



Fundamentos de Big Data

UNIDADE 04

HDFS

Se você procura uma carreira em TI eu entendo que você *gosta* de computadores, certo? E, se gosta de computadores, entendo que você tenha um pouco de interesse de entender um pouco mais sobre como eles funcionam. Portanto, vamos conversar um pouco sobre sistemas de arquivos.

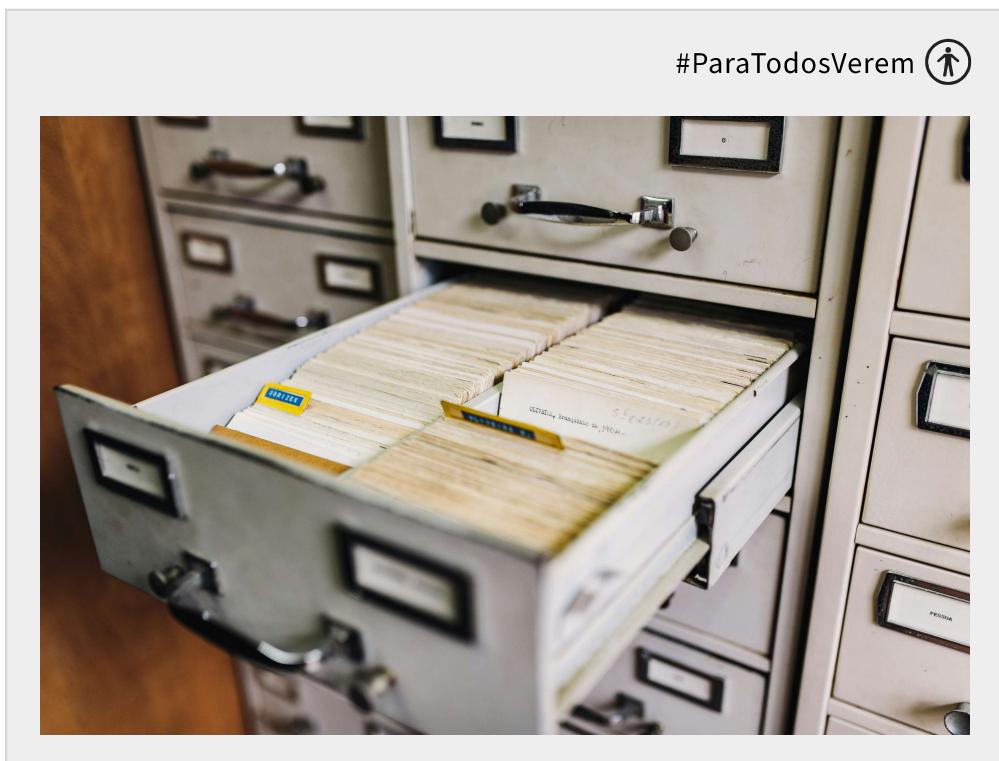


IMPORTANTE

Em cursos tecnológicos é raro falarmos sobre a História da Computação. Afinal, o nosso interesse é muito mais na **prática** do que na **teoria**. Isso posto, para que possamos falar sobre o HDFS é imprescindível entendermos um pouco do histórico das coisas. Na

semana anterior conversamos brevemente sobre o HDFS: agora, nos aprofundaremos sobre ele.

Antes mesmo da existência dos computadores já tínhamos "sistemas de arquivos". São formas de organizarmos dados. Na época, eram dados físicos, como folhas de papel organizadas por pastas em gavetas, tal como nesta imagem. É inspirado nestes exemplos do mundo físico que temos o conceito de arquivos e pastas de computador.



© Maksym Kaharlytskyi/Unsplash

Agora, e como guardávamos os dados antigamente em computadores? Vejamos:

Década de 1950

Fitas magnéticas



Nos primeiros dias da computação, os dados eram armazenados em **fitas magnéticas**. Elas eram eficientes para a época, mas tinham a desvantagem de serem um meio de armazenamento sequencial. Isto significa que não era necessariamente uma forma rápida de acessar dados, já que você precisaria ler toda a fita até encontrar um dado específico.

Década de 1960

Discos rígidos



Foi nesta época em que fomos introduzidos aos **discos rígidos** (*hard disks*, ou HDs). Eles permitiam o acesso **direto** aos dados, o que era muito mais rápido do que as fitas magnéticas. Contudo, estes HDs eram muito grandes (maiores do que uma geladeira) e muito caros.

Na realidade, os primeiros HDs começaram a ser vendidos na segunda metade da década de 1950. Contudo, foram popularizados na década seguinte.

Esta época é conhecida como a era **pré-relacional** dos dados: os dados eram gravados em sistemas de banco de dados hierárquicos ou em rede. No modelo hierárquico, os dados eram organizados em uma estrutura de árvore, com um nó raiz conectado a diversos nós filhos, o que tornava complexo representar relações muitos para muitos. Já no modelo em rede, os dados eram organizados em uma estrutura de grafo, mais eficiente para representar esses tipos de relações, mas ainda complexo para consultas. Ambos os modelos eram mais rígidos e difíceis de modificar do que o modelo relacional, que surgiu na década seguinte.

Década de 1970 **Sistemas de arquivos hierárquicos** ▼

Até chegarmos aqui não tínhamos muito a noção de organização de arquivos por pastas (diretórios). Foi nesta década em que os sistemas de arquivos começaram a se tornar mais sofisticados. Os **sistemas de arquivos hierárquicos**, como o sistema de arquivos do Unix, foram desenvolvidos nesta época. Estes sistemas de arquivos organizavam os arquivos em uma estrutura de diretórios (pastas), tornando-os mais fáceis de gerenciar. É a mesma ideia que usamos até hoje em nossos computadores.

