de dados.

Além disso, a **velocidade** é um aspecto importante. Os recursos de servidor e armazenamento devem ser capazes de processar os dados na mesma velocidade da rede de *big data*. Caso contrário, podem se formar gargalos, especialmente em redes de alta velocidade. Uma solução comum para esse problema é a utilização de **computação em cluster**, distribuindo o processamento entre várias máquinas para aumentar a capacidade de processamento e evitar a sobrecarga de uma única máquina.

Figura 2. *Cluster* de servidores. Aqui, cada "torre" dessas é um servidor. Elas trabalham em conjunto para processar dados.



© IM Imagery/Adobe Stock

A computação em *cluster* é uma abordagem que reúne recursos de várias máquinas para executar tarefas de forma colaborativa. Um *cluster* é gerenciado de maneira centralizada para coordenar o trabalho em cada máquina individual (nó, ou *node*, em inglês). Os benefícios dessa abordagem incluem (Morais et al., 2018):

### Pool de recursos



Com isso, podemos combinar o espaço de armazenamento, CPU e memória disponível de vários servidores para armazenar dados, permitindo o processamento de grandes conjuntos de dados.

---

## Alta disponibilidade

---



Com isso, temos altos níveis de tolerância a falhas e garantias de disponibilidade para impedir que falhas de *hardware* ou *software* afetem o acesso a dados e processamentos, sendo de grande relevância para análises em tempo real.

---

## Escalabilidade

---



A ideia da escalabilidade é a de facilitar o crescimento da operação. Tem muito mais gente usando o seu serviço? Não precisa trocar de servidor, não: apenas adicione mais servidores, e está tudo certo.

---

## Explicando com uma analogia

---

Sei que são vários conceitos sendo apresentados de uma vez só. Portanto, o que acha de pensarmos em uma analogia? Imagine que estamos em uma **praça de alimentação de um shopping movimentado**. Nesses restaurantes, os *chefs* preparam pratos incríveis usando ingredientes de alta qualidade. Assim, temos o seguinte:

### 1. *Cluster*:

- A **praça de alimentação inteira** é o nosso *cluster*. Lá há vários restaurantes, e cada restaurante com as suas respectivas cozinhas, e cada lugar é especializado em diferentes tipos de culinária (dados).
- Cada restaurante representa um **nó** no *cluster*. Esses nós trabalham juntos para processar pedidos e preparar refeições.
- Quando há muitos clientes (pedidos) no shopping podemos pensar em expandir a praça de alimentação ao abrir mais restaurantes (nós) para atender à demanda.

### 2. *Node (Nó)*:

- Cada restaurante individual é um nó.

Os chefs (linguagens de programação e bibliotecas das linguagens) trabalham lá.

- Eles possuem os seus próprios ingredientes, utensílios e habilidades. Alguns são especialistas em sushi (dados estruturados), outros em churrasco (dados não estruturados).
- Se uma cozinha de um restaurante está sobrecarregada, podemos pensar em expandir aquela cozinha (aumentar memória e processador) ou, se isso não for possível, talvez abrir mais restaurantes (comprarmos mais nós) para aliviar a demanda.



© Geraldine Lewa/Unsplash

### 3. Pool de recursos:

- Imagine que temos um **depósito central** e **uma rede de energia elétrica e água centralizadas** no shopping. Esse é o nosso *pool* de recursos.
- Aqui, os restaurantes (nós) compartilham água, luz e alguns outros recursos adicionais, como alguns ingredientes básicos.
- Se um restaurante precisa de mais água para preparar os seus pratos, ele pode consumir da rede compartilhada do shopping. O mesmo vale para a energia elétrica e alguns outros recursos.

### 4. Alta disponibilidade:

- A **praça de alimentação** nunca fecha. Ela está sempre pronta para servir.
- Se um restaurante (nó) tiver problemas ou estiver em reformas, outros restaurantes atendem os clientes.
- Assim, os clientes (usuários) sempre têm acesso a pratos deliciosos (dados) quando desejam.

### 5. Escalabilidade:

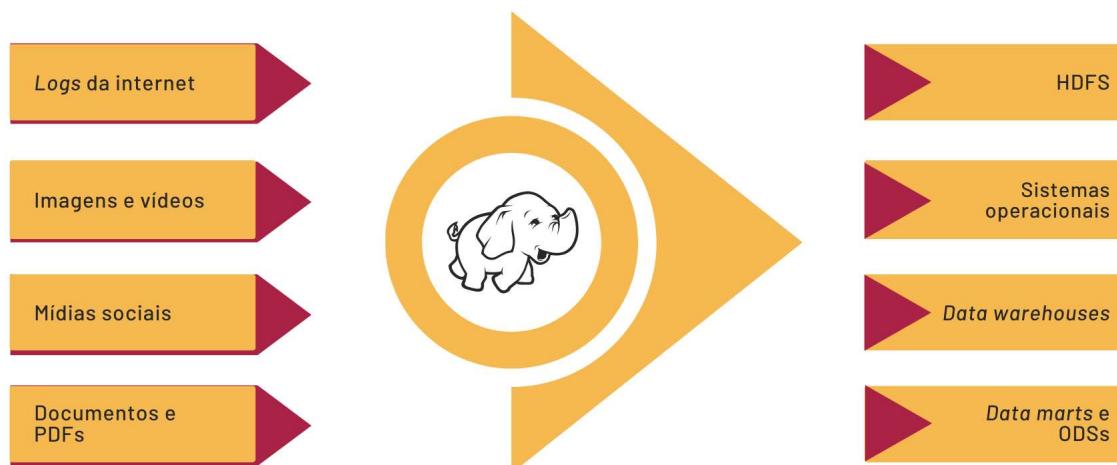
- À medida que mais pessoas chegam ao shopping, podemos pensar em **expandir**.
- Podemos pensar em abrir outras **praças de alimentação** em outros lugares do shopping (*clusters* adicionais) para atender a mais clientes.
- Isso é escalabilidade: a capacidade de crescer conforme a fome (demanda) aumenta.

