FUNDAMENTOS DE BIG DATA

SOLUÇÕES EDUCACIONAIS INTEGRADAS

MapReduce na prática: agregação parcial por meio do Combiner para otimizar desempenho

Roger Robson dos Santos

OBJETIVOS DE APRENDIZAGEM

- Classificar os desafios relacionados a desempenho no MapReduce no Hadoop.
- Reconhecer os fatores que impactam a escalabilidade de processamento no MapReduce.
- Desenvolver, de maneira prática e com códigos em Java, a utilização de *combiners* e *partitions* para otimizar desempenho.

Introdução

Diversas empresas buscam profissionais que entendam os processos de análise de grandes volumes de dados. Para entender esses processos, os profissionais precisam conhecer ferramentas de *big data*. Uma dessas ferramentas é o Hadoop, capaz de trabalhar com a enorme quantidade de dados que grandes empresas coletam diariamente. Dessa maneira, entender o Hadoop tem sido um pré-requisito para diversas vagas no contexto de *big data*. Ou seja, o mercado privilegia os profissionais que sabem lidar com essa ferramenta.

Neste capítulo, você vai conhecer técnicas de utilização do Hadoop e alguns recursos dessa ferramenta. Você também vai ler sobre os desafios relacionados ao desempenho do MapReduce no Hadoop. Além disso, vai ver como utilizar códigos práticos para tarefas com o MapReduce.

Desafios de desempenho no MapReduce no Hadoop

Big data com Apache Hadoop

Entre os maiores desafios computacionais da atualidade, destacam-se o armazenamento, a manipulação e a análise de grandes volumes de dados, que têm criado um enorme problema para diversas empresas. Entre esses dados, encontram-se informações de serviços locais, sistemas da *web* e diversos outros serviços disponíveis. Google e Facebook são duas das empresas que possuem grande volume de dados relacionados a pesquisas e buscas em suas redes, além de dados de usuários, regiões e associações entre anúncios, produtos e consumidores. Todo esse tráfego diário pode gerar terabytes de dados que precisam ser armazenados de maneira eficaz, além de manipulados e analisados em tempo hábil, enquanto novos dados chegam ao sistema (THE APACHE SOFTWARE FOUNDATION, 2020).

Soluções de *big data* são responsáveis por criar mecanismos capazes de armazenar e analisar grandes volumes de dados, processando cálculos pesados para identificar comportamentos e disponibilizar serviços específicos a grandes empresas, que normalmente se deparam com problemas computacionais. Entre as soluções de *big data*, destaca-se o ecossistema Hadoop.

O ecossistema Hadoop reúne uma gama de ferramentas que possibilitam o processamento e o armazenamento de grandes volumes de dados de forma distribuída, por meio de *clusters* de computadores (com baixo custo e tolerância a falhas). Ele já vem sendo utilizado por diversas empresas, como Facebook, Google e IBM.

Entre os benefícios dessa ferramenta, está o fato de ela ter código aberto, o que faz com que o projeto ganhe novas atualizações desenvolvidas por uma comunidade de grandes empresas ativas. Além disso, o ecossistema Hadoop

gera grande economia, pois é uma ferramenta livre e permite a utilização de *clusters* com diversos computadores em redes convencionais. Ademais, o Hadoop possui diversos serviços em nuvem com preços acessíveis às empesas (THE APACHE SOFTWARE FOUNDATION, 2020).

O Hadoop possui diversos componentes em sua arquitetura. Veja a seguir.

- **Hadoop Distributed File System (HDFS):** componente responsável por armazenar e manipular grandes volumes de dados, além de possuir um sistema de tolerância a falhas que permite evitar a perda de dados durante um processamento.
- **Hadoop Common:** componente que possui diversos utilitários em toda aplicação, além de bibliotecas capazes de manipular arquivos. Esse componente pode ser utilizado para disponibilizar integrações de interface para outros sistemas, como CloudSource e AWS S3.
- **Hadoop MapReduce:** componente responsável por processar grandes volumes de dados de maneira distribuída, por meio da biblioteca MapReduce, com as funções de processamento `Map` e `Reduce`.