Na mesma época, surgiu o conceito de **modelos relacionais** dos dados. Foi aqui que surgiu o conceito de tabelas de dados baseados na álgebra relacional, com uma estrutura fixa e rígida. Conceitos como normalização, chaves primárias e ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

surgiram aqui. Este foi o modelo que vimos na disciplina de Banco de Dados para TI, e que é amplamente usado até hoje.

Década de 1980 Modelos de dados orientados a objetos



Na década de 1980, a Microsoft introduziu o sistema de arquivos **FAT** (*File Allocation Table*) com o MS-DOS. Já o Linux contava com o **ext2** (*second extended file system*).

Foi nesta década em que surgiram os **modelos de dados orientados a objetos**. Em 1980, com as linguagens de programação orientadas a objeto (Delphi e C++), surgiu o modelo de banco de dados orientado a objetos. Esse modelo auxiliava no suporte de forma natural aos objetos das linguagens de programação.

Década de 1990 NTFS



Em 1993, com o Windows NT 3.1, a Microsoft introduziu o **NTFS** (*New Technology File System*), que oferecia melhor desempenho, confiabilidade e segurança do que o FAT. Usamos o NTFS até hoje, no Windows. Nesta mesma década de 1990, com o advento da Internet e o crescimento explosivo dos dados, surgiram os primeiros sistemas de arquivos projetados para lidar com **grandes volumes de dados**.

Foi nesta década em que surgiram os **modelos multidimensionais** e o conceito de *data warehouse*, impulsionados principalmente pelo crescimento da internet. Isso trouxe novos requisitos para o armazenamento de dados, como a necessidade de redundância, escalabilidade, segurança e capacidade de processamento de dados não estruturados. Essa evolução foi fundamental para lidar com a crescente quantidade e complexidade dos dados gerados na época.

O Linux introduziu novos sistemas de arquivos, como o **ext3** e o **ext4**. Também foi nesta época em que surgiu o **sistema de arquivos distribuídos** do Hadoop, o **HDFS**. Ainda, foi nesta época em que fomos introduzidos aos **sistemas de arquivos em nuvem**, como o Amazon S3.

Nos anos 2000, surgiram os **bancos de dados não relacionais**, também conhecidos como **NoSQL**. Eles se diferenciam dos bancos de dados relacionais por não possuírem um esquema de tabelas em linhas e colunas normalizadas. Em vez disso, empregam o conceito de chave-valor (*Key Value Store*, ou **KVS**), onde cada dado é associado a uma chave única. Essa chave de partição é utilizada para recuperar valores, conjunto de colunas ou documentos semiestruturados que contenham atributos de itens relacionados. Os bancos de dados NoSQL oferecem mais flexibilidade e escalabilidade horizontal, o que significa que podem se expandir para lidar com grandes quantidades de dados distribuídos em várias máquinas. Essa característica os torna especialmente indicados para lidar com *big data*.

Armazenamento em nuvem

Você já pensou ter que armazenar e processar um alto volume de dados (digamos, 500 TB) em seu próprio computador? Isto exigiria uma quantidade enorme de espaço de armazenamento e poder de processamento, o que pode ser caro e bem difícil de gerenciar, ainda mais se várias pessoas precisarem acessar diferentes informações ao mesmo tempo.



© Tumisu/Pixabay

É aqui que o armazenamento em nuvem entra em cena. A nuvem oferece uma solução para o desafio do Big Data, fornecendo uma plataforma escalável e flexível para armazenar e processar grandes volumes de dados. Com a nuvem, você pode facilmente aumentar ou diminuir sua capacidade de armazenamento e processamento conforme necessário, o que significa que você só paga pelo que usa.



