Armazenamento em Nuvem

Muitos serviços em nuvem que usamos são acima de tudo serviços de dados – Dropbox, OneDrive, Google Docs ou, menos diretamente, Instagram, Facebook e NetFlix, que armazenam grande número de dados mutlimídia. Eles excutam na nuvem que, armazena e fornece métodos de acessar, armazenar e compartilhar o conteúdo multimídia. Cada serviço de armazenamento é otimizado de diferentes maneiras para atender os diferentes requisitos de velocidade, escalabilidade, confiabilidade e consistência que cada aplicação requer.

Para dar um exemplo, o acesso a um filme em uma plataforma de *streaming* requer uma grande velocidade de acesso, mas pode ignorar a perda de alguns dados como um *frame* do filme que você está assistindo que chega com atraso excessivo (até certo ponto é mais aceitável uma imagem de menos qualidade que ignora o frame em atraso que a imagem congelar e aguarda a chegada de todos os frames). Já um sistema para armazenamento de arquivos geral, pode ter um tempo de acesso menor aos dados, mas é imprescindível que todos os dados sejam transferidos corretamente evitando-se o armazenamento e recuperação de dados corrompidos. E ainda é muito diferente os requisitos de dados de transações, como pagamentos e vendas, que trafegam pouquíssimos dados, mas requerem uma garantia de consistência muito grande e grande número de acessos. Você ainda pode pensar em muitos outros tipos de dados com requisitos diferentes. Para isso os serviços em nuvem costumam oferecer grande número de alternativas de serviços de armazenamento, não apenas os sistemas de arquivo mais comuns, mas também serviços mais especializados, como armazenamento de objetos, bancos de dados relacionais e NoSQL etc. fornecendo ainda soluções altamente escaláveis (de MBs até centenas de Terabytes de dados). Você vai aprender aqui alguns dos tipos de sistemas de armazenamento mais comuns na nuvem.

Modelos de Armazenamento

São muitos os modelos de armazenamento em nuvem, então vamos nos concentrar em alguns modelos principais, como os sistemas de arquivos gerais, os de armazenamento de objetos, os modelos de bancos de dados e sistemas de arquivos distribuídos para grandes volumes da dados.

Sistemas de arquivos

Você certamente está familiarizado com sistemas de arquivos de sistemas operacionais, como os do Windows ou Linux. Neles os dados são organizados em uma árvore de diretórios ou pastas. Este modelo é uma abstração de armazenamento de dados extremamente intuitiva e útil, e a API padrão para a versão derivada do Unix do sistema de arquivos é chamada de Interface de Sistema Operacional Portable (POSIX), sendo este o sistema que você usa no seu Windows, Linux ou mesmo Apple, podendo por linhas de comando, interfaces gráficas ou APIs, criar, ler e manipular arquivos e pastas. Esse modelo permite o acesso direto por programas na maior parte das linguagens (Python, Java, C++, R etc.) e fornece um mecanismo direto para representar relacionamentos hierárquicos dos dados (os diretórios). **Google Drive** e **DropBox** são exemplos de implementação desses sistemas em nuvem.

Armazenamento de objetos

Um armazenamento de objetos, ou ***blobs*** (*binary large objects*, como arquivos de imagens, som e multimídia), simplificam o modelo do sistema de arquivos eliminando a hierarquia dos objetos e proibindo atualizações dos objetos após terem sido criados. Diferentemente de arquivos texto como documentos, programas e páginas web, em geral você não atualiza uma imagem ou um vídeo, mas substitui o arquivo binário (o *blob*) inteiro, o que justifica essa simplificação nos sistemas de objetos. Isso também traz outras vantagens como garantia de autenticidade e, em muitos casos, sistemas de objetos são a solução preferencial para o armazenamento de documentos como contratos, documentos de identificação de pessoas, documentos financeiros e de pagamentos etc.

Cada objeto possui um identificador único e, em geral, as soluções permitem serem associados vários metadados ao objeto. No caso de uma imagem fotográfica por exemplo, você pode pensar dados que informam o local da foto, o tipo de resolução, data e hora, etc. Uma vez carregados os objetos não podem ser modificados podem ser apenas excluídos e alguns sistemas dão suporte a controle de versão dos objetos.

