FUNDAMENTOS DE BIG DATA

Laerte de Marque

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Hadoop Distributed Filesystem

OBJETIVOS DE APRENDIZAGEM

- Descrever os principais desafios de armazenamento em cenários de *big data*.
- Reconhecer o Hadoop Distributed FileSystem (HDFS), assim como sua arquitetura, seus componentes e sua linha do tempo.
- Apontar o funcionamento do HDFS sobre o Hadoop, assim como sua tolerância a falhas.

Introdução

Os desafios do armazenamento de dados são encarados por pequenas, médias e grandes empresas. Afinal, no momento em que a empresa começa a caminhar em direção ao armazenamento em massa, começam a surgir problemas que devem ser enfrentados. Uma das ferramentas capazes de ajudar nesse cenário é o HDFS, que trabalha perfeitamente com grandes massas de dados.

Neste capítulo, você vai conhecer os desafios do armazenamento no contexto do *big data*. Você também vai ver por que o HDFS é uma solução importante nesse ecossistema. Além disso, vai estudar a arquitetura, os componentes e a linha do tempo do HDFS. Por fim, vai conhecer o sistema de tolerância a falhas desse sistema.

Principais desafios de armazenamento em cenários de *big data*

Como você sabe, no mercado contemporâneo, as informações tornaram-se vitais para qualquer empresa. No geral, quando uma empresa vislumbra a

possibilidade de armazenar seus dados para uso futuro, ela começa a enfrentar alguns desafios. Embora muitas pessoas acreditem que o armazenamento de dados seja uma tarefa simples, na prática ele envolve custos, planejamento e diversos outros fatores. Afinal, o que se deseja não é apenas armazenar, mas também obter informações claras e úteis com agilidade. Como você pode imaginar, não faz sentido guardar dados se eles não estiverem disponíveis em tempo hábil quando forem necessários.

Em síntese, portanto, armazenar dados serve para extrair deles conhecimentos que possam beneficiar o negócio da empresa. Hoje, o armazenamento de dados segue regulamentações baseadas em leis (trabalhistas, fiscais, etc.). Os dados e informações vêm de diferentes caminhos; para garimpá-los, muitas vezes é necessário um processo manual e caro (AGGARWAL, 2015).

Quando as empresas começam a analisar o que precisam armazenar, costumam perceber que possuem dados vindos de muitas fontes. Os dados podem estar nos seguintes formatos:

- estruturados, organizados, como os presentes em um banco de dados;
- não estruturados, como os dados de programas de mensagens (por exemplo, WhatsApp e Telegram);
- semiestruturados, provenientes de algumas linguagens de marcação, como HyperText Markup Language (HTML).

Como você pode notar, a situação não é simples. Muitos dados não estruturados vêm de conversas telefônicas ou atendimentos simples por *e-mail*. A dificuldade de armazenamento desses dados (*big data*) começa a ser enfrentada no momento em que a empresa decide que toda ou grande parte da informação é importante para a inteligência do seu negócio.

Quando se fala em *big data*, existem conceitos internacionalmente conhecidos: volume, velocidade, variedade, veracidade e valor (5 Vs) (KHAN; UDDIN; GUPTA, 2014). É importante compreender cada um desses conceitos para superar os desafios de armazenamento que se impõem aos cenários contemporâneos. A seguir, você vai conhecer melhor os 5 Vs.

Volume

As empresas dependem cada vez mais dos dados que recebem e dos dados que produzem para manter suas obrigações legais, trabalhistas, fiscais, etc. Além disso, o volume de dados aumentou significativamente nas últimas décadas, e muitas empresas podem dobrar seu volume a cada ano que passa.

Para compreender melhor, considere as *Random Access Memories* (RAMs) de um computador. No início dos anos 1980, um padrão de computadores conhecido como MSX conquistou muitas gerações usando um processador Z-80; ele tinha em torno de 32 kB de memória disponível ao usuário (dependendo da versão). Hoje, as memórias de computadores ultrapassam os gigabytes e é comum um *notebook* possuir 4 GB de memória.

Como você sabe, as RAMs dos computadores se modificaram muito ao longo das últimas décadas: houve um aumento de aproximadamente 1,58 vez por ano, considerando uma média de 25 anos. Agora imagine esse mesmo incremento sendo aplicado aos dados armazenados por uma empresa. É claro que não existe um padrão comum de aumento de dados, pois isso depende muito do perfil da empresa, mas os dados das organizações vêm se multiplicando. Consequentemente, o volume de dados aumenta, o que representa um dos desafios do armazenamento. O armazenamento físico ou na nuvem tem um custo de alocação física e também um custo de gerenciamento, e é inegável que toda empresa tem mais dados hoje do que alguns anos atrás.