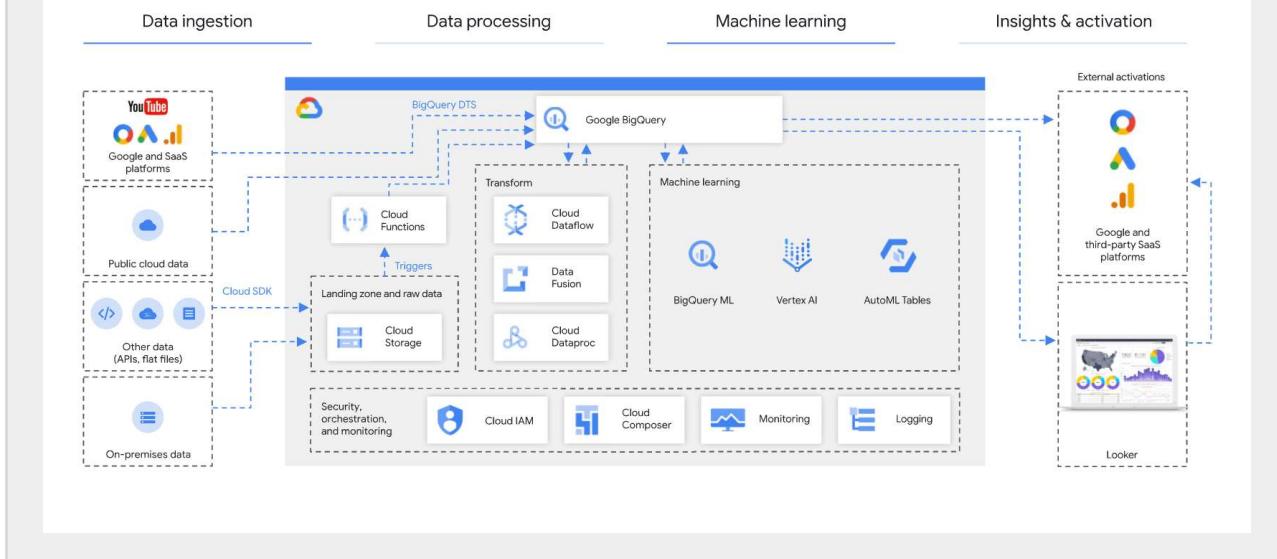
DICA

A computação em nuvem é como alugar um espaço virtual para armazenar seus arquivos e executar programas pela internet, em vez de usar o armazenamento e o poder de processamento do seu próprio computador. Para as empresas, significa não precisar comprar servidores para testar um produto ou solução. Pequenos desenvolvedores podem colocar as suas ideias no ar com um baixo investimento, **pagando somente pelo que efetivamente usou**. Os maiores fornecedores de serviços em nuvem são Google (Google Cloud Platform, ou GCP), Amazon (Amazon Web Services, ou AWS), Microsoft (Azure) e IBM (IBM Cloud).

Muitos provedores de nuvem oferecem serviços de análise de *big data* que podem ajudar a extrair *insights* bem importantes desses dados. Isso pode incluir coisas como *machine learning*, análise preditiva e mineração de dados: é por isso que falamos sobre nuvem aqui.

Veja um exemplo disso na imagem abaixo. Ela é um exemplo de arquitetura em nuvem da Google de um *data warehouse* para um departamento de marketing de uma empresa. Todas as soluções em nuvem da Google estão na caixa em cinza-claro, no centro da imagem. Aqui, temos várias soluções de *big data*, como o BigQuery (um *data warehouse* sem servidor) e o Dataproc (um serviço para executar *clusters* do Spark e Hadoop e que usa o HDFS).

Marketing analytics reference architecture



© Google Cloud/CC BY 4.0

Também temos na imagem o Cloud Storage, um serviço de armazenamento de dados em nuvem para *big data* que não usa HDFS, mas que é compatível com ele. Da mesma forma, Microsoft (Data Lake Storage), Amazon (S3) e outros provedores em nuvem oferecem serviços parecidos. No caso do Amazon S3, trata-se de um serviço projetado para armazenar qualquer quantidade de dados. Tem como pontos fortes o desempenho e escalabilidade, sendo os dados armazenados em *buckets* (na tradução literal, *balde*). Mas na prática lembram grandes pastas separadas umas das outras). É possível selecionar uma ou mais regiões globais para armazenamento físico, a fim de resolver problemas de latência ou atender às necessidades regulatórias de cada país.

Um aspecto importante da computação em nuvem é a **latência**, que é o tempo de atraso para enviar dados e receber uma resposta entre um sistema local e um sistema na nuvem. Esse problema de latência na transferência de dados é cada vez mais relevante e é considerado um dos critérios para decidir se sistemas de *big data* devem ou não ser armazenados na nuvem. Eles podem interagir com um *data warehouse* também baseado na nuvem ou com um *data warehouse* localizado em um data center.

Mas, e então? Quais tipos de DW existem?

***Data warehouse* tradicional**

Temos um problema comum em empresas grandes: nestes lugares, temos uma grande quantidade de dados espalhados em diferentes sistemas. Cada sistema armazena os dados de uma maneira diferente, e isso pode tornar difícil para você obter uma visão completa e precisa dos dados. Aqui é onde entra o *Data Warehouse* (DW).

Um DW funciona como um grande armazém de dados que coleta informações a partir de uma variedade de fontes de bases relacionais e as organiza de uma maneira que facilita a análise e a geração de relatórios. Ele é projetado para armazenar dados históricos e cumulativos que são usados para previsão, relatórios e análise de dados. É claro que isso não funciona como um toque de mágica: precisamos garantir que estas informações sejam consistentes e que os dados em si estejam limpos e bem integrados. Por isso, os princípios de um DW são (Hurwitz *et al.*, 2016):

+ Orientado ao sujeito

Um DW é organizado em torno de assuntos importantes para a organização, como clientes, produtos ou vendas. Por exemplo, em um e-commerce, o DW pode ser organizado em torno de assuntos como pedidos de clientes, detalhes do produto, histórico de vendas, entre outros.

+ Organização de eventos relacionados

Os dados em um DW são organizados de uma forma em que os eventos relacionados estejam conectados. Por exemplo, em um DW de um e-commerce, as informações sobre uma venda (como o cliente que fez a compra, o produto comprado, a data da compra etc.) seriam armazenadas juntas para facilitar qualquer análise feita por um analista de dados ou cientista de dados.

+ Informação não volátil

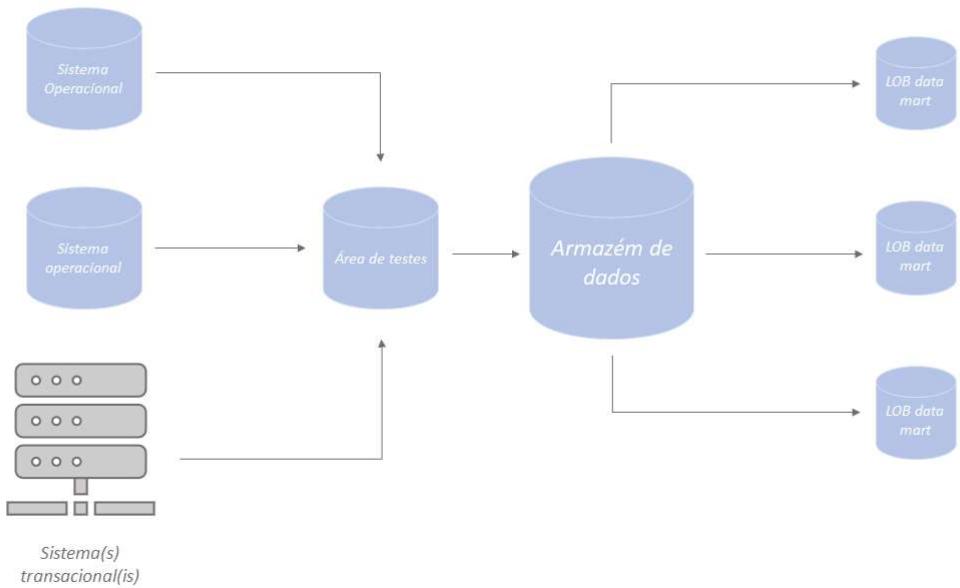
Uma vez que os dados são carregados em um DW, eles não são alterados ou excluídos. Isso significa que o DW mantém um registro histórico dos dados, permitindo análises de tendências ao longo do tempo. Mesmo que um produto deixe de ser vendido no e-commerce manteríamos todo o seu histórico para análises futuras.