Ficou mais claro como essa arquitetura técnica funciona? A computação em *cluster*, também chamada de *clustering*, é uma alternativa ao processamento de dados convencional. Neste modelo, as tarefas são divididas em subtarefas e executadas em paralelo em diferentes computadores (nós). Para garantir a eficiência, é crucial dimensionar corretamente as subtarefas, balancear a carga de trabalho e agendar as tarefas adequadamente. Isso otimiza o uso dos recursos computacionais e garante opções de recuperação em caso de falhas nos computadores do *cluster*.

A utilização de *clusters* em uma infraestrutura de *big data* requer uma solução que gerencie a associação dos nós, coordene o compartilhamento de recursos e agende o trabalho em cada nó. Isso significa que soluções de software devem ser utilizadas juntamente com o hardware de um *cluster* para garantir seu uso adequado. No contexto de *big data*, o **Hadoop** é um exemplo de plataforma baseada em computação em *cluster* que oferece alto desempenho para o processamento de dados.

O Hadoop é composto por vários componentes que são usados para processar diferentes tipos e fontes de dados heterogêneos, como imagens, vídeos, mídias sociais, documentos e dados da internet. Esses dados podem ser armazenados no Hadoop, reintegrados aos sistemas de informação operacionais ou preparados para análises posteriores por meio de um *data warehouse* ou *data mart* (Davenport, 2017).

Figura 3. Ecossistema de tecnologia de *big data*



Fonte: Adaptado de Davenport (2017).

O ecossistema Hadoop oferece uma forma rápida de ingerir dados, além de processá-los e armazená-los para reutilização. A seguir, vamos ver mais detalhes dele.

## Hadoop

Houve um tempo em que os dados estavam crescendo a um ritmo exponencial e as empresas estavam lutando para armazenar e processar esses dados. Foi então que o Hadoop entrou em cena.

O Hadoop, nomeado em homenagem ao brinquedo do elefante do filho do seu criador, Doug Cutting, foi desenvolvido pela Apache Software Foundation em 2006. Ele foi projetado para resolver o problema de lidar com *big data* – o que, como você já deve saber, são conjuntos de dados tão grandes e complexos que os sistemas de processamento de dados tradicionais não conseguem lidar.

A importância do Hadoop está na sua capacidade de processar e armazenar enormes quantidades de dados de qualquer tipo, rapidamente e de forma eficiente. Com o Hadoop, as empresas podem analisar os dados em detalhes e usar as informações obtidas para tomar decisões estratégicas.

O Hadoop usa um **sistema de arquivos distribuídos**, conhecido como Hadoop Distributed File System (HDFS), que permite que os dados sejam armazenados em clusters de computadores comuns. Isso significa que as empresas não precisam investir em hardware caro e especializado.



### CURIOSIDADE

O "sistema de arquivos", em termos simples, é a forma como o computador organiza e armazena arquivos em um disco rígido/SSD. É como se fosse a estrutura de pastas em um armário, onde você organiza seus documentos. O NTFS (*New Technology File System*) é um tipo de sistema de arquivos usado principalmente em computadores com Windows. Ele oferece recursos avançados de segurança, como criptografia e controle de acesso, além de suportar arquivos de grande tamanho. Já o HDFS (*Hadoop Distributed File System*) é um sistema de arquivos projetado para lidar com grandes volumes de dados em *clusters* de computadores. Ele divide os arquivos em blocos e os distribui

entre os nós do *cluster* para processamento paralelo, o que é ideal para lidar com big data. Da mesma forma, computadores que usam macOS, Linux e celulares em geral possuem também outros tipos de sistemas de arquivo, como o Ext4 e o APFS.

Além disso, o Hadoop usa um modelo de programação chamado **MapReduce** para processar os dados. O MapReduce divide a tarefa de processamento em pequenas partes, que podem ser processadas simultaneamente. Isso torna o processamento de grandes conjuntos de dados muito mais rápido do que seria possível com sistemas tradicionais.

Hoje, o Hadoop é usado por empresas de todos os tamanhos e setores, desde startups até gigantes da tecnologia como Facebook e Google. Inclusive, o MapReduce nasceu dentro do próprio Google. Ele desempenha um papel crucial na transformação digital, permitindo que as empresas obtenham insights valiosos de seus dados.



### DICA

Pense no Hadoop como uma grande fábrica que produz e processa dados em larga escala, enquanto o MapReduce é o método específico que essa fábrica utiliza para realizar parte desse processamento. O Hadoop é como a fábrica em si, com todos os seus departamentos e maquinário, capaz de lidar com grandes volumes de dados de forma distribuída. O MapReduce, por outro lado, é como um dos processos dentro dessa fábrica, onde os dados são divididos em tarefas menores (*map*) que são processadas em paralelo em diferentes máquinas e depois os resultados são combinados (*reduce*) para produzir o resultado final.