Essas características trazem dão aos sistemas de armazenamento de objetos vantagens de simplicidade, desempenho e confiabilidade para lidar com dados tipo *blob* quando comparados com os sistemas de arquivos tradicionais, mas fornecem pouco suporte para pesquisa dos documentos sendo necessário que o usuário crie e conheça corretamente o esquema de identificação dos arquivos.

O **S3**, ou *Simple Storage Service*, da AWS ou o **Blob Storage Service** da Microsoft Azure são exemplos de serviços em nuvem para o armazenamento de objetos.

Bancos de Dados Relacionais e NoSQL

Você já deve estar familiarizado com bancos de dados e SQL. Bancos de dados relacionais são uma coleção estruturada de dados e implementam entidades (as tabelas, como tabelas Clientes, Fornecedores e Produtos) e seus relacionamentos (Cliente-Produto, Produto-Fornecedores etc.). Um sistema gerenciador de banco de dados (SGBD) é um software que armazena, recupera e gerencia esses dados de maneira segura e eficiente. Seus componentes dão suporte a uma série de facilidades:

i) SQL, uma linguagem de consulta aos dados;
ii) Criação de modelos de dados (DDL, que permite o usuário definir tabelas e relacionamentos);
iii) Suporte a transações e recuperação de falhas (para garantir uma execução confiável apesar de falhas do sistema, o que inclui os mecanismos de *commit* e *rollback* de atualização com a SQL);
iv) Suporte a concorrência, isto é, os dados podem ser acessados e atualizados concorrentemente por vários usuários simultâneos. O

gerenciamento do acesso e das atualizações concorrentes é feito pelo SGBD para garantia da consistência dos dados.

v) Indexação dos dados e otimização do acesso, que garante uma alta eficiência para o processamento transacional com múltiplos usuários e grandes volumes de dados.

vi) E diversos utilitários para gerenciamento do sistema e seus bancos de dados (serviços de import/export de dados, backup e recuperação etc.).

Essas características tornam os SGBD a solução preferencial para o armazenamento de dados transacionais (com atualizações constantes) com múltiplos usuários, e a maior parte dos sistemas transacionais que conhecemos empregam poderosos SGBD desse tipo como os sistemas de cartão de crédito, de transações bancárias, de comércio eletrônico e vendas, sistemas ERP e de faturamento, de recursos humanos e folha de pagamentos etc.

Dada sua importância na construção de aplicações, os principais SGBD, tanto comerciais/proprietários como **Oracle, MS SQL**, e open source como **MySQL, PostGreSQL**, e **MariaDB,** são encontrados como serviço na maior parte dos provedores, havendo ainda versões próprias de cada fornecedor como o **RDS AWS** ou o **Spanner** da Google.

Uma importante característica dos SGBD relacionais é o suporte para as propriedades **ACID**. São essas propriedades que permitem a execução de transações (acessos e atualizações) concorrentes de múltiplos usuários garantindo a consistência e integridade dos dados. Para implementar essas propriedades os bancos de dados relacionais fazem uso de uma série de mecanismos *lock* de registros, de salvamento dos dados de transações correntes para *commit* e *rollback*, e *log* com o histórico de transações. Esses mecanismos, mesmo que implementados de modo eficiente, proporcional um grande ônus na execução das operações, mas são essenciais para garantia dos dados e são em boa parte a razão do sucesso desses gerenciadores. É importante que você note que nem sistemas de arquivos tradicionais e nem sistemas de armazenamento de objetos permitem a atualização de dados de forma concorrente, e muito menos com a eficiência dos gerenciadores relacionais.

Propriedades ACID

Atomicidade: Toda a transação é bem-sucedida ou falha

Consistência: O acesso aos nunca retorna um estado inválido ou inconsistente

Isolamento: Transações concorrentes não interferem entre si, e executam do mesmo que estivessem isoladas

Durabilidade: Uma vez concluída uma transação seus resultados são preservados, independente de falhas futuras do sistema

O SGBD relacionais são ideais para processamento transacional de bem dados estruturados (tabelas), mas seus *pesados* mecanismos para o controle transacional e garantia das propriedades ACID, torna-os ineficientes para dados pouco estruturados e dados multimídia e grandes volumes de dados em formato texto, ou quando é necessário tratar grandes volumes de dados predominante para inserção e consulta. Você pode pensar em aplicações como as de portais de multimídia como Spotify e NetFlix, ou de redes sociais como o LinkedIn, Instagram e FaceBook.