+ Inclusão de todas as fontes operacionais aplicáveis

Um DW deve incluir dados de todas as fontes operacionais relevantes dentro de uma organização, com definições consistentes e valores atualizados. Isso significa que se uma empresa tem vários sistemas diferentes onde os dados são criados e armazenados (como um sistema de gerenciamento de relacionamento com o cliente, um sistema de ponto de venda, um sistema de gerenciamento de estoque, entre outros), o DW deve incluir dados de todos esses sistemas. Além disso, os dados devem ser consistentes (o mesmo termo não deve ser usado para significar coisas diferentes em diferentes sistemas) e atualizados (os dados no DW devem refletir as informações mais recentes disponíveis).

Os *Data Warehouses* tradicionais foram projetados para lidar com dados estruturados, principalmente para sistemas de informação operacionais e transacionais das empresas, sendo altamente centralizados. Com o surgimento do *big data*, estes sistemas passam por mudanças significativas para se adaptarem às novas demandas das organizações. No entanto, é importante ressaltar que esses sistemas são altamente estruturados e otimizados para finalidades específicas. A figura a seguir ilustra a abordagem de fluxos de dados com DW e repositórios de dados tradicionais.

Figura 1. Fluxo de dados para armazéns de dados e repositórios de dados tradicionais



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

Observe que os dados gerados pelos sistemas de informação operacionais e transacionais alimentam o armazém de dados, permitindo que sejam derivados em *Line of Business (LOB) data marts*. Exemplos de LOBs comuns incluem vendas, "marketing", "financeiro", "logística" e "recursos humanos". As empresas devem continuar a utilizar esse modelo para administrar os dados estruturados e operacionais que caracterizam sistemas de registro. Assim, fornecerão aos analistas de negócios a capacidade de analisar informações-chave, tendências e assim por diante.

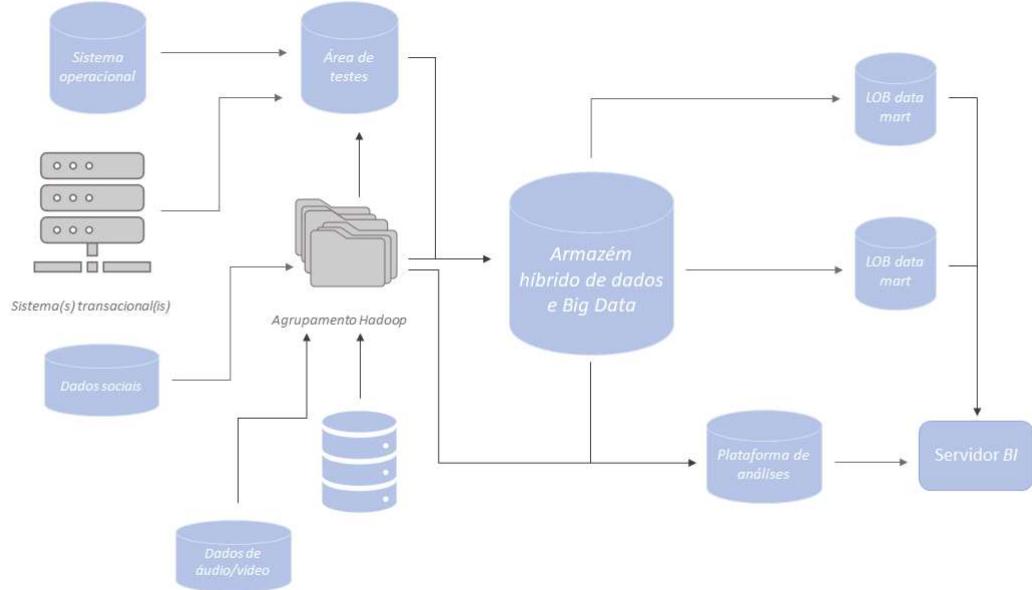
Modelo híbrido *big data* e *data warehouse*

Por outro lado, o *big data* muda a forma como os DW são estruturados, promovendo um modelo híbrido. Nesse modelo, os dados operacionais altamente estruturados e otimizados permanecem no DW tradicional, enquanto os dados altamente distribuídos e sujeitos a mudanças em tempo real são gerenciados por uma infraestrutura baseada em Hadoop (ou um sistema NoSQL semelhante).

Essa abordagem híbrida combina as capacidades do armazém de dados com as do ambiente de *big data*. Ela permite que os dados operacionais estruturados interajam com as fontes de informação menos limpas ou perfiladas do mundo do *big data*. Isso significa que as empresas podem manter uma base de dados de negócios estável em seus armazéns tradicionais, que contêm dados históricos confiáveis, ao mesmo tempo

em que aproveitam fontes de dados menos estruturadas e mais dinâmicas do *big data*. A figura a seguir ilustra um exemplo dessa abordagem de hibridação entre armazenamento tradicional e *big data*.

Figura 2. Fluxo de dados para armazenamento de *big data* e repositório de dados



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

Neste exemplo, vamos considerar a implementação de um *cluster* Hadoop que recebe dados de várias fontes, como dados de mídia social, dados de áudio/vídeo e outros tipos de dados, alimentando um modelo de armazenamento híbrido de *big data*. Isso permite a inclusão de novas plataformas de análise, juntamente com a análise de mineração de dados e inteligência de negócios (BI).

Neste cenário, antes de combinar os dados históricos de transação com o *big data* menos estruturado, fazemos uma análise inicial de um grande volume de dados. Essa análise pode revelar padrões interessantes, como prever mudanças sutis nos negócios ou encontrar soluções potenciais para diagnósticos médicos. A análise inicial é conduzida usando ferramentas como o MapReduce, com o suporte do HDFS. Durante esse processo, são removidos os dados desnecessários e identificados os dados relevantes para o contexto do negócio. Depois dessa etapa, as informações restantes são transformadas para garantir que os metadados estejam corretos e, em seguida, combinadas com os dados históricos tradicionais do armazém. Isso resulta em análises mais precisas e significativas.

A propósito: que tal retomarmos um pouco a conversa sobre o HDFS?

| HDFS

O HDFS é um **sistema de arquivos distribuído** (lembra do histórico de sistemas de arquivos?) que faz parte do *framework* Hadoop. Ele foi projetado especialmente para guardarmos e processarmos grandes volumes de dados em um *cluster* de computadores. A ideia principal do HDFS é **dividir os dados em blocos e distribuí-los por vários nós** de um *cluster* de computadores. Isso permite que grandes volumes de dados sejam armazenados e processados de forma eficiente. Além disso, o HDFS é altamente tolerante a falhas. Isso significa que, mesmo que um dos computadores do *cluster* falhe, os dados ainda estarão disponíveis sem interrupção.

Um sistema de arquivos distribuído como o HDFS deve garantir:

- **Segurança:** acesso às informações, tendo o controle de privacidade e gerenciando as permissões de acesso.
- **Tolerância a falhas:** mesmo que um dos computadores do cluster falhe, os dados ainda estarão disponíveis sem interrupção.
- **Integridade:** controle das modificações, conforme as permissões. Isto é: não é qualquer um que pode mexer em todos os dados livremente.
- **Consistência:** garantia de que o dado seja o mesmo para todos.
- **Desempenho:** o sistema de arquivos distribuídos deve ter alto desempenho.

O HDFS não deve ser considerado como o lugar em que **armazenaremos permanentemente** os arquivos, para sempre. Na verdade, ele é um serviço de dados que oferece um conjunto de capacidades necessárias para **processar** grandes volumes de dados a altas velocidades, sendo uma boa escolha para *big data*.

Arquitetura HDFS

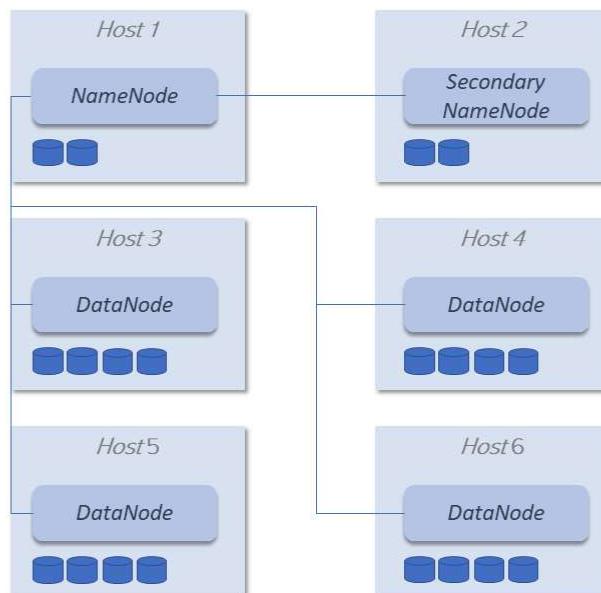
Veja que até o momento explicamos como funciona o *MapReduce* e algumas outras partes da arquitetura do Hadoop, mas não falamos sobre como funciona o HDFS. Para isso, vamos pensar em uma analogia, beleza? Vamos supor que você tenha um grande livro que precisa ser guardado, mas ele é tão grande que não cabe em uma única prateleira.

O HDFS funciona mais ou menos assim, dividindo este livro em várias partes e guardando cada parte em uma prateleira diferente, espalhadas por vários armários. No caso, este livro grande seria um arquivo gigante que precisaríamos processar de forma distribuída por vários computadores (nós).

O HDFS possui dois tipos principais de nós: o *NameNode* e os *DataNodes*. O *NameNode* é como o gerente geral da biblioteca, que sabe onde cada parte do livro está guardada. Ele não guarda os dados em si, apenas mantém um registro de onde os dados estão localizados nos *DataNodes*. Já os *DataNodes* são como as prateleiras dos armários, onde as partes do livro são realmente guardadas. Cada *DataNode* armazena várias partes do livro e faz cópias das partes de outros lugares para garantir que, se uma prateleira molhar ou pegar fogo, ainda teremos outras cópias em outras prateleiras. É daí que vem a "tolerância a falhas" que comentamos anteriormente.

Uma das principais vantagens do HDFS é permitir o armazenamento de dados em qualquer formato, controlando os arquivos que são distribuídos pelos nós (servidor do *cluster*) do sistema. Ou seja: podemos guardar vídeos gigantes (como um filme em 4K), arquivos em tabelas ou qualquer outro formato. É baseado em processos que rodam em segundo plano, conhecidos como *daemons*, que ficam residentes no processamento das máquinas, conforme podemos ver na figura a seguir.

Figura 3. Arquitetura HDFS



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

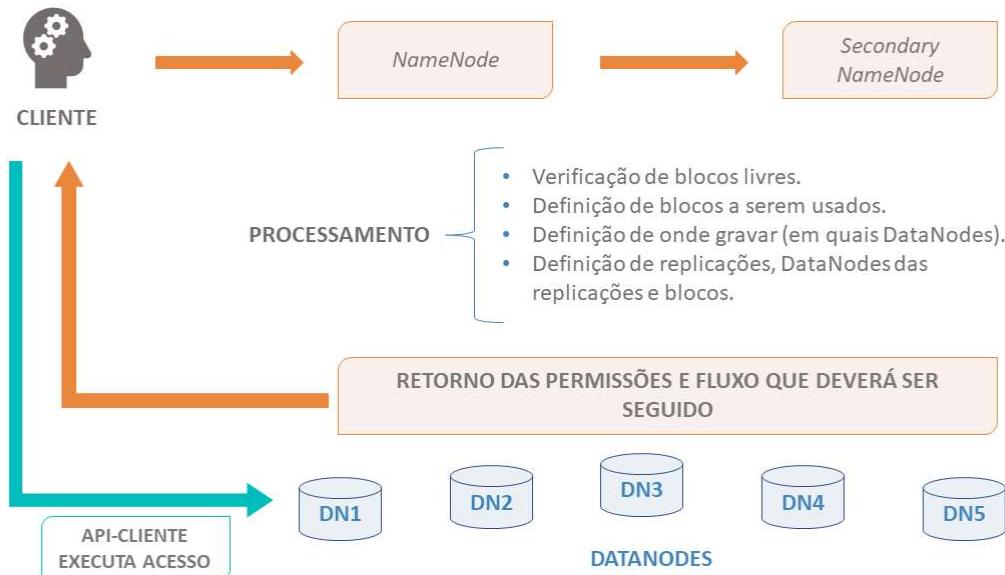
Vamos entender melhor esta imagem? Ela é a representação visual do HDFS. Veja aqui que temos três componentes principais:

1. O **NameNode** é o sistema principal (antigamente chamado de "mestre", ou *master*). Na imagem, ele está no canto superior esquerdo. Veja como ele está conectado aos **DataNode/Secondary NameNode**. É ele quem gerencia o espaço de nomes do sistema de arquivos e regula o acesso aos arquivos pelos clientes.
 - À direita do **NameNode** temos o **Secondary NameNode**. Ele não é um **NameNode** de *backup*, mas, sim, um ajudante que realiza tarefas de manutenção, como criar pontos de verificação e compactar o *log* de transações do **NameNode**.
 - Veja que temos vários **DataNodes**, mas somente **um NameNode** e **um Secondary NameNode**.
2. Veja que temos também quatro **DataNodes** neste exemplo. Estes são os sistemas secundários (antigamente chamados de "escravos", ou *slaves*). Eles são os trabalhadores do sistema. Eles armazenam e recuperam blocos de dados quando são instruídos pelo **NameNode** e informam o **NameNode** sobre a presença e a saúde dos blocos de dados.
 - Os **DataNodes** recebem os dados a partir de um arquivo original que é armazenado no HDFS, sendo segmentado em blocos com tamanho predeterminado (por exemplo, 128 MB). Cada **DataNode** recebe e armazena os blocos devidos e o **NameNode** registra o conhecimento da estrutura do arquivo original.
 - O tamanho deste bloco varia para cada empresa e processo. É um processo que geralmente é realizado e monitorado pelo profissional de **engenharia de dados**.
3. Percebe também que cada um destes nodes possui um **host** (hospedeiro)? No caso, temos 6 **hosts**. Aqui, cada **host** representa um computador individual (ou nó) no *cluster*.
 - Cada **host** possui também alguns dados sendo processados e distribuídos. Eles são representados por aqueles cilindros azuis dentro de cada **host**.

O **NameNode** mantém o *namespace* (sistema que relaciona o nome dos arquivos e onde eles estão armazenados) do sistema de arquivos. Um aplicativo pode especificar o número de réplicas de um arquivo que deve ser mantido pelo HDFS. O número de cópias de um arquivo é chamado **fator de replicação** desse arquivo, informação que é armazenada pelo **NameNode**. Um usuário ou um aplicativo pode criar diretórios e armazenar arquivos dentro das pastas usando o HDFS. Também pode criar e remover arquivos, mover um arquivo de um diretório para outro ou renomear um arquivo.

Com essas informações, podemos descrever o processo para gravar um arquivo no HDFS, conforme apresentado na figura a seguir.

Figura 4. Gravação no HDFS



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

1. O programa-cliente (como um usuário mexendo no Databricks, ou uma API que algum desenvolvedor *back-end* implementou) solicita a gravação do arquivo ao *NameNode*.
2. O *NameNode* verifica os *DataNodes* funcionais e seus espaços, divide o arquivo em blocos, mapeia as replicações necessárias desses blocos e todos os *DataNodes* envolvidos e devolve a informação ao cliente.
3. O cliente, por meio de uma *Application Programming Interface* (API) de acesso do HDFS, envia os blocos do arquivo original para cada *DataNode* correspondente, conforme determinado pelo *NameNode*.
4. Após enviar os blocos, o *NameNode* atualiza seus metadados.

