FUNDAMENTOS DE BIG DATA

Thiago Nascimento Rodrigues

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

MapReduce na prática: submissão de tarefas ao Hadoop em Java

OBJETIVOS DE APRENDIZAGEM

- Determinar o processo de submissão de tarefas MapReduce em Java no Hadoop.
- Estabelecer o monitoramento da execução da tarefa no Hadoop.
- Demonstrar comandos e processos envolvidos para submissão da tarefa MapReduce em Java.

Introdução

O Hadoop disponibiliza uma interface de programação de aplicações MapReduce completa e flexível para os cenários mais variados. Uma maneira prática de se desenvolver soluções baseadas nessa infraestrutura de programação é utilizando os diferentes modos de execução do Hadoop. Isso permite a construção de um código-fonte mais robusto e mais preciso quanto ao atendimento dos requisitos de um sistema para processamento de *big data*.

Neste capítulo, você vai estudar as diferentes etapas que compõem o processo de submissão de tarefas MapReduce no Hadoop e vai aprender como o monitoramento dessas tarefas pode ser feito por meio dos serviços e das interfaces

disponíveis no Hadoop. Além disso, você vai se familiarizar com os principais comandos que dão suporte à execução de tarefas e à coleta dos resultados gerados por uma solução MapReduce.

Tarefas MapReduce no Hadoop

O **MapReduce Hadoop** é um *framework* que dá suporte ao desenvolvimento de aplicações que processam grandes quantidades de dados em paralelo e em grandes *clusters* de maneira confiável e tolerante a falhas. De maneira geral, um fluxo de trabalho MapReduce (conhecido como ***job***) divide o conjunto de dados de entrada em blocos independentes, que são processados pelas tarefas de mapeamento (etapa `map`) de forma completamente paralela. O Hadoop efetua a classificação das saídas dos mapeamentos, que, por sua vez, são inseridas como entradas para as tarefas de redução (etapa `reduce`). Normalmente, tanto a entrada quanto a saída de um fluxo de trabalho são armazenadas em um sistema de arquivos. O Hadoop é responsável pelo agendamento e pelo monitoramento de tarefas, além da reexecução daquelas que apresentam alguma falha.

A estrutura MapReduce do Hadoop é composta de três elementos principais: um único gerenciador de recursos — ***Resource Manager*** —, um gerenciador de trabalho do nó — ***Node Manager*** — presente em cada nó do *cluster*, e um gerenciador de aplicação MapReduce associado a cada aplicação em execução. Esse conjunto de elementos constitui uma camada de sistema do Hadoop conhecida como **YARN** (do inglês *Yet Another Resource Negotiator*, cuja tradução é "mais um negociador de recursos").

A Figura 1 descreve como o YARN opera durante a execução de uma aplicação MapReduce. Para executar uma aplicação no YARN, um cliente precisa interagir com o gerenciador de recursos e solicitar que ele dispare um processo mestre de aplicação (etapa 1). O gerenciador de recursos então encontra um gerenciador de nó que pode iniciar a aplicação mestre em um *container* (etapas 2a e 2b). O que uma aplicação mestre faz exatamente depende do que foi submetido pelo cliente. Ela poderia, por exemplo, simplesmente executar um cálculo no *container* em que está sendo executado e retornar o resultado ao cliente. Ou poderia solicitar para o gerenciador de recursos mais *containers* (etapa 3) e usá-los para executar uma computação distribuída (etapas 4a e 4b).