4 V's Big Data

V: Volume de dados

V: Velocidade, do Fluxo de Dados

V: Variedade, dados não estruturados

V: Veracidade, sobre os dados *poderem* ser confiáveis ou não

Assim, para algumas classes de aplicativos se tornaram muito comuns os bancos de dados NoSQL. Esses sistemas, privilegiam a capacidade de escalar para grandes quantidades de dados, e um grande número de usuários,

característicos dos dados *Big Data* que ser tornaram comuns para muitas aplicações modernas como as redes sociais, aplicações internet de *streaming*, apps como Waze e Uber, e que hoje já alcançam algumas aplicações mais tradicionais como os grandes portais e apps de vendas, e aplicações de bancos digitais. Por conta disso esses bancos de dados se tornaram muito populares na nuvem.

Uma forma comum empregada por esses bancos de dados é organizarem os dados em uma estrutura de **chave-valor**. Nessa estrutura, um número arbitrário de registros e dados de diferentes formatos (valores) é associado a uma única chave. Se você conhece formatos **JSON (JavaScript Object Notation)** ou a estrutura de dicionários em Python, esses bancos de dados organizam as informações de modo bastante semelhante a essas estruturas, adicionando capacidades e APIs de recuperação (consulta aos dados), atualização etc.

JSON ou Python Dictionary Style

```
{ nome: 'Adriana', idade: 36, amigos: [ 'Daniel',
'Carol', ... ], bio: '...long text', foto: ... }
```

O termo NoSQL significa '***not Only SQL***' e tem origem no fato de que esses bancos de dados não darem suporte pleno à álgebra relacional (implementada pela SQL), embora possam implementar boa parte dela, e incluírem outras funcionalidades para os tipos de dados que suportam. Esses bancos de dados sempre fornecem uma linguagem de consulta, mas só mais recentemente a maioria vem implementando uma interface SQL por ser uma linguagem ampla, padrão e facilmente empregada, e você pode encontrar a maior parte da SQL disponível. Você entenderá melhor essa limitação se compreender que realmente não faz sentido você fazer um select para recuperar na *saída da select* uma foto ou o texto da biografia de uma pessoa em uma rede social, fazer um *group by* onde o argumento é um documento, ou o *join* dados de páginas do FaceBook! Assim, essa limitação não é exatamente dos bancos de dados, mas

está mais associada ao tipo de dados que eles tratam, e bancos de dados relacionais modernos têm a mesma limitação ao tratar esses tipos de dado.

Dentre as funcionalidades que incluem, bancos de dados NoSQL permitem a ingestão rápida de grandes quantidades de dados não estruturados flexibilizando as propriedades ACID. Log de transações não são criados e podem criar dados *temporariamente* inconsistentes. Mas isso não parece ser um grande problema se você imaginar que você, e milhares de outros usuários, estão postando dados na sua rede social ou fotos no Instagram.

Outra facilidade que dá muito mais flexibilidade aos bancos de dados NoSQL é que dados arbitrários podem ser armazenados sem modificações prévias no esquema de banco de dados, como ocorre em um banco relacional (create table, alter table etc.). Novas colunas, campos de dados, podem ser introduzidos ao longo do tempo pela própria aplicação conforme os dados evoluem. E, tendo muitos desses gerenciadores surgido já em ambientes de nuvem, eles costumam ser distribuídos em vários servidores e suportarem replicação dos dados em múltiplos data centers. Em consequência disso, eles frequentemente substituem a consistência dos bancos de dados relacionais (A**C**ID) pela *consistência eventual.* Certamente você não iria querer isso em dados de uma operação de cartão de crédito ou transações bancárias, mas essa consistência eventual, não é um problema se você imaginar que está fazendo um post em uma rede social, ou o upload de um vídeo no TikTok, podendo haver *locais* em que o post ou a imagem serão atualizados alguns minutos mais tarde, e que atualizações síncronas iriam acarretar uma perda grande da velocidade das atualizações.

Esse relaxamento das propriedades ACID pode ser melhor entendido com o teorema CAP que você verá a seguir, e ele também ajudará a você a compreender algumas características de sistemas de armazenamento distribuídos que veremos em seguida.

Antes, veja abaixo alguns dos sistemas de armazenamento mais comuns empregados em nuvem e alternativas aos modelos NoSQL Chave-Valor.