Variedade dos dados

Os dados podem vir de diversas fontes, desde um simples cartão de ponto de um funcionário até mensagens nas redes sociais. A questão da variedade é um desafio, pois é necessário capturar diversos mecanismos e controles diferentes, alguns padronizados e outros não. Nesse contexto, também é preciso decidir que dados são relevantes, o que pode ser crucial para futuras necessidades ligadas a dados e informações. Para tomar esse tipo de decisão, é necessária uma grande análise conduzida por especialistas, e o custo pode mudar conforme o que se deseja armazenar. Para algumas empresas, talvez seja dispensável armazenar conversas telefônicas, mas não para outras. Em suma, essa é uma das fases mais complexas pois envolve o entendimento do negócio e implica a seleção de dados que podem ajudar a empresa no futuro.

Velocidade

A velocidade é uma vantagem competitiva para a empresa. Se dados atualizados não são acessíveis em tempo real, as decisões podem ser prejudicadas, e os dados podem não servir mais. Assim, imensas bases de dados de pouco adiantam se não estão disponíveis em tempo real para cumprirem seu objetivo, que é orientar a tomada de decisão.

Para compreender melhor, considere uma compra na internet: se o *site* é muito lento, ou se a sua conexão cai e você perde o seu carrinho de compras, a sua tendência é simplesmente abandonar a tarefa e tentar em outro momento, não é? No mundo dos negócios, a velocidade é mais crucial ainda: se uma empresa tem um *big data* e não consegue acessá-lo com agilidade para recuperar dados e comparar diversas operações, ela pode ficar atrás da concorrência ou tomar uma decisão precipitada por falta de tempo.

Veracidade

De nada adianta um imenso ecossistema de *big data* se você não puder ter certeza de que os dados provenientes dele estão corretos. Aqui, o desafio encontra-se na análise e no algoritmo usado para a extração dos dados; a ideia é que ambos consigam suprir as necessidades da empresa. Sem ter certeza sobre a veracidade dos dados armazenados e sobre o seu valor real, não é possível extrair informações úteis. Todo o projeto é comprometido em caso de falta de veracidade: se as informações que uma organização armazenou com muito custo durante uma intensa fase de análise tiverem apenas um bit alterado, podem ser estragadas e comprometidas.

Valor

Não é possível dizer que um ou outro desafio seja mais importante, mas com certeza os dados devem gerar valor para a empresa. O valor se associa a todos os custos implicados em armazenar, gerenciar e extrair os dados, mas, em especial, ele tem a ver com a importância dos dados para a empresa (valor gerado). Logo na montagem de um *big data*, os custos se mostram relevantes: há custos de armazenamento, de consultoria, de treinamento e de análise dos dados e das necessidades atuais e futuras da empresa. No final, quando tudo estiver funcionando a contento e da maneira esperada, obviamente a empresa deve enxergar valor nos dados que armazena e transformá-los em bons negócios.

Arquitetura, componentes e linha do tempo do HDFS

A partir de agora, você vai conhecer melhor a ferramenta HDFS e verificar como ela pode ser útil no enfrentamento dos desafios de armazenamento de *big data*. Além disso, você vai conhecer os componentes e a arquitetura do HDFS.

O HDFS possui muitas vantagens: robustez, escalabilidade e simplicidade (GOLDMAN *et al.*, 2012). O HDFS é um projeto *open source*, e isso significa que ele permite que você tenha acesso aos códigos-fonte dos programas. Os projetos *open source* têm várias vantagens; entre elas, você pode considerar a seguinte: desenvolvedores do mundo todo, ao "enxergarem" o código, são capazes de realizar e apontar melhorias, assim o *software* não permanece estático, mas está em aperfeiçoamento contínuo, o que é uma das propostas da análise e do desenvolvimento de *software* em geral. Por outro lado, alguns profissionais defendem que, se o *software* pode ser visto por todos, é mais fácil encontrar formas de burlá-lo em caso de más intenções. A discussão é polêmica, mas a verdade é que o *open source* contribui para acelerar e melhorar muitos projetos, pois a união de muitos profissionais acelera o avanço da ferramenta e a torna mais eficiente. O HDFS é mantido pela Apache Software Fundation (ASF), que é a fundação do famoso apache; ele é aperfeiçoado por toda a sua comunidade.