Trocando informações

Lembra que falamos anteriormente que geralmente trabalhamos "na nuvem"? Em outras palavras, o serviço do Hadoop é executado entre diferentes computadores usando a internet. Se você estiver trabalhando em casa, precisará de uma conexão com a internet para acessar o Hadoop da sua empresa. É aí que vale a pena explicarmos um pouco alguns conceitos da internet antes de avançarmos.

GLOSSÁRIO

TCP (*Transmission Control Protocol*)

TCP/IP

RPC (*Remote Procedure Call*)

Pipelines de Dados

TCP (*Transmission Control Protocol*)

É um dos principais protocolos da Internet. Ele se originou na implementação inicial da rede, onde complementava o Protocolo de Internet (IP). Assim, o conjunto de TCP e IP é geralmente chamado como **TCP/IP**. O TCP fornece entrega confiável, ordenada e com verificação de erros de uma sequência de octetos (*bytes*) entre aplicativos que são executados em computadores conectados em uma rede IP. Ou, em outras palavras: garante que os dados sejam completamente transferidos entre dois computadores conectados na rede ainda que existam alguns problemas temporários na conexão. O TCP é orientado à conexão, e uma conexão entre cliente e servidor é

estabelecida antes que os dados possam ser enviados. O TCP é o protocolo usado para **governar e padronizar** a forma pela qual enviamos e recebemos dados da Internet.

1 / 4

Os protocolos de comunicação HDFS rodam em camadas sobre o TCP/IP. Um cliente estabelece uma conexão com uma porta TCP configurável na máquina *NameNode*, utilizando o *ClientProtocol* com o *NameNode*. Os *DataNodes* conversam com o *NameNode* usando o protocolo *DataNode*. Uma abstração de chamada de procedimento remoto (RPC) envolve o protocolo do cliente e o protocolo *DataNode*. Por padrão, o *NameNode* nunca inicia nenhuma RPC. Em vez disso, ele responde apenas a solicitações RPC emitidas por *DataNodes* ou clientes (Hadoop, 2023).

Os *DataNodes* questionam o *NameNode* constantemente para verificar se há alguma atividade a ser feita. Assim, o *NameNode* é informado sobre quais *DataNodes* estão ativos e quanto ocupados estão. Eles também se comunicam entre si, para que possam cooperar durante as operações normais do sistema de arquivos. Os blocos de um arquivo tendem a ser armazenados em múltiplos *DataNodes* e, tendo em vista que o *NameNode* é o elemento central para o correto funcionamento do agrupamento, este pode e deve ser replicado para se proteger contra uma única falha.

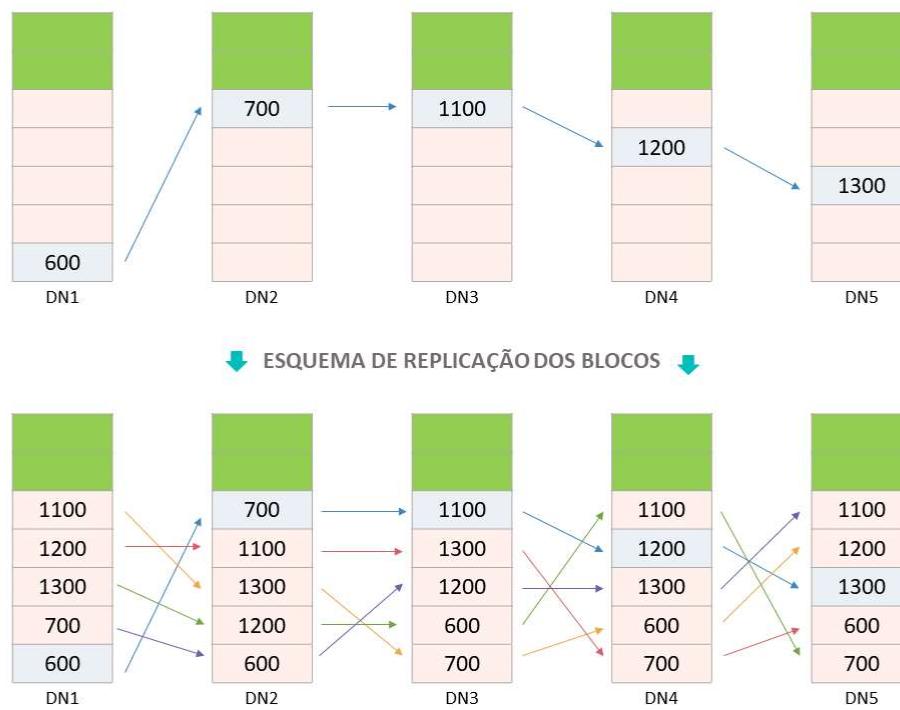
O HDFS suporta a capacidade de criar *pipelines* de dados, que consistem em uma conexão entre múltiplos nós de dados que existe para dar suporte ao movimento de dados através dos servidores. Um aplicativo-cliente grava um bloco no primeiro nó de dados do *pipeline*, que assume e encaminha os dados para o próximo *node* no *pipeline*, continuando até que todos os dados e todas as cópias destes estejam gravados no disco.

É importante entender como essa arquitetura suporta a tolerância a falhas. Basicamente, quando o arquivo original é dividido em blocos, estes são replicados e gravados nos *DataNodes*. Os blocos e suas réplicas são armazenados em diferentes

DataNodes. Por exemplo, na figura a seguir, um arquivo é dividido em cinco blocos, identificados pelos IDs 600, 700, 1100, 1200 e 1300. Cada um desses blocos é armazenado em um *DataNode* diferente.

Cada bloco contém um pedaço dos dados de um arquivo. Vamos supor que estejamos processando um arquivo de texto que contenha um livro. Neste exemplo, o bloco ID 600 poderia conter os primeiros capítulos do livro, o bloco ID 700 poderia conter alguns capítulos do meio do livro, e o bloco ID 1300 poderia conter os últimos capítulos.

Figura 5. Replicação de blocos



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

Nesse caso, a replicação é feita utilizando o **fator de replicação 5** (cinco réplicas de cada bloco) e elas são distribuídas nos *DataNodes*, de forma que, se houver uma falha em um bloco ou em um *DataNode* inteiro, o HDFS vai reconhecer o problema e o *NameNode* vai buscar os dados de qualquer outra replicação. No mercado de trabalho é comum usarmos **um fator de replicação 3** para evitar problemas de velocidade na gravação (isto é: replicamos os dados em alguns nós, mas tentamos reduzir a quantidade de nós em que replicamos para que a operação de cópia seja mais rápida).



SAIBA MAIS

Recomendamos a leitura da seção 7.5, intitulada "Big Data e data warehouses" do livro **Business Intelligence e Análise de Dados**

para Gestão do Negócio, que trata exatamente sobre o assunto desta aula!

SHARDA, Ramesh; DELEN, Dursun; TURBAN; Efraim. **Business intelligence e análise de dados para gestão do negócio.** Trad: Ronald Saraiva de Menezes. 4. ed. Porto Alegre: Bookman, 2019.

Business Intelligence e Análise de Dados para Gestão do Negócio

Comparativo entre HDFS e NTFS/Ext4

Lembra quando comentamos sobre o histórico dos sistemas de arquivo? Pois bem: lá, vimos que não são todos os lugares que usam o HDFS. Computadores em Windows usam o NTFS. Computadores que usam Linux usam o Ext4, e os computadores Apple usam o APFS.

O **NTFS** é um tipo de sistema de arquivos usado pelos sistemas da Microsoft. Ele tem recursos como *journaling*, que mantém um registro de mudanças em arquivos. Isso é útil porque permite restaurar arquivos se ocorrer um erro durante a gravação. Por outro lado, o **Ext4** é o sistema de arquivos padrão em instalações do Linux e também é usado no Android. Ele aloca dados em extensões, o que significa que pode descrever arquivos longos e fisicamente próximos em uma única entrada, reduzindo a fragmentação dos arquivos. Em contrapartida, o HDFS, usado no ecossistema Hadoop, armazena dados em blocos de tamanho fixo, garantindo que cada bloco seja completamente utilizado, o que é diferente do NTFS ou Ext4, que usam blocos mínimos. Na figura abaixo, é possível ver como os blocos são ocupados nos sistemas NTFS/Ext4 e no HDFS.

Figura 6. Comparativo entre HDFS e NTFS/Ext4



Fonte: Adaptado de Santos *et al.* (2021, p. 90).

A vantagem do HDFS é um armazenamento eficiente, mantendo blocos inteiros de dados independentemente do tamanho do arquivo. A perda com espaço pode ocorrer apenas no último bloco. O HDFS também faz a réplica dos blocos para a tolerância a falhas. No caso de leitura, para aumentar a velocidade de processamento e retornar em menor tempo, retorna a réplica que está mais perto de quem solicita o dado (com base no tempo de processamento). Outra vantagem é em relação ao tamanho dos blocos, que são bem maiores do que os blocos de sistemas não distribuídos.

Apache Sqoop

As informações armazenadas em sistemas de gerenciamento de banco de dados relacional podem ser movidas para o Hadoop. Além de mover dados em tempo real, é necessário carregar ou descarregar dados em massa; para isso, pode ser utilizada a ferramenta Sqoop (SQL-to-Hadoop), que oferece a capacidade de extrair dados a partir de armazenamentos não Hadoop, transformá-los em uma forma utilizável por esse ecossistema e carregá-los no HDFS, utilizando o processo ETL (*Extract, Transform, Load* ou *Extrair, Transformar e Carregar*) que mencionamos nas semanas anteriores.

O Sqoop apresenta quatro características-chave (Hurwitz *et al.*, 2016):

- + Importação em massa

Pode importar tabelas individuais ou bases de dados inteiras para o HDFS.

+ Entrada direta

Pode importar e mapear bases de dados SQL (relacionais) para o Hive e HBase.

+ Interação de dados

Pode gerar classes Java para interagir com os dados via programa.

+ Exportação de dados

Pode exportar dados diretamente do HDFS para uma base de dados relacional, com as especificações da base de destino a ser utilizada.

O Sqoop simplifica significativamente o processo de integração de dados, reduzindo o tempo e os esforços necessários para mover e sincronizar grandes conjuntos de dados. A importação da base de dados externa em si para o HDFS é feita por uma tarefa MapReduce criada pelo Sqoop.

| Conclusão

Nesta unidade, abordamos o sistema de arquivos distribuídos do Hadoop, conhecido como HDFS, e explicamos sua arquitetura, composta pelo *NameNode* e pelos *DataNodes*. Exploramos como o processo de distribuição dos blocos de dados, resultantes da divisão de arquivos, ocorre nos *DataNodes*, e como suas réplicas permitem a tolerância a falhas do HDFS.

Além disso, ao compreender a arquitetura, examinamos o processo de gravação de dados, utilizando comandos baseados em POSIX, com estruturas similares às do Linux. Isso proporcionou uma visão mais detalhada de como o HDFS, do ecossistema Hadoop, realiza a gravação de arquivos (dados) em massa, sendo amplamente utilizado em soluções de *big data*.

| Referências bibliográficas

BASSO, D. E. **Big data**. Curitiba: Contentus, 2020.

HADOOP. **HDFS architecture guide**. Disponível em:

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. Acesso em: 18 jul. 2023.

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

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.

SANTOS, R. R. et al. **Fundamentos de big data**. Porto Alegre: SAGAH, 2021.