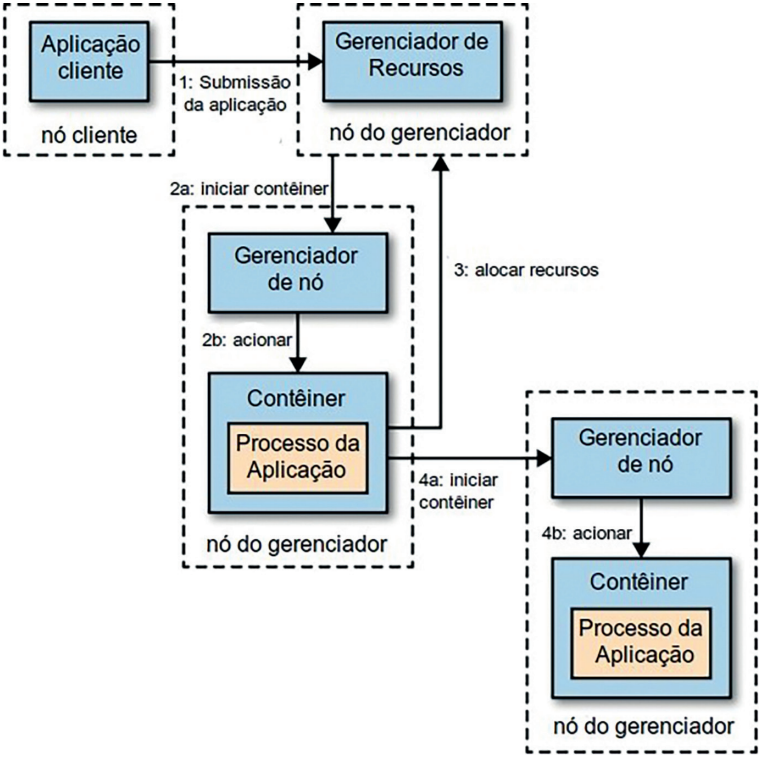

Figura 1. Como o YARN executa uma aplicação.
Fonte: Adaptada de White (2015).

Já a estrutura de armazenamento do Hadoop — o **HDFS** (do inglês *Hadoop Distributed File System*, ou Sistema de Arquivos Distribuído do Hadoop) — é baseada no modelo mestre-escravo (ou mestre-trabalhador). Os nós escravos — chamados *DataNodes* — têm a responsabilidade de manter os dados reais do sistema de arquivos. O próprio *DataNode*, porém, não tem conhecimento do sistema de arquivos geral; seu papel é armazenar, servir e garantir a integridade dos dados pelos quais é responsável. O nó mestre — chamado de *NameNode* — é responsável por saber qual dos *DataNodes* contém qual bloco de dado e como esses blocos são estruturados para formar o sistema de arquivos completo. Quando um cliente olha para o sistema de arquivos e deseja recuperar um arquivo, é por meio de uma solicitação ao *NameNode* que a lista de blocos necessários é recuperada (TURKINGTON; MODENA, 2015).

A execução de uma aplicação MapReduce no Hadoop depende de um número mínimo de passos de desenvolvimento e configuração. As aplicações

devem especificar os locais de entrada/saída de dados e fornecer as funções de mapeamento e redução por meio de implementações de interfaces ou classes abstratas apropriadas. Finalizada essa etapa, a aplicação deve ser enviada como um *job* (arquivo JAR — do inglês Java *archive*, ou arquivo Java) para o Hadoop, juntamente com as configurações necessárias para o *Resource Manager*. Neste ponto, ele assume a responsabilidade de distribuir o *software*/as configurações aos nós trabalhadores, agendar tarefas e monitorá-los, fornecendo informações do estado e diagnóstico do *job* para o cliente.

O Hadoop conta com dois modos de implantação e execução de *jobs*: o modo pseudodistribuído e o modo completamente distribuído. Uma implantação de nó único do Hadoop é conhecida como execução no **modo pseudodistribuído**. Nesse caso, todos os serviços do Hadoop, incluindo os serviços mestre e escravo, são executados em um único nó de computação. Esse tipo de implantação é útil para testar aplicações de maneira rápida enquanto o desenvolvimento está em progresso. Assim, considerações relativas ao uso de recursos de *cluster* do Hadoop podem ser abstraídas, além de ser uma maneira conveniente de se experimentar o inteiro mecanismo de funcionamento do Hadoop. Por essas razões, este capítulo é baseado no modo pseudodistribuído do Hadoop. Por outro lado, uma implantação do Hadoop em que os serviços mestre e escravo do Hadoop são executados em um ambiente de *cluster* de computadores é conhecido como **modo completamente distribuído**. Esse é o modo adequado para *clusters* de produção e *clusters* de desenvolvimento (DEROOS *et al*., 2014).