Model	Amazon	Google	Azure
Files	Elastic File System (EFS), Elastic Block Store (EBS)	Google Cloud attached file system	Azure File Storage
Objects	Simple Storage Service (S3)	Cloud Storage	Blob Storage Service
Relational	Relational Data Service (RDS), Aurora	Cloud SQL, Spanner	Azure SQL
NoSQL	DynamoDB, HBase	Cloud Datastore, Bigtable	Azure Tables, HBase
Graph	Titan	Cayley	Graph Engine
Warehouse analytics	Redshift	BigQuery	Data Lake

Além da estrutura Chave-Valor, você ainda pode encontrar bancos de dados NoSQL baseados em outras estruturas de armazenamento, e alguns bastante populares como: armazenamento baseado em documento (**CouchDB. MongoDB**); armazenamento baseado em colunas (**HBase, Cassandra**); e ainda os baseados em grafos (**Neo4J**).

Teorema de CAP

O teorema de CAP afirma que em um sistema distribuído as propriedades de Consistência, Disponibilidade e Tolerância ao Particionamento (CAP = Consistency, Availability, Partition Tolerance) nunca podem ser obtidas simultaneamente. É necessário, portanto, escolher entre duas dessas propriedades, nunca conseguindo alcançar as três ao mesmo tempo.

CAP

Consistência: O dado gravado (atualizado) é disponível de forma imediata para uso, isto é, todo cliente terá a mesma visão dos dados.

Disponibilidade: É assegurado que o dado permanece ativo, mesmo quando da ocorrência de falhas e nas atualizações de SW/HW.

Tolerância ao Particionamento: Capacidade de um sistema continuar operando mesmo depois de uma falha na rede ou falhas parciais, garantindo que as operações serão completadas, mesmo que componentes individuais não estejam disponíveis.

Assim, no uso de sistemas distribuídos, é necessário se optar por ter apenas duas das características de consistência forte, alta disponibilidade ou tolerância ao particionamento.

- **Sistemas CA** (Consistência e Disponibilidade): Neste caso o sistema não seberá lidar com falhas parciais, como falha de uma partição e, caso ocorra, todo o sistema estará indisponível.
- **Sistemas CP** (Consistência e Particionamento): Aqui não há garantia de que os dados estarão sempre disponíveis, havendo interrupções na disponibilidade dos dados para os usuários.
- **Sistemas AP** (Disponibilidade e Particionamento): Nesses sistemas sacrifica-se a consistência pela disponibilidade dos dados, garantida ainda na ocorrência de falhas parciais. Neste caso os sistemas aceitam escritas e atualização mas irão sincronizar os dados depois.

O mais comum para aplicações em nuvem é encontrarmos sistemas do tipo CA, como o MongoDB, ou AP, quando a consistência de dados não é tão importante (sistema que envolvem ingestão de grandes volumes de dados) buscando deixar as aplicações e seus dados 100% (ou a maior parte do tempo) ativas.