O HDFS surgiu de um mecanismo de busca totalmente imerso na configuração *open source* criado por Mike Cafarella e Doug Cutting. A ideia básica era viabilizar resultados mais rápidos distribuindo o processamento entre vários computadores. Esse mecanismo de busca era chamado "Nutch" e foi dividido; a parte baseada em computação distribuída/processamento distribuído se tornou o Hadoop, que teve seu nome inspirado no brinquedo do filho de Doug, um elefante. Inclusive, a logomarca do Hadoop deriva da imagem desse animal, que também representa a grandiosidade (grande capacidade). Hoje, o HDFS é usado por diversas empresas em ambiente local ou em serviços hospedados na nuvem.

A vantagem do uso do HDFS é que ele permite armazenar, analisar e gerenciar grandes massas de dados de forma muito rápida, com muita confiança. Esse é um fator decisivo para as empresas na adoção do HDFS, pois é possível fazer um *cluster* Hadoop em servidores simples (e até mesmo em máquinas comuns) que geram custos muito baixos. Além disso, sua estrutura e seu funcionamento provaram ser confiáveis, de modo que o Hadoop é usado por grandes *players*, como Amazon e Google.

A seguir, veja a linha do tempo simplificada do projeto.

- 2002: nasce o Nutch, motor de busca *open source*.
- 2003: Google publica o Google File System (GFS).
- 2004: é definido o Sistema NDFS, baseado no MapReduce do Google.
- 2006: o projeto se transforma no Hadoop.

- 2008: o Hadoop bate o recorde mundial ao ser o sistema mais rápido a ordenar 1 terabyte.
- 2009: O HDFS é consolidado.
- A partir de 2012: o Hadoop se torna uma referência e diversas versões são lançadas.

O HDFS é um projeto e seu principal objetivo, desde o início, foi ser um projeto resistente. Se no HDFS ocorrem problemas em uma entidade do *cluster* (um "nó", como é popularmente conhecido um servidor do *cluster*), os dados são redirecionados e balanceados entre os nós restantes.

Uma das vantagens da arquitetura do HDFS é permitir o armazenamento de dados em qualquer formato. O HDFS, sistema de arquivos responsável por tudo isso, controla os arquivos que são distribuídos entre os nós presentes no sistema. Os nós padrão HDFS possuem *daemons*, que são basicamente processos executados em segundo plano. Esses processos ficam residentes no processamento da máquina e servem para gerenciar toda a estrutura do HDFS. A seguir, veja quais são os processos.

- **NameNode:** é capaz de armazenar os metadados necessários para a estrutura total do sistema de arquivos. O NameNode atua como mestre da arquitetura, determinando a estrutura de arquivos armazenados no HDFS (ou metadados). Assim, cabe ao NameNode efetuar o gerenciamento dos nós e o devido monitoramento dos estados dos nós no HDFS.
- **DataNodes:** armazenam o conteúdo dos arquivos em pequenos pedaços conhecidos como "blocos". Cada bloco representa uma fração de tamanho predeterminado do arquivo original. O HDFS quebra o arquivo em blocos de tamanho fixo e armazena esses blocos, de modo distribuído, em diversos DataNodes espalhados pela rede.
- **NameNode Secundário (Secondary NameNode):** verifica o *log* interno do NameNode e os *logs* de operações. Sua principal função é armazenar um *backup* dos metadados do HDFS. Portanto, em caso de falha do NameNode, o Secondary NameNode permite a recuperação do estado atual do HDFS por meio do *backup* dos metadados.

Na Figura 1, veja uma representação visual da arquitetura do HDFS.

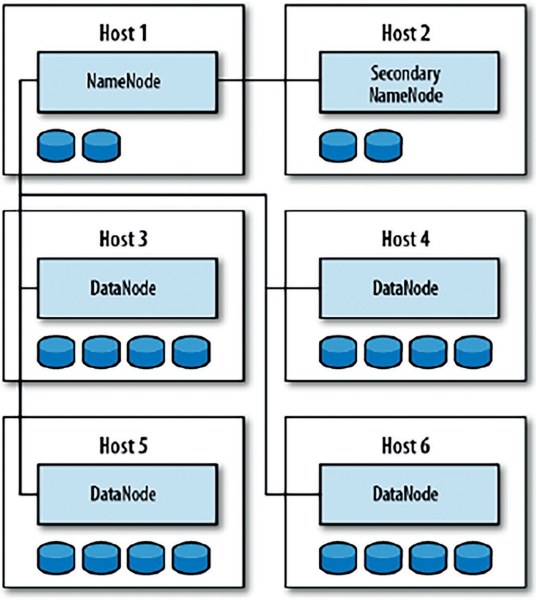

Figura 1. Arquitetura HDFS.
Fonte: Sammer (2012, p. 10).