Alguns pré-requisitos são necessários para que o ecossistema Hadoop possa ser executado. Um deles é que um JRE (do inglês Java *Runtime Environment*, ou Ambiente de Execução Java) esteja instalado no sistema. Isso pode ser verificado por meio do comando:

```
java -version
```

que deve retornar a versão do JRE disponível. Além disso, o completo conjunto de arquivos que constituem uma instalação Hadoop deve ter sido baixado e descompactado no sistema (THE APACHE SOFTWARE FOUNDATION, 2019). O nome do arquivo correspondente a uma instalação Hadoop tem o formato:

```
hadoop-X.Y.Z.tar.gz
```

onde `X.Y.Z` correspondem ao número da versão do Hadoop. Neste capítulo, foi utilizada a versão 3.3.0. Todas as configurações requeridas para a preparação do modo de execução Hadoop se concentram nos arquivos presentes no diretório etc.

A configuração do Hadoop para operar em modo *cluster* envolve a alteração de quatro arquivos principais. Todas as configurações são baseadas em pares chave/valor (elementos `<name/>` e `<value/>`, respectivamente) definidos dentro de elementos `<property/>` em cada arquivo. O primeiro arquivo a ser alterado é o arquivo `core-site.xml` que contém as especificações do *NameNode*. A principal configuração é a definição do URI (do inglês *unified resource identifier*, ou identificador uniforme de recurso) em que o *NameNode* vai executar. Para a operação em modo *cluster*, o URI deve iniciar com o prefixo `hdfs://` seguido do *host* e do número da porta. O código a seguir apresenta o nome da propriedade e o URI identificando que o *NameNode* vai executar como *localhost* e vai escutar requisições na porta 9000.

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Fique atento

Em versões mais antigas do Hadoop, o nome da propriedade usada para configurar o URI do *NameNode* era `fs.default.name`. Nas versões mais novas, ela foi substituída por `fs.defaultFS`.

O segundo arquivo a ser configurado é o `hdfs-site.xml`. Esse arquivo contém as propriedades relativas ao sistema de arquivos distribuído do Hadoop — o HDFS. Uma propriedade importante a ser definida é o número de réplicas de cada bloco de arquivo armazenado no HDFS — propriedade `dfs.replication`. É recomendável que o valor indicado nessa propriedade não seja superior ao número de nós trabalhadores (escravos) do *cluster*. O código a seguir apresenta a configuração a ser inserida no arquivo.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

O arquivo `mapred-site.xml` contém informações referentes às operações de MapReduce. Basicamente, é preciso indicar o YARN como *framework* responsável pela execução das operações de mapeamento e redução. Para isso, deve-se utilizar a propriedade `mapreduce.framework.name`. O próximo código apresenta a configuração que deve ser adicionada ao arquivo.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

O quarto arquivo a receber configurações para operacionalizar o modo *cluster* do Hadoop é o `yarn-site.xml`. Nele, deve-se indicar o nome do *host* onde o *Resource Manager* vai executar (propriedade `yarn.resourcemanager.hostname`). Além disso, os serviços auxiliares utilizados pelo YARN devem ser apontados na propriedade `yarn.nodemanager.aux-services`. Para aplicações MapReduce, o mecanismo de embaralhamento (`mapreduce _ suffle`) deve ser configurado. O trecho de código a seguir aponta as configurações que devem ser adicionadas ao arquivo.

```
<configuration>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>localhost</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce _ shuffle</value>
  </property>
</configuration>
```

Uma vez configurado o *cluster* (neste caso, em modo pseudodistribuído), antes da submissão de qualquer tarefa, é preciso executar ainda a formatação do sistema de arquivos que vai operar sobre ele. Isso é feito por meio do comando `hdfs`. A partir desse ponto, o *cluster* pode entrar em operação e receber a submissão de tarefas. Primeiramente, os serviços responsáveis

pelo *NameNode* e *DataNode* são inicializados por meio do comando `start--dfs.sh`. Na sequência, os serviços de *Resource Manager* e *Node Manager* são ativados por meio do comando `start-yarn.sh`.

Fique atento

Para que os comandos do Hadoop fiquem disponíveis em qualquer diretório em que o *prompt* de comando esteja, é preciso que a variável de ambiente `HADOOP _ HOME` seja definida. Ela deve conter o caminho para o diretório de instalação do Hadoop (diretório onde o arquivo `.tar.gz` foi descompactado). E, para que a definição dessa variável não se perca em função de uma reinicialização de sistema, é recomendável que ela seja definida em *scripts* como `.bashrc` ou `.profile`. Por exemplo, se o diretório de instalação do Hadoop for `/usr/local/hadoop-3.3.0`, o seguinte comando deve ser executado ou adicionado aos arquivos sugeridos:

```
export HADOOP _ HOME=/usr/local/hadoop-3.3.0
```

A submissão de uma tarefa para o Hadoop significa, na prática, indicar qual arquivo JAR de aplicação deve ser executado no *cluster*. Além disso, devem ser informados o diretório onde se encontram os arquivos a serem processados e o diretório onde os dados gerados pela aplicação MapReduce devem ser armazenados. Isso pode ser feito pelo YARN, por meio do comando com o seguinte formato:

```
yarn jar <arquivo JAR> <diretório de entrada> <diretório
de saída>
```

ou por meio de um comando análogo disponível no próprio Hadoop:

```
hadoop jar <arquivo JAR> <diretório de entrada> <diretório
de saída>
```

Monitoramento de tarefas no Hadoop

O trabalho de monitoramento de tarefas no Hadoop pode ser bem simplificado quando as interfaces *web* dos serviços inicializados são utilizadas. Nesse sentido, o *Resource Manager* disponibiliza várias informações a respeito do próprio *cluster* e das tarefas submetidas a ele. O endereço padrão para se acessar sua interface é `http://localhost:8088`. A Figura 2 apresenta a página inicial da interface. A seção Cluster Metrics (métricas do *cluster*)

exibe um sumário do *cluster*. Isso inclui o número de aplicações que estão em execução no momento (e em vários outros estados), os números relativos aos recursos disponíveis (por exemplo, Memory Total, que significa total de memória), além de informações a respeito dos *Node Managers*.

A tabela principal dessa interface (Figura 2) exibe todas as aplicações que já foram executadas ou que estão em execução. As subseções da seção Applications possibilita que as aplicações sejam filtradas de acordo como o seu *status* de execução: submetidas (*submitted*), aceitas (*accepted*), em execução (*running*), concluídas (*finished*), falhas (*failed*) e abortadas (*killed*). Todas as etapas de processamento de uma aplicação Hadoop são armazenadas de forma a permitir que sejam consultadas mesmo após sua conclusão. Essas informações podem ser consultadas por meio da interface de *job* cujo URI padrão é `http://localhost:19888`. Acessando esse endereço por meio de um navegador, é possível obter todo o detalhamento de qualquer aplicação que já tenha sido submetida ao *cluster* Hadoop.

Um exemplo das informações que podem ser recuperadas é apresentado na Figura 3. Enquanto uma aplicação (*job*) está em execução, é possível monitorar o seu progresso por meio dessa página. A tabela na parte inferior da página exibe o *status* de processamento das etapas de mapeamento e redução. Nessa mesma seção, a coluna Total apresenta o número total de tarefas `map` e `reduce` que foram executadas para aquele *job*. As demais colunas exibem o *status* de cada uma dessas tarefas: pendente, ou esperando para ser executada (*pending*); em execução (*running*); ou completa — finalizada com sucesso (*complete*). Ainda mais abaixo nessa mesma tabela, é possível encontrar o número total de tentativas de execução de tarefas `map` e `reduce` que falharam ou foram abortadas.

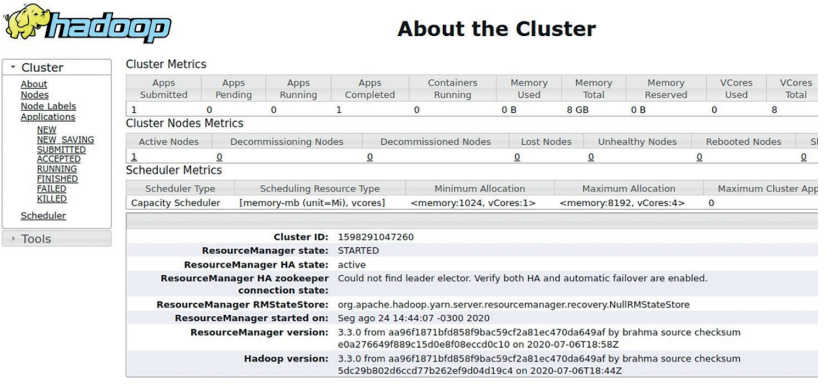

Figura 2. Interface de monitoramento do *Resource Manager*.