O Hadoop é um projeto de *software*. Isto significa que ele é composto por vários pequenos pedaços, ou subprojetos:

#### Hadoop Common



É basicamente um conjunto de ferramentas compartilhadas usadas por todos os outros projetos do Hadoop. É como o kit de ferramentas essencial que todos os projetos usam para fazer seu trabalho.

---

## Hadoop Distributed File System (HDFS)



É o sistema de arquivos distribuído do Hadoop. Pense em um enorme armazém de dados dividido em blocos. O HDFS é como os trabalhadores que organizam esses blocos de dados para que possam ser acessados e processados de forma eficiente por diferentes partes do sistema.

---

## MapReduce



É como uma equipe de trabalhadores que dividem grandes tarefas em pequenas partes para serem processadas em paralelo. Eles pegam essas partes processadas e as combinam para obter o resultado.

---

## YARN (Yet Another Resource Negotiator)



É como um gerente de recursos em uma empresa, alocando espaço de escritório e recursos necessários para cada equipe de trabalho. No caso do Hadoop, o YARN alocará a quantidade certa de CPU e memória para cada parte do sistema que está executando um trabalho.

---

## Hadoop MapReduce 2



É uma versão melhorada do MapReduce original, que utiliza o YARN para gerenciar melhor os recursos e oferecer suporte a diferentes tipos de aplicativos além do MapReduce.

---

## Hive



---

Funciona como um tradutor que converte consultas em linguagem SQL (que aqui chamamos de HiveQL) em tarefas que o Hadoop pode entender e processar. Ele facilita a análise de grandes conjuntos de dados de forma semelhante ao uso de um banco de dados tradicional.

## Pig



---

É como um assistente pessoal que ajuda a processar e analisar dados de forma mais rápida e eficiente. Ele usa uma linguagem de alto nível, chamada Pig Latin, para simplificar tarefas complexas de processamento de dados.

## HBase



---

É como um arquivo digital de acesso rápido e aleatório. Ele é útil quando você precisa acessar dados rapidamente e não quer esperar pela busca lenta de um arquivo gigante.

---

Desses, talvez os mais importantes sejam o Common, o MapReduce e o HDFS. Os subprojetos centrais do ecossistema Hadoop são o Hadoop MapReduce e o HDFS. Segundo Hurtwitz (2018), esses componentes fornecem a estrutura e serviços de integração básicos para suportar as exigências centrais de soluções *big data*. Os demais subprojetos do ecossistema fornecem os elementos para construir e administrar aplicativos de *big data* com propósitos para o mundo real.

Ainda, o Hadoop apresenta os seguintes componentes e funcionalidades (Santos *et al.*, 2021):

## NameNode



Pense no *NameNode* como o cérebro do sistema de arquivos Hadoop (HDFS). Ele mantém uma lista de todos os arquivos e onde eles estão armazenados no *cluster*.

---

### ***DataNode***



Os *DataNodes* são como os armazéns do Hadoop. Eles guardam os dados reais que compõem os arquivos e os mantêm seguros.

---

### ***JobTracker***



O *JobTracker* é como um maestro no Hadoop. Ele recebe pedidos de trabalho, os divide em tarefas menores e as distribui para serem executadas pelos *TaskTrackers*.

---

### ***TaskTracker***



Os *TaskTrackers* são os trabalhadores do Hadoop. Eles recebem tarefas do *JobTracker* e as executam nos dados armazenados nos *DataNodes*.

---

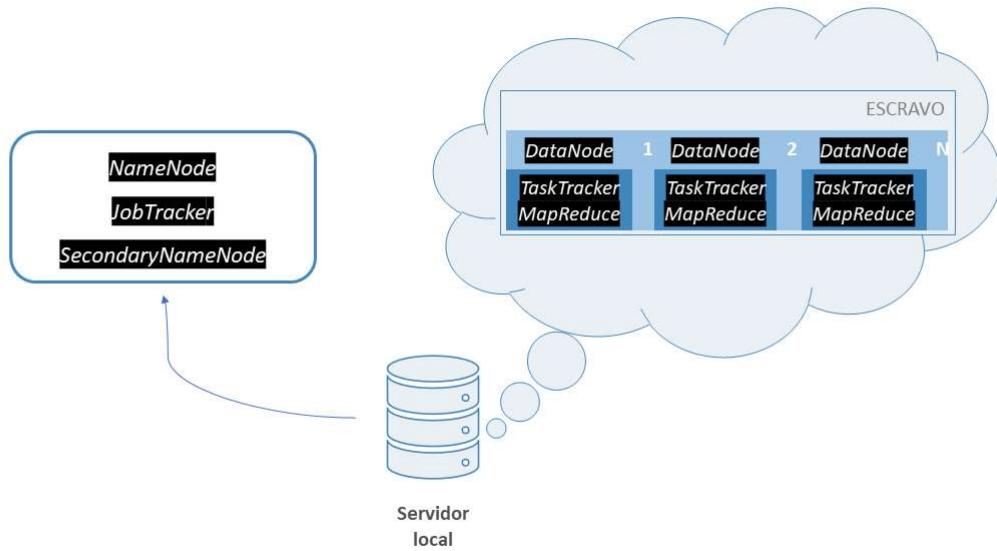
### ***SecondaryNameNode***



O *SecondaryNameNode* é um ajudante do *NameNode*. Ele faz *backup* dos dados do *NameNode* para garantir que não sejam perdidos em caso de falha.