Desafios de desempenho do Hadoop MapReduce

O Hadoop MapReduce é um paradigma de programação baseado em duas funções simples, a `Map` e a `Reduce`, presentes em diversas linguagens de programação funcionais. Essas funções ficaram mais conhecidas quando foram implementadas pelo Google no Hadoop, mostrando-se adequadas para trabalhar com problemas particionados ou fragmentados em subproblemas. Isso ocorre porque as funções `Map` e `Reduce` podem ser aplicadas separadamente a um conjunto de dados (DEAN, 2004).

O Hadoop permite a utilização de máquinas convencionais para a criação de um *cluster*. Contudo, isso pode impactar positiva ou negativamente as atividades realizadas pelo MapReduce. Esse impacto pode ocorrer em gargalos na utilização de disco, *Random Access Memory* (RAM), *Central Processing Unit* (CPU) e rede dispostos nas máquinas que compõem o *cluster*.

Durante as tarefas do MapReduce, é importante definir o número de tarefas mapeadoras, o que possibilita controlar a quantidade de mapeadores e o tamanho de cada tarefa. Dessa maneira, durante a execução, o

Hadoop divide as tarefas em partes que podem ser executadas em paralelo pelas máquinas do *cluster*. Porém, inicializar uma nova tarefa mapeadora geralmente leva alguns segundos, o que pode gerar um atraso e uma sobrecarga no *cluster*. Assim, existem diversas configurações e mecanismos que auxiliam na otimização do MapReduce (GOLDMAN *et al*., 2012). A seguir, veja algumas dicas.

- Crie tarefas que utilizem pelo menos 2 a 5 minutos de execução cada. Caso o tempo médio de uma execução seja menor do que 2 minutos, aumente o tamanho do mapeador por meio do parâmetro `mapred.min.split.size`. Isso serve para alocar menos mapeadores, reduzindo o atraso e a sobrecarga na inicialização de novas tarefas.
- Em tarefas com uma quantidade de dados superior a 1TB, considere sempre o aumento do bloco com o conjunto de dados para 256Mb ou mesmo 512Mb. Dessa maneira, você vai diminuir o número de tarefas do MapReduce.
- Opte por utilizar blocos de dados maiores para diminuir a quantidade de tarefas Java Virtual Machine (JVM), que geram sobrecarga devido à inicialização de novas tarefas durante as execuções.
- Garanta que os arquivos de entrada tenham formatos grandes, para que sejam divididos em diversos arquivos menores durante as execuções.
- Evite utilizar *Redundant Array of Independent Disks* (Raid) em máquinas com `TaskTracker` e `DataNode`, pois isso pode impactar o desempenho do *cluster* Hadoop. O Raid é uma tecnologia capaz de unir discos de armazenamento ou criar espelhamentos. Um espalhamento pode impactar consideravelmente o desempenho, pois utiliza o disco a todo momento para gerar cópias para outro disco.
- Configure os arquivos mapred.local.dir e dfs.data.dir, que apontam o diretório onde cada um dos seus discos está alocado. Dessa maneira, você garante a utilização da capacidade total das máquinas do *cluster*.
- Filtre os registros enquanto realiza tarefas mapeadoras, e não redutoras.
- Use o mínimo de combinações para formar uma chave de saída do MapReduce.
- Realize ajustes de RAM por meio do parâmetro `mapred.child.java.opts`, que permite ajustar o uso de memória máxima sem acionar uma troca no MapReduce, possibilitando a otimização das tarefas.

- Entre o mapeador e o redutor, implemente um Combiner que possibilite realizar a agregação dos dados antes que eles cheguem ao redutor. Dessa maneira, as execuções do MapReduce combinam-se de forma inteligente, reduzindo a quantidade de dados a serem gravados no disco e que devem ser transferidos entre os estágios das tarefas `Map` e `Reduce`.

Fatores que impactam a escalabilidade de processamento no MapReduce

Escalabilidade do ecossistema Hadoop

O ecossistema Hadoop permite obter escalabilidade com facilidade. Na maioria dos casos, mudanças no ambiente do *cluster* resultam em pequenas modificações em arquivos de configuração. Dessa maneira, o Hadoop pode ser um ecossistema simples tanto para um *cluster* de cinco máquinas quanto para um *cluster* de mil máquinas. Ou seja, as limitações se relacionam apenas às especificações físicas dos computadores alocados ao *cluster*.

Uma das desvantagens relacionadas à arquitetura do *cluster* Hadoop é que essa arquitetura possui somente um nó mestre alocado em uma máquina. Esse nó mestre é vital para o funcionamento do *cluster* e pode ocasionar perdas de escalabilidade caso esteja sobrecarregado, ou caso falhe durante uma tarefa escalada.

HDFS

O HDFS é o componente do Hadoop utilizado para o armazenamento dos volumes de dados no *cluster*. Essa ferramenta permite criar um sistema escalável de arquivos distribuídos, que visa a garantir a escalabilidade do sistema à medida que for necessário armazenar grandes volumes de dados em uma única máquina ou em um *cluster* de máquinas.

O HDFS possui um sistema tolerante a falhas que permite utilizar blocos de tamanhos fixos, normalmente 64Mb, que são dispostos em diversas máquinas do *cluster* Hadoop. Para o funcionamento dessa aplicação, são necessários três componentes principais. A seguir, veja quais são eles.

- `NameNode`: nó mestre responsável por armazenar os metadados, enviar tarefas e monitorar o *status* das tarefas atribuídas aos `DataNodes`.
- `DataNode`: local onde os blocos com os dados ficam armazenados. Cada `DataNode` deve realizar as tarefas e enviar relatórios de suas atividades para o `NameNode`. Caso um `DataNode` pare de funcionar, o `NameNode` aloca sua atividade a outro `DataNode`.
- `SecondaryNameNode`: `NameNode` secundário que fica alocado em outra máquina e armazena pontos de checagem a cada período de tempo predefinido, garantindo a recuperação daquele ponto caso o `NameNode` principal falhe.

MapReduce

Durante uma tarefa do MapReduce, os dados são consultados, processados e armazenados em disco. Essas atividades são feitas em conjunto com o HDFS, normalmente em blocos de 64Mb. Porém, para otimizar as tarefas do MapReduce, é importante alterar esses tamanhos conforme o tamanho dos arquivos alocados. Dessa maneira, quanto menos blocos houver, menores serão as quantidades de tarefas do MapReduce.

O MapReduce tem um processo que separa suas atividades `Map` e `Reduce`. Dessa maneira, permite rodar processos em paralelo em máquinas de *cluster* Hadoop. Para compor essas atividades, o MapReduce possui diversas funções. A função `Map` é responsável por receber os dados de entrada e alocar os pares de chave e valor. Em seguida, a função `Sort` recebe os dados da função `Map` e os organiza, atribuindo entrada para cada `Reduce` associado a uma chave específica. Enquanto isso, a função `Shuffle` transfere suas saídas do `Map` para os `Reduce` como entrada inicial. A função `Reduce`, por sua vez, recebe os dados do `Shuffle` e devolve uma lista de chave e valor contendo os registros como saída das tarefas.

Uma execução do MapReduce, assim como o HDFS, possui um nó mestre, chamado `JobTracker`. Ele é responsável por dar suporte às tarefas do MapReduce por meio de coordenadas que são enviadas aos `TaskTrackers`. Já os `TaskTrackers` são responsáveis por executar as tarefas do MapReduce supervisionadas pelo `JobTracker` e informar seus progressos.

Na Figura 1, a seguir, veja a arquitetura do funcionamento do MapReduce com o HDFS. Observe que cada nó é considerado um conjunto de `TaskTracker` com `DataNode` funcionando nas atividades designadas pelo `JobTracker` e pelo `NameNode`. Em cada nó do *cluster*, fica armazenada uma quantidade de blocos com os dados no serviço `DataNode`. Desta maneira, o `NameNode` supervisiona e requisita os blocos do `DataNode` com as informações, quando necessário. Enquanto isso, o `JobTracker` supervisiona a submissão das tarefas do MapReduce aos `TaskTrackers` e monitora a execução dessas atividades.

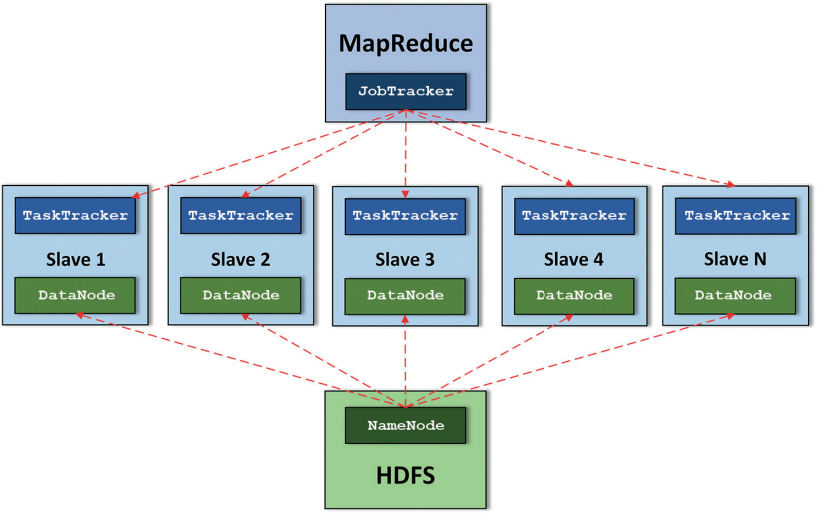

Figura 1. Arquitetura de funcionamento do MapReduce com o HDFS.

Distorção dos dados nas atividades do MapReduce

O modelo de programação MapReduce, apesar do seu alto desempenho, ainda sofre com a distorção de dados. Tal distorção ocorre invariavelmente durante a análise e afeta seriamente a eficiência das tarefas. O tempo de conclusão de um trabalho do MapReduce depende da tarefa em execução mais lenta disposta naquele trabalho. Assim, se uma tarefa em uma das máquinas do *cluster* demorar muito mais tempo do que as demais tarefas para finalizar, ela poderá atrasar o andamento de todo o trabalho.

Normalmente, a distorção de dados consiste no desequilíbrio na quantidade de dados atribuídos a cada tarefa, ou mesmo no desequilíbrio na quantidade de trabalho necessário para processar os dados. Os dados encontrados em diversas aplicações do mundo real são muitas vezes diferentes e distorcidos, o que aumenta a ocorrência desse tipo de problema. Em diversos casos, o MapReduce processa uma coleção de registros em forma de chave e valor, um por um. Esse processo, dependendo da aplicação, pode exigir mais CPU e memória do que outros registros, gerando registros maiores que resultam no atraso da entrega das atividades (KWON *et al*., 2011).

Tolerância a falhas

Um *cluster* Hadoop deve desempenhar diversas atividades para o armazenamento e o processamento de dados. Além disso, são necessários mecanismos que possam tratar falhas durante a execução. Suponha que você tem um *cluster* com apenas cinco máquinas. Caso uma máquina falhe, você vai identificá-la rapidamente, não é? Entretanto, no caso de um *cluster* com mais de mil máquinas, é necessário possuir um mecanismo que identifique a falha e a corrija durante a execução. Afinal, todo trabalho rodando em diversas máquinas pode falhar a qualquer momento devido a um componente de *hardware*, a processos que um sistema não possa executar ou mesmo a *scripts* que possuem falhas durante a execução.

Um trabalho do MapReduce inicia suas tarefas distribuindo-as entre os nós do *cluster*. Caso uma das atividades falhe em algum momento, o mecanismo do Hadoop permite que o trabalho seja recuperado a partir de um ponto e executado em outro nó até que seja concluído. Quando um `TaskTracker` de um nó, que é responsável por executar uma atividade, fica sem receber comunicação por um período de tempo, ele é marcado como falho, e sua atividade é realocada pelo `JobTracker`, que encaminha a execução para o próximo `TaskTracker` disponível. Nesses casos, o nó que parou de receber comunicação será alocado em uma lista negra e poderá ser retirado assim que o seu sistema for reiniciado e a sua capacidade retornar ao normal.

Entre as falhas mais graves, destaca-se a queda de um nó mestre, pois ele coordena a execução do `JobTracker` que controla todos os nós escravos `TaskTrackers`. Nesse caso, o Hadoop não consegue tratar a falha, e o *cluster* inteiro fica indisponível até ela ser corrigida. Na maioria dos casos, tarefas que estavam executando são perdidas e nunca mais são recuperadas.

Utilização de *combiners* e *partitions* para otimização de desempenho

Durante a utilização do MapReduce, é comum executar tarefas no mapeador que gerem grandes blocos de dados intermediários. Em seguida, esses blocos são passados para o redutor para processamentos posteriores, o que resulta em um enorme congestionamento da rede. Para solucionar esse problema, o MapReduce possui otimizações com o Partition e o Combiner.

O Partition possibilita a divisão da tabela com base nos valores de colunas específicas. Ou seja, ele verifica, durante uma consulta, as partições relevantes a serem consultadas e ignora as partições irrelevantes. Dessa maneira, agiliza o processo de consulta dos dados durante as execuções. Isto é, mesmo que o volume de dados seja gigantesco, o Partition consegue filtrar somente os dados que estão nas partições relevantes, permitindo um ganho significativo no tempo das consultas.

O Combiner no MapReduce é conhecido também como "minirredutor". Sua principal função é processar os dados que vêm da saída do mapeador antes de eles serem repassados para o redutor, ou seja, ele executa um processo entre o mapeador e o redutor.

É comum os mapeadores gerarem inúmeras chaves repetidas durante o processo. O Combiner processa os dados dos mapeadores, fazendo uma agregação dos dados que possuem chaves repetidas e reduzindo a quantidade de dados de saída. Assim, o Combiner diminui o tamanho dos dados que serão passados para o redutor e também reduz a sobrecarga da rede. Entretanto, o redutor ainda é necessário para que todos os dados que não possuem propriedades iguais sejam processados.

A Figura 2 representa um processo do MapReduce para contagem de palavras iguais no conjunto de 20 palavras listadas no Quadro 1.

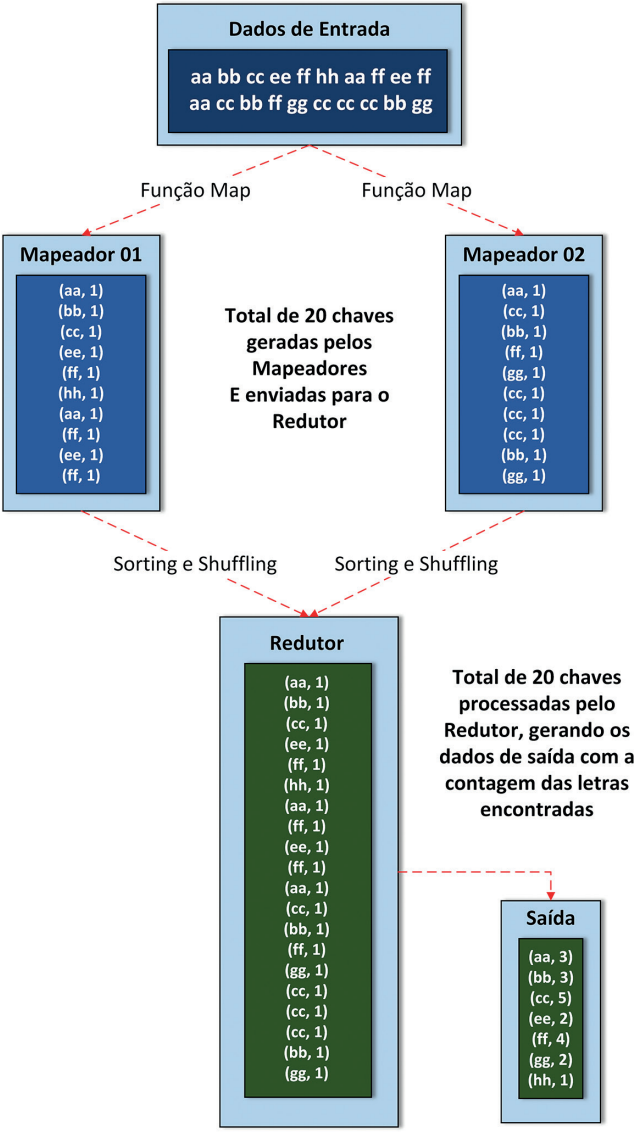

Figura 2. Processo do MapReduce sem o Combiner.

Como você viu na Figura 2, o processo `Map` é dividido entre dois mapeadores; cada mapeador recebe um conjunto de 10 palavras e faz o pré-processamento adicionando uma chave a cada palavra. Em seguida, os dados são enviados para as funções `Sorting` e `Shuffling`, que preparam os dados e os enviam ao redutor responsável por fazer a contagem das palavras iguais. Esse envio de dados ocorre através da rede e pode gerar uma grande carga na rede, ocasionando atrasos nas tarefas do MapReduce. Assim que os dados chegam ao redutor, ele faz a contagem das ocorrências e em seguida gera uma saída com a quantidade de ocorrências de cada palavra.

Quadro 1. Classe `JobMapper`

aa	bb	cc	ee	ff
hh	aa	ff	ee	ff
cc	cc	cc	bb	gg
aa	cc	bb	ff	gg

A Figura 3 representa o mesmo processo do MapReduce mostrado na Figura 2, ou seja, um processo que faz a contagem das palavras iguais no conjunto de 20 palavras listadas no Quadro 1.

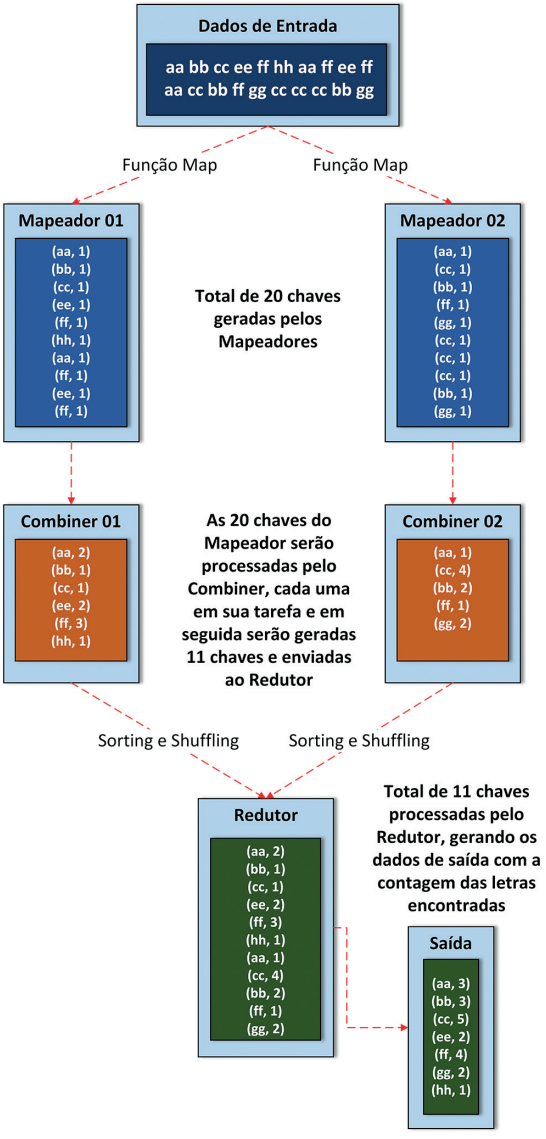

Figura 3. Processo do MapReduce com o Combiner.

Como você viu na Figura 3, o processo `Map` é dividido entre dois mapeadores; cada mapeador recebe um conjunto de 10 palavras e faz o pré-processamento adicionando uma chave a cada palavra. Nesse momento, é adicionado um Combiner para cada mapeador; o Combiner faz o pré-processamento dos dados antes de enviá-los para as funções `Sorting` e `Shuffling`. Note que ele soma as palavras repetidas e já adiciona os valores conforme a ocorrência das palavras. Esse processo é tão importante, que diminui consideravelmente a carga antes gerada na rede para o envio de todos os dados. Ou seja, agora, em vez de receber os 20 dados, o redutor vai receber apenas 11 dados, verificar suas ocorrências e gerar uma saída com o resultado.

Saiba mais

A redução que o Combiner proporciona em um conjunto de dados pequeno pode parecer irrelevante. Entretanto, em um cenário de *big data*, é comum realizar processos que ficam semanas rodando em bases de dados. Ou seja, uma diminuição de 30% de dados trafegados na rede pode resultar em horas de redução de tempo de processamento.

O exemplo a seguir apresenta o pseudocódigo utilizado para realizar os testes das Figuras 2 e 3. Lembre-se de que o processo da Figura 2 está sem o Combiner, que é chamado por meio do comando `jobex.setCombinerClass(ContadorDePalavrasReducer.class);` (THE APACHE SOFTWARE FOUNDATION, 2020).

```
// Importa as bibliotecas do Java

package org.example.wordcount;


import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import   org.apache.hadoop.mapreduce.lib.output.
FileOutputFormat;

import   org.apache.hadoop.mapreduce.lib.output.
TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;


public class ContadorDePalavras {

       public int run(String[] args) throws Exception {

              if (args.length != 2) {

                     System.out.println("Usage:  [input]
[output]");

                     System.exit(-1);

              }


              Job jobex = jobex.getInstance(getConf());

              jobex.setJobName("wordcount");


              // Chama a Classe Main (ContadorDePalavras)

              jobex.setJarByClass(ContadorDePalavras.class);


              // Especifica os formatos dos dados em Text e
Inteiro

              jobex.setOutputKeyClass(Text.class);
```

```
        jobex.setOutputValueClass(IntWritable.class);

        // Chama a Class Mapper
        jobex.setMapperClass(ContadorDePalavrasMapper.
class);

        // Chama a Class Combiner
        jobex.setCombinerClass(ContadorDePalavrasRedu
cer.class);

        // Chama a Class Reducer
        jobex.setReducerClass(ContadorDePalavrasRedu
cer.class);

        jobex.setInputFormatClass(TextInputFormat.
class);
        jobex.setOutputFormatClass(TextOutputFormat.
class);

        Path inputFilePath = new Path(args[0]);
        Path outputFilePath = new Path(args[1]);

        // São informados os dados de forma recursiva
        FileInputFormat.setInputDirRecursive(job, true);
        FileInputFormat.addInputPath(job,
inputFilePath);
        FileOutputFormat.setOutputPath(job,
outputFilePath);
```

```
                // Caso já exista algum arquivo de saída, ele
será excluído
                FileSystem fs = FileSystem.newInstance(getConf());
                if (fs.exists(outputFilePath)) {
                        fs.delete(outputFilePath, true);
                }

                return jobex.waitForCompletion(true) ? 0: 1;
        }
}


public class ContadorDePalavrasMapper extends
                        Mapper<LongWritable, Text, Text, In-
tWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context
context)
                        throws IOException, InterruptedException
{
                String line = value.toString();

                StringTokenizer tokenizer = new
StringTokenizer(line);
                while (tokenizer.hasMoreTokens()) {
                        word.set(tokenizer.nextToken());
```

```
                    context.write(word, one);
             }
      }
}

public class ContadorDePalavrasReducer WordcountExampleReducer extends
             Reducer<Text, IntWritable, Text, IntWritable>
{

      private IntWritable totalPalavrasContadas = new IntWritable();

      public void reduce(final Text key, final Iterable<IntWritable> values,
                    final Context context) throws IOException, InterruptedException {

             int totalContados = 0;
             Iterator<IntWritable> iterator = values.iterator();

             while (iterator.hasNext()) {
                    totalContados += iterator.next().get();
             }

             totalPalavrasContadas.set(totalContados);
             context.write(key, totalPalavrasContadas);
      }
}
```

Referências

DEAN, J.; GHEMAWAT, S. *MapReduce:* simplified data processing on large clusters. In Communications of the ACM, 2008.

GOLDMAN, A. *et al. Apache Hadoop:* conceitos teóricos e práticos, evolução e novas possibilidades. 2012. Disponível em: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/JAI_2012_Cap%203_Apache%20Hadoop.pdf. Acesso em: 27 out. 2020.

KWON, Y. *et al.* A study of skew in MapReduce applications. *In:* OPEN CIRRUS SUMMIT, 5., 2011, Moscow. *Proceedings* [...]. Moscow: [*s. n.*], 2011,

THE APACHE SOFTWARE FOUNDATION. *MapReduce tutorial*. 2020. Disponível em: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. Acesso em: 27 out. 2020.

Leituras recomendadas

CAPRIOLO, E.; WAMPLER, D.; RUTHERGLEN, J. *Programming hive:* data warehouse and query language for Hadoop. [*S. l.*]: O'Reilly", 2012.

DU, D. *Apache hive essentials*. Birmingham: Packt, 2015.

JEYARAJ, J.; PUGALENDHI, G.; PAUL, A. *Big data with Hadoop MapReduce:* a classroom approach. Burlington: CRC, 2020.MINER, D.; SHOOK; A. *MapReduce design patterns:* building effective algorithms and analytics for Hadoop and other systems. [*S. l.*]: O'Reilly", 2012.

WHITE, T. Hadoop: the definitive guide. 4th ed. [*S. l.*]: 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