Hadoop e MapReduce

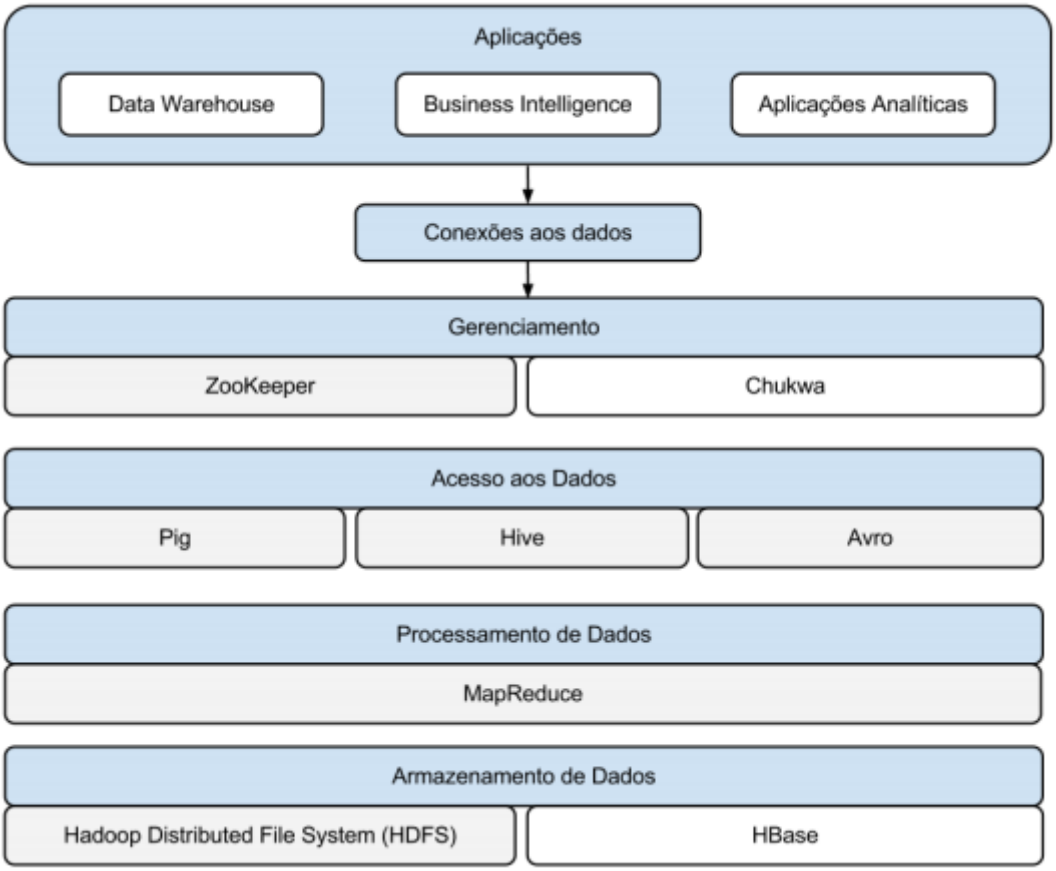
A popularização da Internet, a transformação digital e o surgimento de novas aplicações (apps, redes sociais, plataformas de streaming e inúmeros serviços como Whatsapp, Uber, Airbnb, Waze etc.) levaram ao surgimento de uma quantidade de dados nas últimas décadas sem precedente. Por simplicidade colocamos todos esses dados em um conjunto único que denominamos *Big Data* e que caracterizamos aqui, pelos 4 V's (volume, velocidade, variedade e veracidade). Para o processamento desses grandes volumes de dados com o desempenho e disponibilidade exigidos os sistemas tradicionais de armazenamento são em geral insuficientes. A solução é o uso de um sistema de arquivos distribuído em que os dados são distribuídos em inúmeros nodes (servidores) para processamento paralelo.

A solução mais amplamente empregada e em uso pelos maiores ambientes de nuvem é o **Hadoop Distributed File System (HDFS)**, um sistema de arquivos distribuído integrado ao arcabouço do projeto open source Apache Hadoop. O HDFS tem origem em uma solução da Google, o GFS, com o qual a Google buscou aperfeiçoar o seu mecanismo de busca para dar conta do processamento de **todo** o conteúdo da Web.

O HDFS é um sistema de código aberto, **implementado em Java**, e que oferece suporte ao armazenamento e ao processamento de **grandes volumes de dados em um agrupamento de computadores heterogêneos de baixo custo** (você pode entender aqui processadores commodities de x86). Isso é muito importante para a escalabilidade e o baixo custo exigido das soluções em nuvem. A quantidade de dados pode chegar petabytes com um desempenho aceitável, o que não é atingido por qualquer sistema de arquivos tradicional. Para alta disponibilidade dos dados, uma característica importante é que o sistema emprega **múltiplas réplicas dos dados** (é um sistema AP que você viu acima).

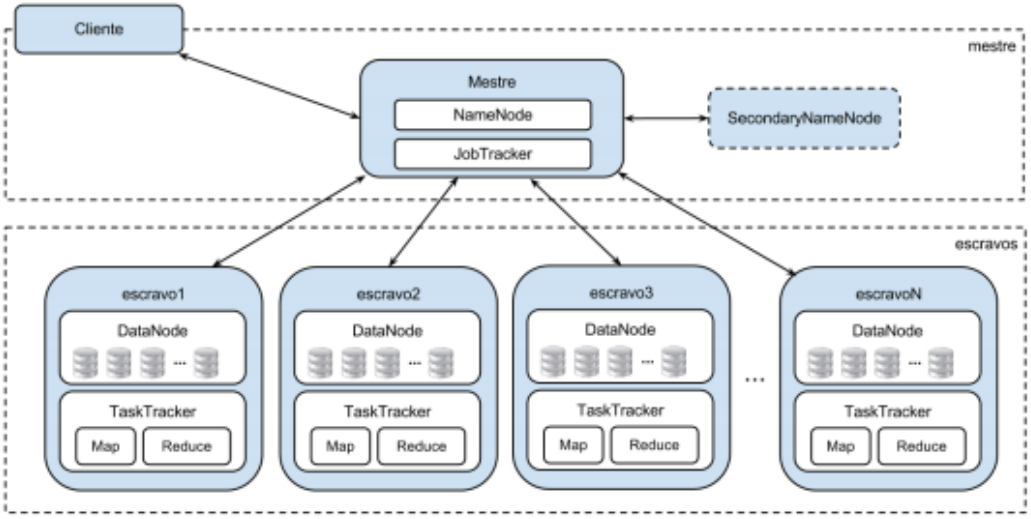
Para você ter uma ideia da importância desse sistema algumas das maiores empresas de internet e mesmo tradicionais fazem uso deles em sistemas de nuvem: IBM, Oracle, Adobe, Twitter, Yahoo, LinkedIn, Last.FM, NYT, FaceBook, e-Bay.

No arcabouço do projeto Apache Hadoop há também ferramentas importante que operam em conjunto com o HDFS. Um dos principais é o **MapReduce**, que permite, integrado ao HDFS, o processamento distribuído pelos nodes do sistema de armazenamento. Existindo ainda outros subprojetos bastante utilizados como o **Avro** (um esquema de seriação de dados baseado em *schemas* para armazenamento compactado no HDFS), o **Hbase** (que implementa um 'banco de dados' sobre o HDFS e é a versão de código aberto do banco de dados BigTable da Google), o **Hive** (desenvolvido no FaceBook e que permite um acesso 'SQL Like' a dados do HDFS), o **Pig** (uma linguagem de alto nível para fluxo de dados e transformações – ETL) e o **ZooKeeper** (criado pelo Yahoo! para coordenação de aplicações distribuídas de alto desempenho). E você pode ver na figura abaixo como esses projetos encontram-se relacionados dentro do projeto Apache Hadoop.



Componentes do Hadoop

A execução típica de uma aplicação Hadoop os dados encontram-se distribuídos em diversos DataNodes. Os DataNotes (para que você tenha uma ideia, podem ter configurações com os dados distribuídos em 128, 256 ou mais nodes!) contêm partições de dados e a ideia é que o processamento é levado para esses nodes e executam em paralelo. O HDFS é a parte responsável pelo gerenciamento do armazenamento distribuído dos dados, enquanto o MapReduce, é responsável pelo processamento distribuído e consolidação desses resultados. Essa é uma visão bastante simplista de como uma aplicação Hadoop funciona, mas bastante útil para uma visão geral e para que você possa prosseguir no entendimento desta solução.



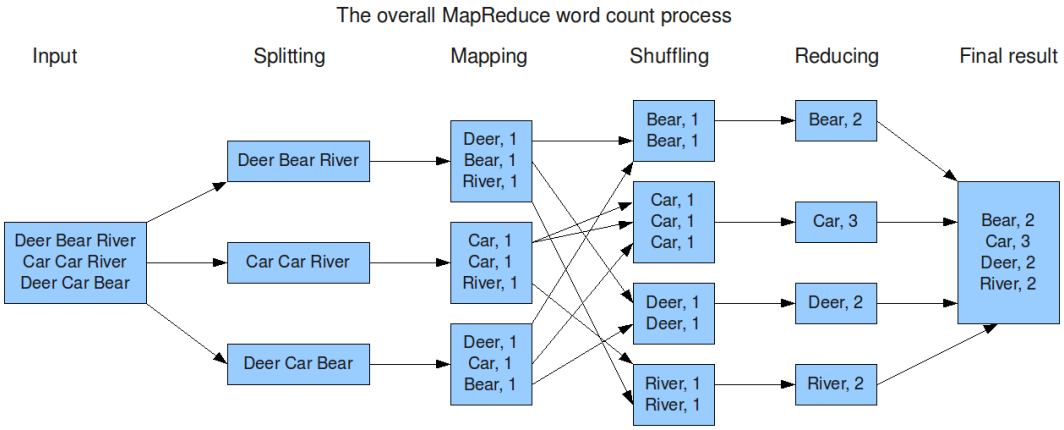
Uma aplicação típica em um conjunto de nodes emprega cinco processos: **NameNode, DataNode, SecondaryNameNode, JobTracker** e **TaskTracker**, sendo os três primeiros do modelo de execução do MapReduce e os dois últimos do sistema de arquivo HDFS. Os componentes NameNode, JobTracker e SecondaryNameNode são únicos para toda a aplicação, como você pode ver na figura acima, enquanto que o DataNode e JobTracker são processos instanciados para cada máquina. De forma resumida temos:

- **NameNode**: gerencia os arquivos armazenados no HDFS. Ele mapeia a localização dos arquivos e a distribuição dos arquivos em blocos, incluindo as réplicas de dados. Você pode pensar que o mesmo é feito

pelo sistema de arquivos do seu computador, mas sem incluir réplicas dos dados ou a distribuição em diferentes nodes.

- **DataNode**: enquanto o NameNode gerencia os blocos de arquivos, os DataNodes são responsáveis pelo efetivamente pelo armazenamento/gravaçã dos dados.
- **JobTracker e TaskTracker**: Coordenam a execução das tarefas do MapReduce.
- **SecondaryNameNode**: utilizado para auxiliar o NameNode a manter seu serviço, sendo uma alternativa de recuperação no caso de uma falha do NameNode.

O sistema de réplicas e de tolerância a falhas, tem por objetivo não só manter a disponibilidade dos dados em caso de falhas físicas ou de software dos componentes, mas comumente é empregado para permitir a manutenção de software e hardware sem a indisponibilidade dos serviços, uma característica essencial para sistemas em nuvem.



A imagem acima como um sistema HPDF/MapReduce pode ser empregado para uma aplicação típica da internet, e que pode tirar proveito da distribuição dos dados e do paralelismo da arquitetura HDFS/MapReduce, é uma aplicação de contar termos em documentos (empregado para indexação e busca de termos em páginas da internet). Acima as páginas estão representadas pelas linhas à esquerda, e você pode pensar em uma página somente com o conteúdo ‘Deer Bear River’, outra com ‘Car Car River’ etc. O dados são distribuídos (splitting) e

o processamento, de contagem de palavras ou termos, efetuado em paralelo (**mapping**). Em seguida os resultados são organizados (shuffling) e resumidos (**reducing**) para se obter o resultado final.

Sumário

Muita coisa sobre armazenamento na nuvem não? É que de fato muito do que fazemos na nuvem é o processamento de informações, e essas estão representadas pelos dados armazenados. O armazenamento é então uma parte essencial dos sistemas em nuvem. Em uma situação típica, você não tem o que processar se não tiver dados. Além disso a aceleração da internet e transformação digital trouxe uma enormidade de dados novos e mais complexos para os quais a computação em nuvem buscou boa parte das soluções que você estudou aqui. É importante também que você entenda que esses sistemas são, muitos deles, bastante recentes e esta é uma área em constante evolução. Os bancos de dados relacionais que conhecemos hoje são muito diferentes dos bancos de dados de 20 anos atrás e, certamente, serão diferentes dos bancos de dados que teremos daqui 20 ou 10 anos, e o mesmo deve acontecer com os demais sistemas de armazenamento.

Referências

Goldman, A., et al. Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. In CSBC 2012 - XXXII Congresso da Sociedade Brasileira de Computação. Disponível em: http://www2.sbc.org.br/csbc2012/anais_csbc/eventos/jai/artigos/JAI%20-%20Cap%203%20Apache%20Hadoop%20conceitos%20teoricos%20e%20praticos,%20evolucao%20e%20novas%20possibilidades.pdf Acesso: 10.03.2021.

Ian Foster and Dennis B. Gannon. **Cloud Computing for Science and Engineering** (1st. ed.), 2017. The MIT Press. Alternativamente Disponível em: https://cloud4scieng.org/ Acesso: 10.03.2021.

Lapusan, T. **Hadoop MapReduce deep diving and tuning.** Disponível em: https://www.todaysoftmag.com/article/1358/hadoop-mapreduce-deep-diving-and-tuning. Acesso: 10.03.2021.