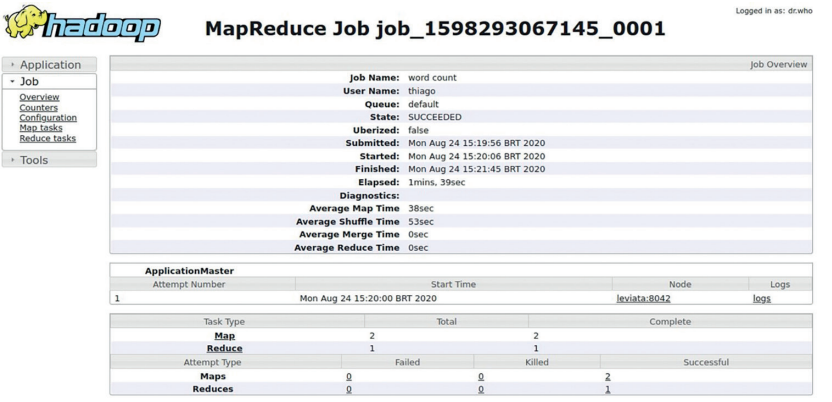

Figura 3. Interface de detalhamento de um *job*.

Outra maneira de monitorar a execução de um *job* no Hadoop é por meio das informações que são impressas no console. O código a seguir mostra um exemplo de saída gerada por uma aplicação MapReduce (algumas linhas foram removidas para facilitar o entendimento). Antes de o *job* iniciar sua execução, o seu identificador é impresso. Esse identificador é importante, porque ele é referenciado sempre que alguma informação do *job* precisar ser recuperada — por exemplo, em arquivos de *log*. Quando uma aplicação tem sua execução concluída, diversas estatísticas (contadores) são impressas no console. As informações contabilizadas por esses contadores são muito úteis para, por exemplo, confirmar que o *job* realizou o que era esperado.

```
2020-08-24 15:19:55,141 INFO input.FileInputFormat: Total
input files to process : 2
...
2020-08-24 15:19:57,085 INFO impl.YarnClientImpl: Submitted
application application _ 1598293067145 _ 0001
2020-08-24 15:19:57,128 INFO mapreduce.Job: The
url to track the job: http://leviata:8088/proxy/
application _ 1598293067145 _ 0001/
2020-08-24 15:19:57,129 INFO mapreduce.Job: Running job:
job _ 1598293067145 _ 0001
2020-08-24 15:20:07,397 INFO mapreduce.Job: Job
job _ 1598293067145 _ 0001 running in uber mode : false
2020-08-24 15:20:07,400 INFO mapreduce.Job: map 0% reduce 0%
```

```
2020-08-24 15:20:50,541 INFO mapreduce.Job:  map 100% reduce 0%
2020-08-24 15:21:46,494 INFO mapreduce.Job:  map 100% re-
duce 100%
2020-08-24  15:21:47,511  INFO  mapreduce.Job:  Job
job _ 1598293067145 _ 0001 completed successfully
       File System Counters
              FILE: Number of bytes read=185
              FILE: Number of bytes written=790155
              FILE: Number of read operations=0
              FILE: Number of large read operations=0
              FILE: Number of write operations=0
              HDFS: Number of bytes read=311
              HDFS: Number of bytes written=67
              HDFS: Number of read operations=11
              HDFS: Number of large read operations=0
              HDFS: Number of write operations=2
              HDFS: Number of bytes read erasure-coded=0
       Job Counters
              Launched map tasks=2
              Launched reduce tasks=1
              Data-local map tasks=2
              Total time spent by all maps in occupied slots
       ms)=77263
              Total time spent by all reduces in occupied
       slots (ms)=53640
              Total time spent by all map tasks (ms)=77263
              Total time spent by all reduce tasks (ms)=53640
              Total vcore-milliseconds taken by all map
       tasks=77263
              Total vcore-milliseconds taken by all reduce
       tasks=53640
              Total megabyte-milliseconds taken by all map
       tasks=79117312
              Total megabyte-milliseconds taken by all re
       duce tasks=54927360
       Map-Reduce Framework
              Map input records=9
              Map output records=15
```

```
        Map output bytes=149
        Map output materialized bytes=191
        Input split bytes=222
        Combine input records=0
        Combine output records=0
        Reduce input groups=8
        Reduce shuffle bytes=191
        Reduce input records=15
        Reduce output records=8
        Spilled Records=30
        Shuffled Maps =2
        Failed Shuffles=0
        Merged Map outputs=2
        GC time elapsed (ms)=191
        CPU time spent (ms)=2690
        Physical memory (bytes) snapshot=670179328
        Virtual memory (bytes) snapshot=7965003776
        Total committed heap usage (bytes)=482344960
        Peak Map Physical memory (bytes)=245366784
        Peak Map Virtual memory (bytes)=2652631040
        Peak Reduce Physical memory (bytes)=180695040
        Peak Reduce Virtual memory (bytes)=2660864000
...
```