---

Figura 4. Processos de funcionamento dos componentes no Hadoop



Fonte: Adaptado de Santos et al. (2021).

Agora que você conhece os nomes, poderemos falar sobre como eles se interagem. Vejamos:

1. **Conexão do cliente:** o cliente (um usuário, como um analista de sistemas) se conecta ao nó mestre relevante (*NameNode* para acessar o sistema de arquivos HDFS, *JobTracker* para submeter trabalhos MapReduce) para interagir com o sistema e submeter o trabalho.
  - É comum nos referirmos aos nomes em inglês no mercado de trabalho. Portanto, em vez de falar "submeter o trabalho", falarei "submeter o *job*".
  - Vamos usar a partir de agora um exemplo de *job* do MapReduce.
2. **Submissão do *job*:** o usuário submete um job MapReduce para o *JobTracker*, especificando o código MapReduce, os dados de entrada e saída e outros parâmetros.
3. **Divisão do *job*:** o *JobTracker* divide o trabalho em tarefas menores chamadas de mapas e reduções, com base nos blocos de dados do HDFS.
  - É comum falarmos "task" em vez de "tarefa".
4. **Distribuição das *tasks*:** o *JobTracker* distribui as tarefas para os *TaskTrackers* disponíveis no *cluster*, considerando a localidade dos dados para minimizar a transferência pela rede.
5. **Execução dos mapas:** os *TaskTrackers* executam os mapas (subtarefas, ou *subtasks*) nos blocos de dados locais, processando e filtrando os dados conforme necessário.
6. **Shuffle e sort:** os resultados intermediários dos mapas são ordenados e agrupados pelo framework MapReduce para serem passados para as tarefas

de redução.

7. **Execução das reduções:** os *TaskTrackers* executam as tarefas de redução (*reduce*), que combinam e resumem os resultados dos mapas para produzir o resultado final.
8. **Conclusão do job:** após a conclusão de todas as tarefas, o *JobTracker* atualiza o *status* do trabalho e disponibiliza os resultados para o usuário.
9. **Recuperação de falhas:** durante todo o processo, o Hadoop é robusto a falhas, e o *JobTracker* pode reatribuir tarefas a outros *TaskTrackers* em caso de falha de *hardware* ou *software*.

Você lembra que falamos anteriormente sobre escalabilidade? Escalabilidade no Hadoop é a capacidade de aumentar o número de máquinas no processamento de dados, e isto pode ser feito com algumas mudanças simples em um arquivo de configuração, sem a necessidade de mexer no código. Os ajustes se concentram principalmente no espaço de armazenamento e na capacidade de processamento dos computadores envolvidos.

## | Falando mais sobre o HDFS

Lembra que mencionei antes sobre a importância do HDFS? Vamos falar um pouco mais sobre ele agora. O HDFS é um sistema de arquivos distribuídos que permite armazenar e recuperar dados em blocos, distribuídos em um *cluster* de máquinas. Ele foi projetado para lidar com falhas de *hardware*, o que significa que se uma máquina falhar, seus dados ainda estarão seguros e acessíveis.



### EXEMPLO

Em *big data* é comum termos arquivos bem grandes.

Imaginemos, por exemplo, um arquivo com 10 GB (ou 10.000 MB).

Você pode pensar que isto seria igual a 10.240 MB, mas você está pensando em GiB, e não em GB). Quando armazenamos um arquivo grande assim no HDFS, ele é dividido em blocos de tamanho fixo (por padrão, 128 MB ou 256 MB). Cada bloco é então replicado em várias máquinas no *cluster* para garantir a redundância. Isso significa que mesmo que uma máquina falhe, você ainda terá acesso aos seus dados.

O HDFS também tem um design primário/secundário (conhecido antigamente por *master/slave*). O primário, conhecido como *NameNode*, mantém o diretório de todos os arquivos no sistema de arquivos e rastreia onde os blocos de arquivo são armazenados no *cluster*. Os secundários, conhecidos como *DataNodes*, são responsáveis por armazenar e recuperar os blocos de dados quando são instruídos pelo *NameNode*.

O HDFS é otimizado para fornecer *streaming* de dados de alta taxa de transferência, o que o torna ideal para aplicações que requerem processamento de grandes conjuntos de dados, como o MapReduce do Hadoop. É por isso que o HDFS é uma parte crucial do Hadoop – afinal, é ele que permite que armazenemos e processemos grandes volumes de dados de um jeito que seja, ao mesmo tempo, eficiente e confiável.

## | Falando mais sobre o MapReduce

Já o MapReduce é um modelo de programação que permite processar grandes conjuntos de dados em **paralelo**, dividindo a tarefa (*task*) em sub-tarefas (*mapas*) menores que podem ser executadas simultaneamente em diferentes nós de um *cluster* de computadores.