A Figura 1 é uma imagem padrão da arquitetura do HDFS. Note que há diversos DataNodes, mas apenas um NameNode e um Secondary NameNode. É realmente assim em uma arquitetura HDFS, ou seja, existe apenas um NameNode e um Secondary NameNode por *cluster* Hadoop, porém os DataNodes podem ser milhares no *cluster*. O motivo de o número de DataNodes variar é que os DataNodes armazenam os dados.

O NameNode pode ser considerado um gestor. Ele recebe confirmações de funcionamento dos dados constantes nos DataNodes e, por meio dessas confirmações, entende a integridade e identifica os blocos de dados disponibilizados para acesso. O NameNode está preparado para ter rapidez em suas pesquisas, pois recebe informações (metadados) da RAM. Quanto mais memória existir, mais capacidade estará disponível para mais blocos de metadados, que o NameNode terá à sua disposição. Os metadados consomem RAM conforme seu uso. Segundo Sammer (2012, p. 8): "Uma estimativa aproximada é que os metadados para 1 milhão de blocos ocupam aproximadamente 1 GB de *heap*".

No HDFS, os DataNodes recebem todos os dados em blocos. A utilização do conceito de blocos permite ao HDFS prover escalabilidade. Para tanto, um arquivo original, que será armazenado no HDFS, é dividido em blocos de tamanho predeterminado (por exemplo, 128 MB), sendo que cada DataNode armazena os blocos devidos, enquanto o NameNode possui conhecimento sobre a estrutura do arquivo original. Portanto, para reconstruir o arquivo armazenado, basta o HDFS remontar o arquivo de acordo com os seus blocos.

Note que, ao dividir um arquivo original em blocos menores, o HDFS é capaz de escalar a sua capacidade de armazenamento, uma vez que diversos DataNodes podem ser utilizados para armazenamento dos blocos, sem a necessidade de armazenamento do arquivo original como um todo (são armazenadas apenas as suas frações, representadas pelos blocos). A arquitetura em blocos, além de facilitar o armazenamento e a tolerância a falhas, permite que o HDFS aproveite melhor o espaço, pois os arquivos são contíguos e não existe sobra de espaço como em um sistema comum (*slack space*). Veja o que afirma Sammer (2012, p. 8):

> O HDFS, de várias maneiras, segue o ***design*** tradicional do sistema de arquivos. Os arquivos são armazenados como blocos e existem metadados que controlam o nome do arquivo para bloquear o mapeamento, a estrutura da árvore de diretórios, as permissões e assim por diante. Isso é semelhante aos sistemas de arquivos Linux comuns, como o `ext3`.

A Figura 2, a seguir, detalha o processo de escrita de um arquivo no HDFS.

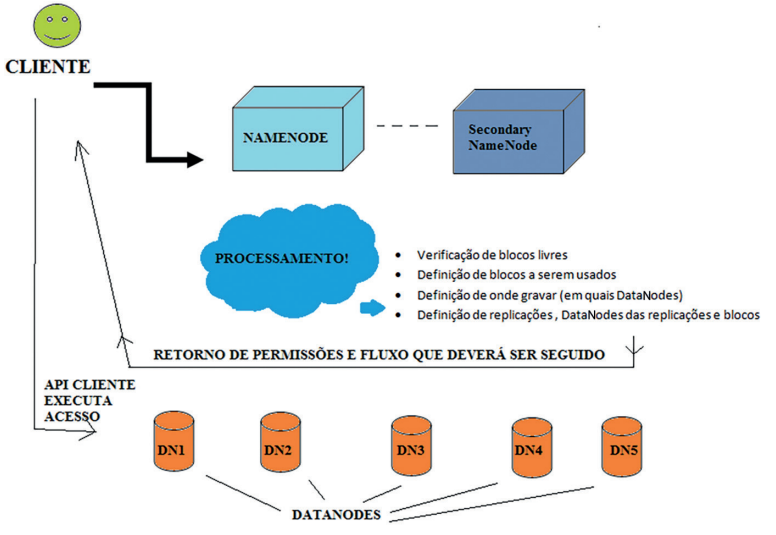

Figura 2. Gravação no HDFS.