Submissão de tarefas MapReduce no Hadoop

Para refinar o entendimento acerca do processo de submissão de tarefas MapReduce no Hadoop, será apresentado a seguir um caso prático. Para isso, uma aplicação de contagem de palavras (em inglês, *wordcount*) escrita em Java será submetida ao *cluster* Hadoop. Tanto o código da aplicação como o arquivo JAR (`hadoop-cases.jar`) preparado para submissão podem ser encontrados em Hadoop Cases (2020).

Assumindo que todas as configurações apresentadas na primeira seção já foram efetuadas, ou seja, as devidas propriedades foram adicionadas aos arquivos `core-site.xml`, `hdfs-site.xml`, `mapred-site.`

`xml` e `yarn-site.xml`, então, os serviços (conhecidos como *daemons*) referentes ao *NameNode* e ao *DataNode* podem ser inicializados por meio do comando

```
start-dfs.sh
```

Para verificar se os serviços foram inicializados corretamente, pode-se acessar a interface *web* disponibilizada no endereço `http://localhost:9870`. A Figura 4 apresenta um trecho da página inicial exibida pela interface gráfica do *NameNode*. Informações gerais a respeito do sistema de arquivos inicializado são apresentadas.

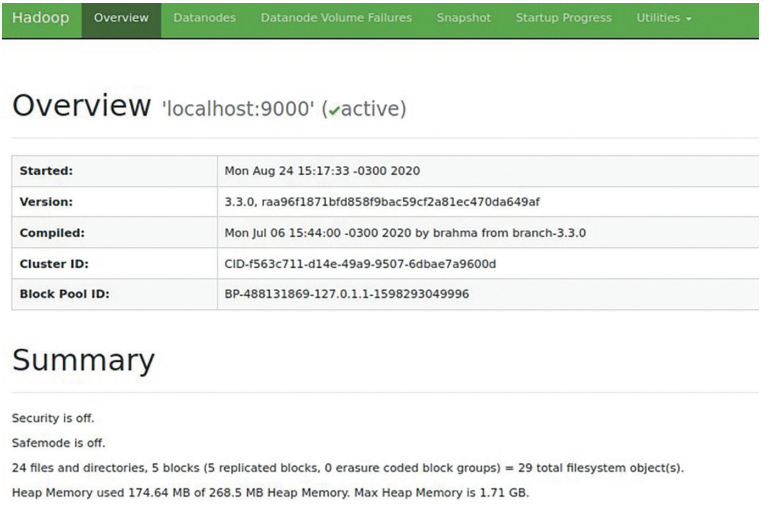

Figura 4. Interface *web* de monitoramento do NameNode.

O Hadoop espera que exista em seu sistema de arquivos distribuído um diretório relativo ao usuário que está operando sobre o HDFS. Esse diretório deve ser interno ao diretório principal `/user` e deve ter o mesmo nome do usuário. Neste caso, assumindo que o usuário Thiago vai submeter *jobs* ao Hadoop, o diretório `/user/thiago` deve ser criado no HDFS. Para isso, os comandos a seguir devem ser executados.

```
hadoop fs -mkdir /user
hadoop fs -mkdir /user/thiago
```

Fique atento

O HDFS assume como diretório padrão o diretório `/user/<nome do usuário>`. Logo, se o caminho absoluto (iniciado por "/") não for utilizado nos comandos de manipulação de arquivos e diretórios, o Hadoop executará os comandos considerando o caminho relativo a partir do diretório padrão. Por exemplo, se o usuário Fulano executar o comando

```
hadoop fs -mkdir dados-entrada
```

o Hadoop tentará fazer a criação de um subdiretório de nome `dados-entrada` no diretório `/user/fulano`.

A aplicação MapReduce de contagem de palavras processa arquivos de texto presentes em um diretório e efetua a consolidação do número de ocorrências de cada palavras nesses arquivos. A resposta gerada pela aplicação é um arquivo de texto composto de pares chave/valor, em que a chave corresponde a uma palavra encontrada nos arquivos processados, e o valor, ao número de ocorrências da respectiva palavra.

Cada linha do arquivo abriga um único par chave/valor. Logo, é preciso informar para a aplicação tanto o diretório onde se encontram os arquivos a serem analisados como o diretório onde a saída será gravada. Como a aplicação se encarrega de criar o diretório de saída, apenas o diretório de entrada (`input`, por exemplo) precisa ser criado no HDFS. Para isso, o comando a seguir deve ser executado.

```
hadoop fs -mkdir input
```

Os arquivos a serem utilizados para essa submissão do contador de palavras ao Hadoop foram apresentados por Goldman *et al*. (2012). O conteúdo dos dois arquivos é exibido no Quadro 1, a seguir. Ambos os arquivos se encontram, inicialmente, apenas no sistema de arquivos local da máquina onde o *cluster* Hadoop foi configurado. Então, eles precisam ser copiados para o HDFS. Para isso, considerando-se que os arquivos estão no diretório `/home/thiago/wordcount` e devem ser carregados para o diretório `/user/thiago/input` do HDFS, o comando a seguir deve ser executado.

```
hadoop fs -copyFromLocal /home/thiago/wordcount/* input
```

Quadro 1. Arquivos de entrada para o contador de palavras

file01	file02
CSBC JAI 2012 CSBC 2012 em Curitiba	Minicurso Hadoop JAI 2012 CSBC 2012 Curitiba Paraná

Fonte: Goldman *et al.* (2012).

Toda a estrutura de diretórios criada no HDFS, assim como os dois arquivos carregados, podem ser visualizados tanto via comandos como via interface *web* do *NameNode*. Para listar os arquivos do diretório `/user/thiago/input` do HDFS, o comando a seguir pode ser executado:

```
hadoop fs -ls input
```

que deve gerar uma saída como a seguinte

```
Found 2 items
-rw-r--r-- 1 thiago supergroup 36 2020-08-24 15:17 input/
file01
-rw-r--r-- 1 thiago supergroup 53 2020-08-24 15:17 input/
file02
```

O mesmo resultado pode ser obtido por meio da interface gráfica do *NameNode* (`http://localhost:9870`). Para isso, é preciso acessar o menu Utilities > Browse the file system. A Figura 5 apresenta a tela exibida após se ter acesso ao menu. Ambos os arquivos são listados da mesma forma que o resultado obtido via linha de comando. Além disso, no campo de busca, é possível verificar que o diretório acessado é o `/user/thiago/input`.

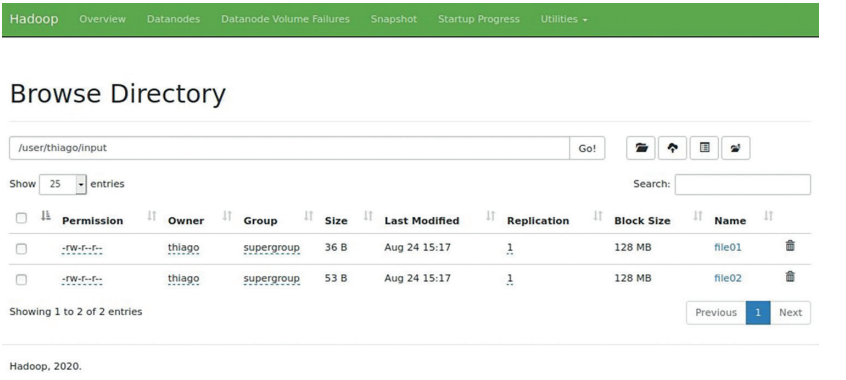

Figura 5. Listagem de arquivos no HDFS via interface *web*.