O processo do MapReduce é composto por duas etapas principais – *Map* e *Reduce*:

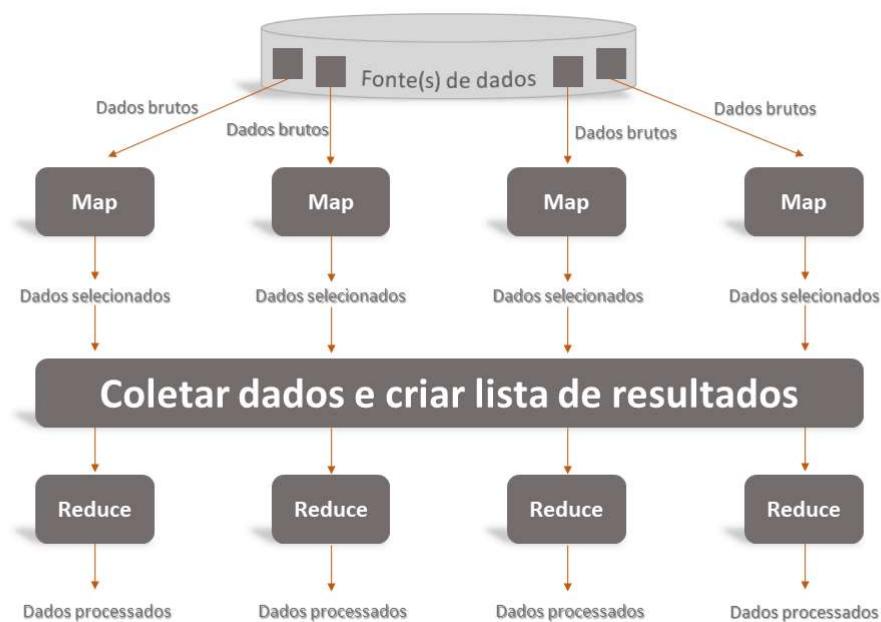
1. *Map*: nessa etapa, a tarefa de processamento é **dividida** em subtarefas menores. Cada subtarefa é então atribuída a um nó diferente no *cluster* para ser processada em paralelo. O resultado de cada subtarefa é um conjunto de pares chave-valor, como um dicionário do Python.
2. *Reduce*: nessa etapa, os resultados da etapa *Map* são coletados e "reduzidos" a um conjunto menor de pares chave-valor. Isso é feito **combinando** todos os valores que têm a mesma chave.

---

Esta estratégia de **dividir primeiro e combinar depois** é bem eficaz para processar grandes conjuntos de dados porque permite que os dados sejam processados onde eles estão armazenados (processamento de dados local). Isso reduz a quantidade de dados que precisam ser transferidos pela rede, o que pode ser um gargalo significativo ao lidar com *big data*.

Logo, o MapReduce é o que permite ao Hadoop processar grandes volumes de dados usando computação distribuída (isto é, várias máquinas ao mesmo tempo). A vantagem disso é a usabilidade: não é necessário ter um curso extenso em programação distribuída para trabalhar com o Hadoop. Se você já tem mais experiência em TI, deve entender que, na prática, isto significa que o Hadoop (e o MapReduce) formam uma **camada de abstração** adicional entre a infraestrutura de TI e o nosso código.

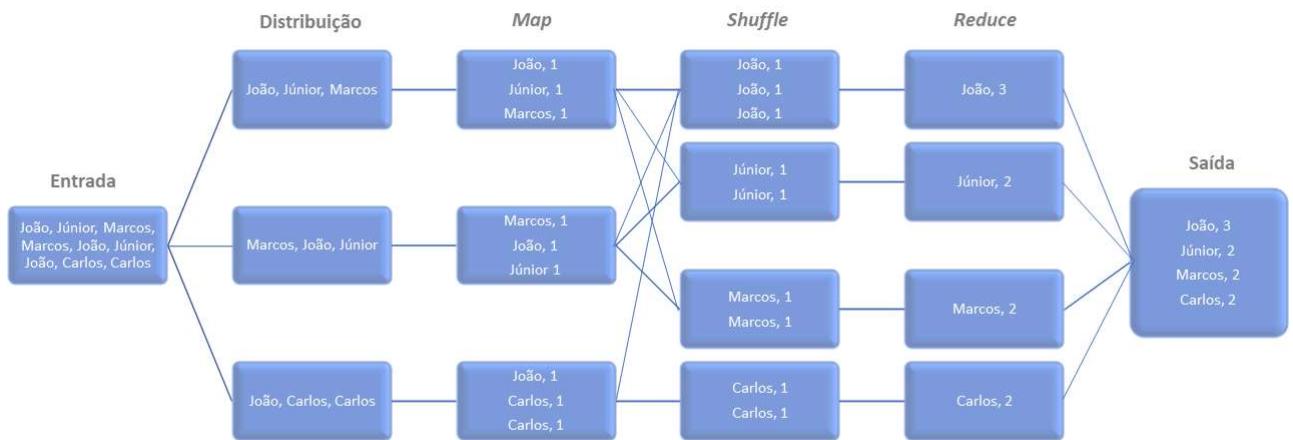
Figura 5. Fluxo de dados em MapReduce



Fonte: Adaptado de Hurwitz *et al.* (2016).

Vamos pensar em um exemplo com alguns dados? Na figura a seguir, temos um exemplo do processamento MapReduce sobre uma entrada de dados.