Veja como ocorre o processo:

- o programa cliente pede acesso de gravação ao NameNode;
- o NameNode verifica, por meio de metadados, os DataNodes funcionais e seus espaços, divide os dados em blocos, mapeia esses blocos na estrutura de arquivos, mapeia as replicações necessárias desses blocos e todos os DataNodes envolvidos e devolve a informação ao cliente (tudo isso ocorre somente se o cliente tem permissão);
- o cliente, por meio de uma *Application Programming Interface* (API) de acesso do HDFS, envia as operações autorizadas aos DataNodes, ou seja, envia os blocos do arquivo original para cada DataNode correspondente, conforme determinado pelo NameNode;
- após o envio dos blocos, o NameNode atualiza os seus metadados.

A seguir, veja algumas conclusões que podem ser obtidas a partir da Figura 2.

- O NameNode é o gestor de todas as operações e possui o mapa geral do armazenamento. Ele também sabe onde ler e onde guardar os arquivos nos DataNodes. O NameNode ainda é responsável por informar ao cliente como o arquivo será dividido e em quais locais será gravado/lido, bem como é responsável por permitir isso ou não.
- O Secondary NameNode não é uma cópia do NameNode (apesar do nome) e ele não se envolve nas operações normais. O Secondary NameNode armazena uma cópia do sistema de arquivos (FSimage) e os arquivos de *logs*. Uma de suas funções mais importantes é manter o arquivo de *logs* com o menor tamanho possível, para que o processo de reinicialização do NameNode seja rápido e eficiente em caso de falhas. Se o NameNode falhar, os metadados podem ser recuperados no FSimage armazenado no Secondary NameNode. Em resumo, o objetivo do Secondary NameNode é prover um *backup* dos metadados, porém ele não se envolve no processamento feito pelo NameNode nem pode substituí-lo. Como você viu, existe apenas um Secondary NameNode por *cluster* HDFS.
- Os DataNodes são os locais onde os dados ficam armazenados fisicamente, ou seja, os blocos dos arquivos originais. Eles podem estar em locais físicos próximos ou distantes. Nesses locais, ficam basicamente as estruturas dos arquivos em blocos e suas replicações. Pode haver milhares de DataNodes em um *cluster* Hadoop. Na Figura 2, há cinco DataNodes.

Agora você vai conhecer melhor o sistema de arquivos do HDFS e ver como é feito o seu armazenamento pelos DataNodes. Como você já viu, todos os dados são armazenados em blocos. Um bloco HDFS padrão tem 64 MB (pode ser configurado na instalação para ser maior), e isso é muito importante para um sistema de *big data*, pois as operações de gravação e leitura se tornam mais eficientes quando são gravados blocos maiores.

Em um sistema HDFS, mesmo arquivos pequenos são gravados eficientemente, pois são gravados em conjunto com outros dados e em blocos grandes de um mesmo tamanho. Para uma breve comparação: os maiores blocos de sistemas de arquivos mais comuns não distribuídos chegam a 64 KB, e esse tamanho é eficiente para eles, pois estão sendo pensados para armazenamento não distribuído.

Por fim, veja uma observação importante sobre o HDFS:

> O único método nativo de acesso ao HDFS é sua API Java. Todos os outros métodos de acesso são criados sobre essa API e, por definição, podem expor apenas a funcionalidade que permitir. Em um esforço para facilitar a adoção e o desenvolvimento de aplicativos, a API do HDFS é simples e familiar para os desenvolvedores, pegando carona em conceitos como os fluxos de E/S do Java (SAMMER, 2012, p. 20).

Ou seja, se uma empresa possuir profissionais habilitados para usar a linguagem Java, será muito mais fácil compreender e utilizar o ecossistema do HDFS.

Funcionamento do HDFS e sua tolerância a falhas

Quando se fala na tolerância a falhas do HDFS, basicamente estão em jogo a divisão dos dados em blocos para serem gravados e a replicação de segurança desses dados. Quando um arquivo é gravado no sistema HDFS, o NameNode determina como o arquivo será dividido em blocos colocando um identificador; após, ele replica os dados para que se tenha uma tolerância a falhas aprimorada. Por exemplo, caso um DataNode falhe com um bloco (fração) do arquivo original, a sua réplica poderá ser obtida em outro DataNode (caso a replicação tenha sido adotada). É importante notar que falhas em ambientes de *big data* são comuns, devido à sua escalabilidade. Afinal, é possível utilizar milhares de máquinas para armazenamento/processamento, aumentando as possibilidades de falhas.