Para a submissão do *job* (`hadoop-cases.jar`) ao Hadoop, é recomendada a utilização do YARN. Nesse caso, a aplicação de contagem de palavras pode ser submetida por meio do comando

```
yarn jar hadoop-cases.jar input output
```

Assim como mostrado na seção anterior, o processo de execução da aplicação pode ser monitorado por meio do *log* gerado no console ou da interface gráfica do *Resource Manager*. A execução finalizada com sucesso desse *job* submetido pode ser verificada pelo trecho de *log* capturado do console e apresentado a seguir.

```
2020-08-24 15:19:57,085 INFO impl.YarnClientImpl: Submitted
application application _ 1598293067145 _ 0001
2020-08-24 15:19:57,128 INFO mapreduce.Job: The
url to track the job: http://leviata:8088/proxy/
application _ 1598293067145 _ 0001/
2020-08-24 15:19:57,129 INFO mapreduce.Job: Running job:
job _ 1598293067145 _ 0001
2020-08-24 15:20:07,397 INFO mapreduce.Job: Job
job _ 1598293067145 _ 0001 running in uber mode : false
2020-08-24 15:20:07,400 INFO mapreduce.Job: map 0% reduce 0%
2020-08-24 15:20:50,541 INFO mapreduce.Job: map 100% reduce 0%
```

```
2020-08-24 15:21:46,494 INFO mapreduce.Job:  map 100% reduce 100%
2020-08-24  15:21:47,511  INFO  mapreduce.Job:  Job
job _ 1598293067145 _ 0001 completed successfully
```

Uma nova listagem dos diretórios do HDFS indica que o diretório `output` foi corretamente criado pela aplicação e que os arquivos de saída também foram gerados. A execução do comando abaixo seguido da saída produzida atestam isso.

```
thiago@leviata:~$ hadoop fs -ls output
Found 2 items
-rw-r--r--  1  thiago  supergroup   0  2020-08-24  18:24
output/ _ SUCCESS
-rw-r--r-- 1 thiago supergroup 67 2020-08-24 18:24 output/
part-r-00000
```

Para que os dados produzidos pela aplicação possam ser recuperados do HDFS, o comando a seguir pode ser executado. Ele fará a cópia do arquivo `part-r-00000` do diretório `output` do HDFS para o sistema de arquivos local. O Quadro 2, a seguir, apresenta o conteúdo do arquivo recuperado. Ele atesta a correta execução da aplicação de contagem de palavras, conforme indicado por Goldman *et al.* (2012).

```
hadoop fs -copyToLocal output/part-r-00000
```

Quadro 2. Arquivo de saída gerado pelo contador de palavras

part-r-00000
2012 4 CSBC 3 Curitiba 2 Hadoop 1 JAI 2 Minicurso 1 Paraná 1 em 1

Referências

DEROOS, D. et al. *Hadoop for dummies*. Hoboken: John Wiley & Sons, 2014.

GOLDMAN, A. *et al*. Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. In: JORNADAS DE ATUALIZAÇÕES EM INFORMÁTICA, 31., 2012, Curitiba. Anais [...]. Curitiba: [UFPR], 2012. p. 88-136. Disponível em: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/JAI_2012_Cap%203_Apache%20Hadoop.pdf. Acesso: 13 set. 2020.

HADOOP CASES. Implementações Java de Casos de Uso de MapReduce para o Hadoop. Disponível em <https://github.com/tnas/hadoop-cases>. Acesso em: 24 ago. 2020.

THE APACHE SOFTWARE FOUNDATION. *Commuinty-led Development "The Apache Way"*. Version 2.0. [Wakefield, Massachusetts]: The Apache Software Foundation, c2019. Disponível em: https://www.apache.org/dyn/closer.cgi/hadoop/common/. Acesso em: 13 set. 2020.

TURKINGTON, G.; MODENA, G. *Learning Hadoop 2*. Birmingham: Packt Publishing, 2015.

WHITE, T. *Hadoop*: the definitive guide. 4th ed. Sebastopol, CA: O'Reilly, 2015.

Leitura recomendada

SITTO, K.; PRESSER, M. *Field guide to Hadoop*: an introduction to Hadoop, its ecosystem, and aligned technologies. Sebastopol, CA: 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