Figura 6. Exemplo de MapReduce para contagem de palavras



Fonte: Adaptado de Santos et al. (2021).

As seguintes etapas estão apresentadas nesse exemplo:

## Entrada



A entrada consiste em nomes como "João", "Carlos" e "Marcos" repetidos em diferentes ordens. Essa é a fase em que os dados brutos são introduzidos no sistema. Poderíamos ter aqui um arquivo gigante armazenado com o HDFS.

## Distribuição



Aqui, os nomes são distribuídos em diferentes conjuntos como "Marcos, João, Júnior" e "João, Júnior, Carlos". Essa etapa é responsável por dividir os dados de entrada em partes menores que podem ser processadas em paralelo.

## Map



Cada nome é mapeado com uma contagem de 1. Por exemplo, cada ocorrência do nome "João" é mapeada como "João: 1". Esta é a fase em

que cada pedaço de dado é processado individualmente. Veja também que cada uma dessas sequências é um conjunto de chave (ex.: João) e valor (ex.: 1).

## Shuffle (embaralhamento)



Aqui, as chaves são, então, embaralhadas e agrupadas juntas com suas contagens. Isso significa que todas as ocorrências do mesmo nome são agrupadas juntas. Essa etapa é responsável por organizar os dados de tal forma que todas as entradas com a mesma chave estarão juntas.

## Reduce



Nessa etapa, as contagens para cada nome são somadas para fornecer uma contagem total para cada nome. Por exemplo, se "João" aparece três vezes, você terá "João: 3". Essa é a fase em que os resultados intermediários da função *Map* são combinados para formar a saída final.

Santos *et al.* (2021) destacam que o MapReduce é amplamente empregado em agrupamento de dados, aprendizado de máquina, visão computacional, inteligência computacional e algoritmos genéticos de otimização. Essas aplicações geralmente demandam muitos recursos de processamento, o que torna o MapReduce uma escolha ideal devido a sua capacidade de processamento distribuído e paralelo.

Veja alguns cenários em que MapReduce pode ser aplicado (Santos *et al.*, 2021):

- **Agrupamento de dados:** o MapReduce é utilizado em situações em que é necessário agrupar informações com um mesmo valor de função armazenadas em um arquivo e processá-las em outro programa. Isso garante que esses dados sejam processados como um único grupo, facilitando a análise e manipulação dos dados.
- **Filtro, análise e validação:** em sistemas que possuem um grande registro de informações, como análise de logs, consulta e validação de dados, o MapReduce é utilizado para coletar todos os registros que correspondem ao que se busca. A função *map* é responsável por obter os registros, enquanto a função *reduce* entrega os itens recebidos após serem transformados, simplificando o processo de análise e validação.

- **Execução de tarefas distribuídas:** o MapReduce é essencial para executar tarefas distribuídas, onde uma grande quantidade de dados é dividida em múltiplas partes. Esses dados são armazenados, mapeados e os cálculos necessários são executados para gerar o resultado esperado. Esse processo distribuído ajuda a melhorar a eficiência e a velocidade do processamento de grandes volumes de dados.

Basicamente, no MapReduce permite a análise de dados em bases de grande volume pela coleta dos dados relevantes e filtragem das entradas, garantindo que os resultados sejam resumidos em saídas menores e mais relevantes.

## | YARN

O Hadoop estava se tornando a solução de escolha para armazenar e processar uma alta quantidade de dados alguns anos atrás. No entanto, havia um problema. O Hadoop estava limitado a executar apenas tarefas MapReduce, o que significava que outras formas de processamento de dados, como **processamento em tempo real** e **processamento de grafos**, não eram possíveis.

Foi então que o YARN (*Yet Another Resource Negotiator*) entrou em cena. Introduzido no Hadoop 2.0, o YARN foi uma grande revolução que transformou o Hadoop de um sistema de processamento de dados baseado em MapReduce em um sistema de processamento de dados de propósito geral.

O YARN atua como o **gerenciador de recursos** do Hadoop, permitindo que várias aplicações de processamento de dados sejam executadas no mesmo *hardware* do Hadoop. Isso significa que, além do MapReduce, o Hadoop agora pode executar aplicações que exigem diferentes modelos de processamento de dados, como o Spark para processamento em tempo real e o Giraph para processamento de grafos.



### DICA

Falaremos sobre o Spark em breve!

---

A introdução do YARN também melhorou a eficiência do Hadoop. Antes do YARN, o Hadoop tinha que reservar uma quantidade fixa de recursos para cada tarefa, independentemente de a tarefa precisar de todos esses recursos ou não. Com o YARN, o

Hadoop pode alocar recursos de forma mais flexível, o que significa que pode executar mais tarefas ao mesmo tempo.

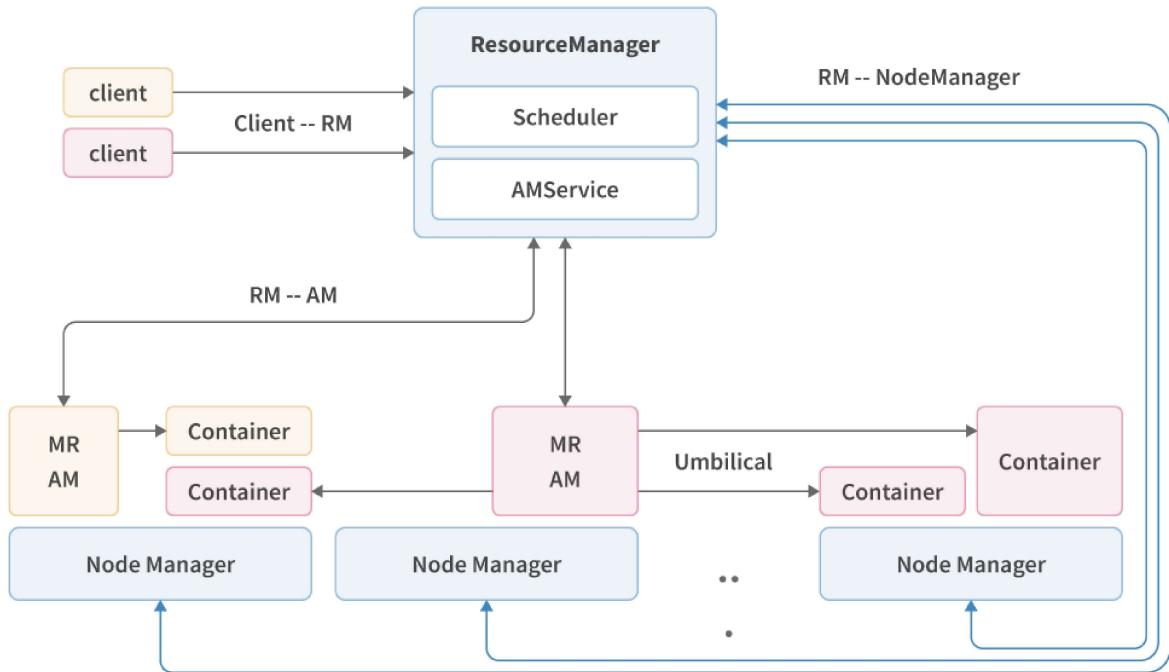
O YARN possui dois componentes principais:

- O *ResourceManager* é responsável por gerenciar os recursos no cluster, como CPU e memória, e coordenar as tarefas entre os *NodeManagers*.
- Os *NodeManagers* são executados em cada máquina do cluster e são responsáveis por monitorar o uso de recursos localmente e executar tarefas conforme solicitado pelo *ResourceManager*.

Na imagem abaixo, podemos ver a arquitetura do YARN com os seus componentes destacados em azul, enquanto os componentes das aplicações estão em amarelo e rosa. No *ResourceManager* (RM), há o *scheduler*, responsável por alocar recursos do sistema para os aplicativos em execução. Todas as informações do sistema necessárias são armazenadas em um *container*, com detalhes sobre CPU, disco, rede e outros recursos necessários para executar os aplicativos no nó e em um agrupamento.

Cada nó possui um *NodeManager* (NM) conectado ao *ResourceManager* global no cluster. Para cada aplicativo em execução no nó, há um *MapReduce Application Master* (MR-AM) correspondente. Se mais recursos forem necessários para dar suporte ao aplicativo em execução, o *Application Master* notifica o *NodeManager*, que, então, negocia a capacidade adicional com o *ResourceManager* (*scheduler*) em nome do aplicativo.

Figura 7. Arquitetura do YARN



Fonte: Adaptado de Santos *et al.* (2021, p. 168).

Vamos entender um pouco mais o esquema acima? Ele mostra a arquitetura de gerenciamento de recursos em um sistema de computação distribuída, que é uma parte fundamental do YARN no Hadoop. Aqui temos o seguinte:

**Client:** esse é o cliente que submete a aplicação ao sistema, como um analista de sistemas que está em um computador.

**ResourceManager (RM):** esse é o sistema primário que gerencia a alocação de recursos computacionais disponíveis no sistema para as aplicações.

**NodeManager:** esse é o sistema secundário que lança e monitora os *containers*.

**Container:** esse é um conjunto de recursos como memória, CPU, disco, rede etc. que são alocados a uma aplicação.

**ApplicationMaster (AM):** esse é um programa específico da aplicação que negocia recursos do *ResourceManager* e trabalha com o *NodeManager* para executar e monitorar as tarefas.

**MapReduce ApplicationMaster (MR AM):** esse é um tipo específico de *ApplicationMaster* para aplicações MapReduce.

O fluxo de trabalho começa com o cliente que submete uma aplicação ao *ResourceManager*. Assim que foi aceito, o *ResourceManager* aloca recursos para iniciar o *ApplicationMaster*. O *ApplicationMaster* então negocia recursos adicionais do *ResourceManager* conforme necessário e trabalha com o *NodeManager* para executar e monitorar as tarefas. O processo continua até que a aplicação seja concluída.

Com o Apache YARN, foi possível o desenvolvimento de diversos *frameworks* de processamento distribuído, como aplicações para processamento em tempo real (Spark, Samza e Storm). Existem, ainda, *frameworks* que facilitam a construção de aplicações sobre o YARN, como o Apache Slider e o Apache Twill.

## Hive

Agora, outra história: havia um tempo em que o Hadoop estava se tornando a solução de escolha para armazenar e processar esses dados. Porém, como tudo na vida, havia um problema. O Hadoop era ótimo para lidar com *big data*, mas exigia conhecimento em Java e MapReduce, o que era um obstáculo para muitos analistas de dados que estavam mais familiarizados com SQL (ou simplesmente para quem não gostava de Java mesmo).