Observe a Figura 3, a seguir.

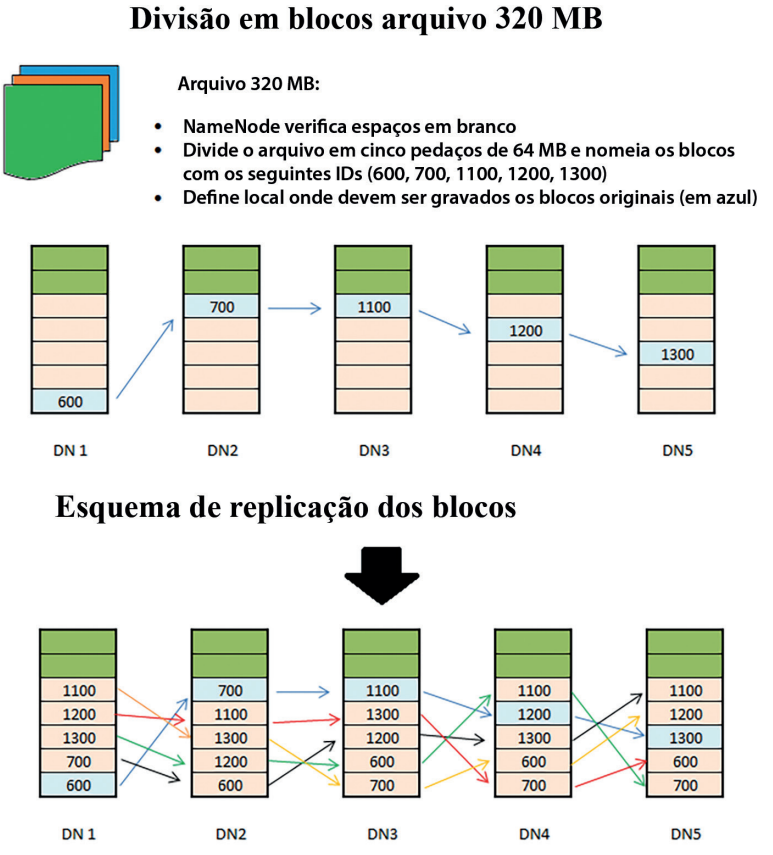

Figura 3. Divisão em blocos e tolerância a falhas com fator de replicação igual a 5. Ou seja, cada bloco é replicado cinco vezes.

Como você pode observar na primeira parte da Figura 3, o HDFS armazena o arquivo em blocos (nesse exemplo, de 64 MB). A vantagem de armazenar os dados em blocos advém de três fatores. Veja a seguir.

- **Velocidade:** imagine que você precisa pintar uma casa de 320 m^2 e pode dividir essa pintura entre cinco pessoas, cada uma com 64 m^2 para pintar. Supondo que os pintores tenham a mesma velocidade e a mesma capacidade, o trabalho será finalizado mais rápido do que se

você pintasse sozinho. No caso do HDFS, todos os DataNodes iniciam o armazenamento ao mesmo tempo, acelerando o processo total.

- **Regravação:** em caso de erros, é mais fácil refazer apenas um bloco. Retome o exemplo da pintura: se um dos pintores pintou inadequadamente os seus 64 m^2, somente essa parte deve ser refeita. O mesmo acontece com os blocos nos DataNodes.
- **Segurança:** os dados estão espalhados em diversos discos e depois serão replicados para a segurança da tolerância a falhas. Para tanto, o HDFS replica o mesmo bloco do arquivo (fração) entre diversos DataNodes. Como você viu, na Figura 3 é utilizado o fator de replicação igual a 5; assim, em caso de falha, sempre será possível obter a réplica do bloco em outro DataNode.

A segunda parte da Figura 3 mostra a replicação propriamente dita. Repare que não é necessário um local específico para cada bloco, e isso também aumenta a velocidade de armazenamento. A ideia geral da replicação é que blocos iguais fiquem armazenados em discos diferentes, por isso você nunca verá o bloco com ID 600 em um único disco, mas o verá espalhado pelos discos de diversos DataNodes. O número de replicações é definido na instalação do sistema. No caso da Figura 3, o fator de replicação adotado é igual a 5, porém tradicionalmente utiliza-se o fator igual a 3, para tolerar até duas falhas de DataNodes. As setas mostram por onde foi feita cada replicação.

Em resumo, a divisão de blocos torna todo o sistema de armazenamento do HDFS mais rápido e eficiente. Além disso, o NameNode possui todo o entendimento da estrutura e sabe onde os dados estão gravados, como se fosse um mapa de toda a estrutura. Por isso, ele consegue gerenciar muito rapidamente gravações, leituras, replicações e outras operações.

Além de todas essas características, o HDFS também é um sistema altamente tolerante a falhas. Em qualquer modo de uso dos dados, o NameNode consegue acompanhar detalhadamente o que acontece, usando de tempos em tempos seu relatório de listas de blocos.

Para entender como o NameNode consegue se recuperar em caso de falha, você vai acompanhar agora um exemplo baseado na Figura 3. Imagine que você quer ler um arquivo e para isso reúne todos os blocos desse arquivo. Você vai começar na seguinte sequência (supondo que o arquivo esteja nos DataNodes e blocos descritos a seguir):

- DataNode 1, bloco 1200;
- DataNode 2, bloco 1100;
- DataNode 3, bloco 1300;
- DataNode 4, bloco 700;
- DataNode 5, bloco 600.

Imagine que o bloco 600 do DataNode 5 ficou com um problema, ou mesmo que todo o DataNode teve um problema, como mostra a Figura 4.

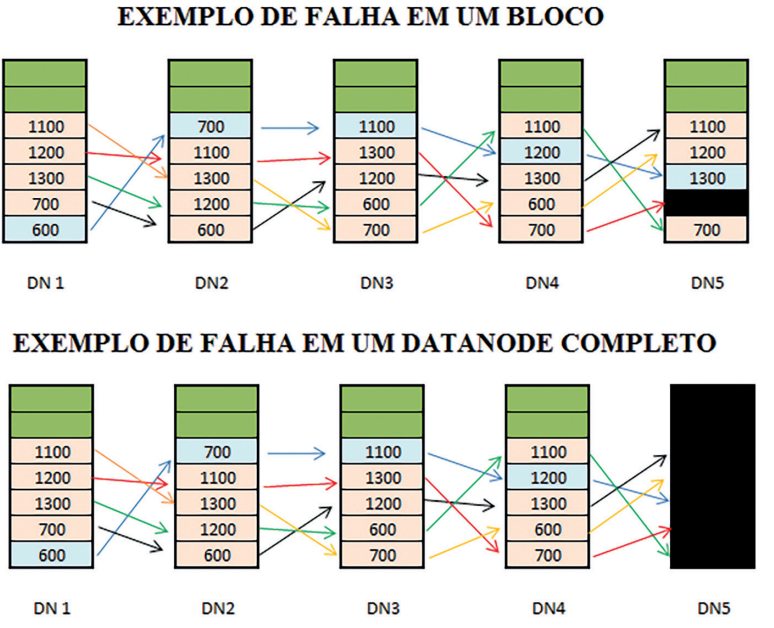

Figura 4. Problemas em um bloco ou em um DataNode.

O que vai acontecer? Primeiro, o HDFS vai reconhecer o problema e o NameNode vai buscar os dados de qualquer outra replicação. Ou seja, o bloco 600 poderá ser lido do DN1, do DN2, do DN3 e inclusive do próprio DN4, onde foi feita a última leitura válida. Isso acontece em ambos os casos: em caso de falha total do DN5 ou em caso de falha apenas no bloco 600 do DN5.

Existem duas possibilidades para leitura/recuperação, como você pode ver a seguir.

1. No caso de todo o DN5 ter sido perdido, serão providenciadas novas cópias de todos os blocos (600, 700, 1100, 1200, 1300) para todos os outros DataNodes funcionais. Será feita uma cópia de cada dado perdido até a obtenção do número de cópias original.
2. No caso de apenas o bloco 600 do DN5 estar com problema, ele será copiado novamente para o DN5 em um local onde não existe problema.

Uma observação importante sobre o processo é que os dados não são recuperados dos locais que estão com problemas. Eles são sempre recuperados de cópias válidas de outros blocos e de outros DataNodes funcionais.

Em resumo, a tolerância a falhas é provida por outras cópias existentes e somente é satisfeita quando os DataNodes alcançam o número de cópias originais existente antes do problema. Note também a importância de o HDFS trabalhar em blocos: isso acelera muito o tempo de recuperação do sistema, pois é preciso replicar apenas blocos com problemas, e não arquivos inteiros.

Na Figura 5, veja o caminho para a gravação em blocos em condições normais.