Foi então que o Hive entrou em cena. Desenvolvido pelo Facebook e depois doado para a Apache Software Foundation, o Hive foi projetado para permitir que pessoas com conhecimento em **SQL** pudessem executar consultas em dados armazenados no Hadoop.

---

Ou seja: o Hive funciona traduzindo consultas SQL em tarefas MapReduce, que são então executadas no Hadoop. Isso significa que os analistas de dados podem usar uma linguagem que já conhecem para trabalhar com *big data*, tornando o Hadoop acessível para mais pessoas. Sabe aquele(a) amigo(a) que trabalha como *front-end*, que até entende SQL, mas que quer ficar bem longe de Java? Pois bem: essa pessoa também pode usar o Hadoop com o Hive!

Além disso, o Hive também introduziu uma estrutura de tabela, semelhante à encontrada em bancos de dados relacionais em SQL. Isto permite aos usuários organizar dados no Hadoop de uma maneira mais estruturada. Isso torna muito mais fácil para os usuários entenderem e trabalharem com seus dados.

Assim, com o Hive você tem acesso tipo SQL (uma implementação SQLite simples, chamada HiveQL) a dados estruturados e análises de *big data* sofisticadas, com o MapReduce. O Hive apresenta três mecanismos para organização de dados (Hurwitz, 2016):

## Tabelas

Consiste em linhas e colunas, mapeadas em diretórios (pastas) no HDFS. Também suporta tabelas armazenadas em outros sistemas nativos.

## Partições

Uma tabela Hive pode suportar uma ou mais partições, que são mapeadas em subdiretórios no sistema de arquivos de base, e representam a distribuição de dados por toda a tabela.

## *Buckets*

No Hive, os dados podem ser divididos em

*buckets.* Estes são armazenados como arquivos no diretório de partição no sistema de arquivos de base. Têm base no *hash* de uma coluna na tabela.

Já o *Metastore* é uma base de dados relacional contendo descrições detalhadas do esquema Hive, incluindo tipos de coluna, proprietários, dados de chave e de valor, estatísticas de tabela e assim por diante. Permite sincronizar dados de catálogo com outros serviços de metadados no ecossistema Hadoop.

## Apache Spark e o Databricks

Comentamos brevemente sobre o Spark aqui. Assim como outras "partes" do Hadoop que surgiram para resolver alguns problemas em específico, com o Spark não foi diferente. O Hadoop era ótimo para lidar com *big data*, mas o processamento de dados em **tempo real e a análise interativa** eram desafios.

Foi para isso que o Spark foi criado. Desenvolvido na Universidade da Califórnia, em Berkeley, o Spark foi projetado para lidar com esses desafios. Ele poderia processar dados em tempo real e permitir a análise interativa, algo que o Hadoop MapReduce não poderia fazer.

O Spark é uma estrutura de processamento de dados em memória, o que significa que ele pode processar dados **muito mais rápido do que o MapReduce**, que depende de operações de leitura e gravação em disco. Isso torna o Spark ideal para aplicações que requerem latência baixa, como processamento de dados em tempo real.

Além disso, o Spark também suporta várias linguagens de programação, como **Java, Scala e Python**, tornando-o acessível a um público mais amplo. Ele também vem com bibliotecas integradas para *machine learning*, processamento de grafos e análise

de dados, tornando-o uma ferramenta bem interessante para *big data* e ciência de dados.

Hoje, o Spark é uma parte essencial do ecossistema Hadoop, permitindo que as empresas processem seus dados de um jeito melhor. De fato, o Databricks é uma empresa que foi fundada pelos mesmos criadores do Spark. É, de uma forma bem simplista e resumida, **uma plataforma colaborativa que usa o Spark e notebooks de código**.

## Conclusão

Nesta unidade, exploramos o ecossistema Hadoop, apresentando uma visão geral de seus principais componentes. O HDFS atua como um repositório distribuído de armazenamento de dados, permitindo que grandes quantidades de dados sejam armazenadas em diferentes nós de um *cluster* de computadores. O MapReduce, considerado o coração do Hadoop, possibilita o processamento eficiente de dados por meio de tarefas de mapeamento e redução, distribuídas e executadas em paralelo. Por fim, discutimos o Spark e o Apache YARN, responsável pelo gerenciamento dos recursos utilizados pelo Hadoop, garantindo uma distribuição equilibrada e eficiente dos recursos do cluster.

## Referências

DAVENPORT, T. H. **Big data no trabalho:** derrubando mitos e descobrindo oportunidades. Rio de Janeiro: Alta Books, 2017.

GRUS, J. **Data science do zero:** noções fundamentais com Python. Rio de Janeiro: Alta Books, 2021.

HURWITZ, J. et al. **Big data para leigos.** Rio de Janeiro: Alta Books, 2016.

IBM. **Hadoop vs Spark:** qual é a diferença? Disponível em:

<https://www.ibm.com/cloud/blog/hadoop-vs-spark>. Acesso em: 17 jul. 2023.

MORAIS, I. S. et al. **Introdução a big data e internet das coisas (IoT).** Porto Alegre: SAGAH, 2018.

PEREIRA, M. A. et al. **Framework de big data.** Porto Alegre: SAGAH, 2019.



© PUCPR - Todos os direitos reservados.