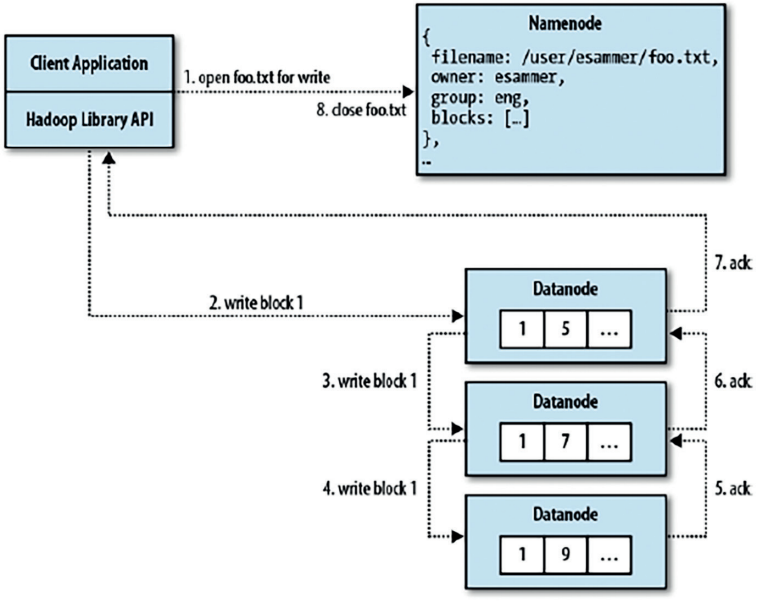

Figura 5. Caminho da gravação em blocos.
Fonte: Sammer (2012, p. 14).

Observe o cliente solicitando acesso ao NameNode. Depois das autorizações e regras de gravação, os dados são passados à API do Hadoop/HDFS, e via API é feito o acesso/gravação. Toda vez que for gravado um bloco, automaticamente a replicação será feita, para preservar a possibilidade de tolerância a falhas.

Fique atento

A replicação pode ser definida pelo administrador do sistema, mas por padrão são feitas três replicações. O arquivo que faz essas definições é o `dfs.replication`, dentro do arquivo `dfs.datanote.data.dir`.

Em qualquer sistema, se você aumentar muito o número de cópias, terá um problema de velocidade. Portanto, se você quer mais velocidade, pode diminuir o número de cópias. Contudo, quanto menos cópias houver, menos seguro será o sistema de replicação. Por outro lado, você terá menos gastos de armazenamento. Nessa mesma lógica, quanto mais cópias existirem, mais seguro será o sistema, mas haverá perda de velocidade e você terá mais gastos com armazenamento. A dosagem ideal, que já vem no sistema, é de três cópias/replicações. O ajuste deve seguir a análise ou o período de testes.

Durante a sua instalação, o HDFS permite diversos testes e oferece três formatos de instalação. Os dois mais importantes são um modo totalmente distribuído, que realmente será o modo de implantação na empresa, e um modo pseudodistribuído, que é capaz de permitir testes até a implementação adequada. Em suma, o ideal é realizar diversos testes no modo pseudodistribuído, que pode ser escolhido na instalação. Depois de determinar todas as suas necessidades, utilize o modo totalmente distribuído, que oferece todo o potencial do HDFS.

Referências

AGGARWAL, C. C. *Data mining:* the textbook. Heidelberg: Springer, 2015.

GOLDMAN, A. *et al.* Apache hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. *In:* CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 32., 2012. Anais [...]. Curitiba: 2012. Disponível em: //www2.sbc.org.br/csbc2012/anais_csbc/eventos/jai/index.html. Acesso em: 13 set. 2020.

KHAN, M. A.; UDDIN, M. F.; GUPTA, N. Seven V's of big data understanding big data to extract value. In: ASEE Zone 1, 2014. *Proceedings* [...]. 2014. Disponível em: https://www.asee.org/documents/zones/zone1/2014/Professional/PDFs/113.pdf. Acesso em: 13 set. 2020.

SAMMER, E. *Hadoop operations:* a guide for developers and administrators. California: O´Reilly, 2012.

Leituras recomendadas

BRAMER, M. *Principles of data mining:* undergraduate topics in computer science. 3rd. ed. Heidelberg: Springer, 2016.

MARR, B. *Big Data in practice:* how 45 successful companies used big data analytics to deliver extraordinary results. USA: Wiley, 2016.

WHITE, T. *Hadoop the definitive guide:* storage and analysis at internet scale. 4th. Ed. California: O´Reilly, 2015.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS