FUNDAMENTOS DE BIG DATA

Fernanda Rosa da Silva

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Paradigma de programação do MapReduce

OBJETIVOS DE APRENDIZAGEM

- Resumir o histórico do MapReduce e sua utilização.
- Definir como o paradigma de programação do MapReduce pode ser utilizado para análise de dados e sua adequação para processamento distribuído.
- Apontar programas em MapReduce para extração de informação por meio de pseudocódigos.

Introdução

Devido ao avanço da tecnologia e à popularização da internet, o compartilhamento de informações cresceu muito nos últimos anos. Por isso, a complexidade dos serviços disponibilizados por meio das redes de computadores tem aumentado, assim como a quantidade de dados processados por essas aplicações. Como você sabe, processar dados requer grande capacidade de armazenamento, mas outros fatores críticos, como desempenho e disponibilidade, também são fundamentais para que se possa atender às requisições da melhor forma.

Neste capítulo, você vai estudar o MapReduce, um modelo de programação utilizado para gerar processamento distribuído. O MapReduce é utilizado para realizar análises de dados em grande escala. Ao longo do capítulo, você vai conhecer os conceitos principais e as características desse modelo.

Surgimento do MapReduce e sua utilização

O MapReduce surgiu da grande necessidade de processar dados que foi tomando forma nos sistemas atuais. Ele é definido por um conjunto de bibliotecas que permite realizar o processamento de grandes quantidades de dados de forma paralela, com o uso de *hardware* de propósito geral e tecnologia de *cluster*. O MapReduce permite que os dados sejam distribuídos com base em suas fases de processamento.

Com o uso do MapReduce, é possível obter milhares de arquivos de *logs* com informações relevantes e validar dados a fim de chegar a um resultado específico. A essência do MapReduce pode ser definida como o uso de algoritmos empregados em diversas situações com foco em solucionar problemas muito comuns na área da tecnologia, como tarefas de busca e otimização de dados (normalmente presentes em sistemas de tomada de decisão), processamento de dados em grande escala e manipulação de dados.

A partir da aplicação desse paradigma, de acordo com Pan, Cruz e Vasconcellos (2012), é possível simplificar estruturas complexas compostas por milhares de dados. Além disso, é possível determinar soluções para problemas de agendamento e planejamento de recursos, de organização de informações, entre outros.

Histórico do MapReduce

A evolução do MapReduce ocorre desde a sua origem (GOLDMAN *et al.*, 2012). A seguir, veja alguns marcos históricos importantes do desenvolvimento do MapReduce.

- **1965:** Gordon E. Moore fez uma análise acerca do número de transistores em circuitos integrados, verificando que esse número dobraria em determinado período de tempo. Por meio desse estudo, surgiu a lei de Moore, utilizada para medir diversas tecnologias e suas capacidades de processamento e memória. Com o tempo, a necessidade de aumentar a velocidade do processamento de dados, assim como sua proporção em sistemas computacionais, acentuou-se, o que ampliou as demandas por dados e provocou uma grande evolução na forma como os dados seriam disponibilizados aos usuários. Porém, dividir uma tarefa e executá-la por meio de unidades paralelas não seria uma missão fácil.
- **2003:** nesse período, a Google se viu diante da necessidade de aplicar melhorias à sua ferramenta de busca para definir uma técnica mais

efetiva que permitisse processar e analisar sua base de dados. Logo, surgiu o paradigma do MapReduce, sugerido pela própria Google. O MapReduce é um modelo de programação paralela para processamento largamente distribuído de grandes volumes de dados. Ele proporcionou a otimização da indexação e da catalogação dos dados que envolvem as diversas páginas *web* e suas relações (DEAN; GHEMAWAT, 2004). Com o uso do MapReduce, tornou-se possível processar dados fragmentados, como pelo Hadoop Distributed File System (HDFS), utilizando diversos computadores para processar as informações como se fossem um só. Além disso, o sistema de busca ficou mais rápido. No mesmo ano, surgiu um sistema de arquivos denominado Google File System (GFS), definido como um sistema de arquivos distribuído criado para suportar, armazenar e processar o grande volume de dados da Google, intermediando a função da tecnologia MapReduce.

- **2004:** após a criação do modelo de MapReduce no ano anterior e da sua aplicação interna aos sistemas, os autores (funcionários da Google) publicaram um artigo apresentando conceitos sobre o modelo, mas sem informar como essa implementação poderia ser aplicada.
- **2005:** um projeto de implementação do MapReduce em conjunto com o uso da tecnologia GFS foi divulgado. Essa implementação fazia referência ao subprojeto Apache Nutch (Lucene), com a intenção de criar um motor de busca na *web*, mais especificamente um rastreador de páginas *web* (*web crawler*) e um analisador de formato de documentos (*parser*).
- **2006:** a partir desse ponto, o MapReduce passa a ser aprimorado, utilizado em conjunto com outras tecnologias e implantado em grandes projetos. Ele agrega velocidade de processamento a grandes centros de dados e funciona como uma solução para multinacionais de grande porte que passaram a utilizar um grande aglomerado de máquinas para comportar seu conjunto de dados. O modelo passou a ser implementado em diferentes linguagens.

Exemplo

Um exemplo de uso é descrito por White (2009): a aplicação do MapReduce à indústria de óleo e gás. Suponha que seja necessário obter milhares de arquivos de *log* em formato de texto com informações sobre fatores como pressão e temperatura de um poço. Tais *logs* serão utilizados para mapear e validar uma grande quantidade de dados, usando uma linguagem específica.

Nesse caso, seria possível criar um programa controlador capaz de analisar todos os arquivos, obtendo como resultado o maior pico de temperatura e

pressão atingido. O MapReduce poderia ser utilizado em conjunto com diversas linguagens de programação, e o programador definiria a lógica a ser utilizada para validar o conjunto de dados.

Uma infinidade de campos pode ser beneficiada com o uso do MapReduce, como é o caso da inteligência computacional. A inteligência computacional atua com grandes bases de dados para buscar informações e localizar valores a fim de tomar uma decisão com o uso de algoritmos, que exigem uma alta taxa de processamento. Além disso, gera um alto custo para uma única máquina. Por isso, adotar o MapReduce seria ideal nesse caso.

Um exemplo real citado por Araújo (2014) é o sistema responsável pela geração e pela distribuição de energia elétrica, que gera um grande volume de dados (em alguns casos, aproximadamente 15 TB de dados por ano). Outro exemplo, citado por Dean e Ghemawat (2004), considera sistemas de indexação de páginas *web* de grande porte, como o que compõe as páginas da Google. Esses sistemas processam mais de 20 TB em documentos a cada iteração com os usuários. Casos como esses incentivam a exploração de paradigmas de programação como o MapReduce.

Como você viu, o MapReduce foi desenvolvido como uma solução para o processamento de dados em grande volume. Ele distribui tal processamento em partes com o uso de uma infraestrutura composta por muitos computadores, garantindo que o processamento seja efetivado em tempo aceitável. Para isso, aplica às máquinas as mesmas configurações, encarregando cada uma delas de se dedicar a um conjunto de dados diferente.

A principal característica do MapReduce é o seu poder de se adaptar a qualquer complexidade e qualquer volume de dados. Ele pode ser definido como uma técnica capaz de abstrair detalhes de paralelização e distribuir dados atendendo a qualquer aplicação que necessite operar com base nesse aspecto com alta escalabilidade e em ambiente de alta complexidade.

Na próxima seção, você vai conhecer as funções `Map` e `Reduce` a partir de sua aplicação a alguns cenários comuns. Você também vai ver como resolver problemas que podem ocorrer no tratamento de grandes volumes de dados tratados de forma distribuída.

Paradigma de programação do MapReduce

O MapReduce é um paradigma de computação desenvolvido para que o processamento de dados em grande escala seja possível em uma estrutura formada por centenas de computadores. Ele é, de acordo com Miner e Shook (2012), um sistema extraordinariamente poderoso.

Por esse motivo, esse paradigma permite ajustar uma solução à estrutura e à raiz do problema, utilizando a ideia principal de mapear e reduzir. Todavia, algumas situações ainda são desafiadoras, por isso é importante definir se a solução aplicada torna a resolução de problemas mais fácil ou mais difícil em cada situação.

A capacidade de uso do MapReduce, ainda de acordo com Miner e Shook (2012), tem limites claros. É necessário avaliar o que pode ou não ser feito em um ambiente, direcionar a melhor maneira de resolver um problema e desenvolver uma solução recursiva para a situação tratada por meio da utilização de recursos de rede, *clusters* e outros componentes que permitem a tolerância para as informações tratadas.

O MapReduce comprovou sua eficiência ao trabalhar com problemas fragmentados, transformando questões complexas em problemas menores e mais fáceis de serem solucionados. Como a sua denominação indica, o MapReduce se resume a duas funções: `Map` e `Reduce`, que podem ser aplicadas separadamente a um conjunto de dados ou trabalhar de forma que suas funcionalidades sejam aplicadas a diversas etapas com o mesmo objetivo.

As funções `Map` e `Reduce` indicam de que forma o mapeamento e a redução dos dados serão realizados, resolvendo problemas de comunicação e permitindo a troca de informações, a tolerância a falhas e o processamento de dados distribuídos. O principal benefício desse modelo é o paralelismo: não há etapas de processamento e tudo é feito simultaneamente. Cada componente é utilizado para atribuir diferentes funções, o que é diferente de tratar cada conjunto de dados em uma etapa específica da computação.

Assim, o MapReduce realiza a divisão de cargas de dados permitindo, ao mesmo tempo, que todas as atividades sejam realizadas de forma paralela e que cada componente se torne responsável por processar um conjunto específico de dados. O uso desse paradigma de programação permite realizar a filtragem e a classificação de dados utilizando um método de redução que executa operações como a organização do

sistema de processamento por meio do uso de servidores distribuídos. Desse modo, ocorre a execução de várias tarefas de forma paralela, bem como o gerenciamento de toda a comunicação entre os servidores e da transmissão de dados.

Essa característica também implica a possibilidade de recuperação de falhas parciais de servidor ou armazenamento durante a operação. Caso o mapeador ou o redutor falhe, o trabalho pode ser reprogramado do ponto em que parou.

A estrutura do MapReduce

Como você viu, a base das aplicações fundamentadas no MapReduce inclui conjuntos de particionamento e processamento que utilizam duas funções principais: `Map` e `Reduce`. A primeira delas armazena os arquivos como entrada, e esses blocos podem ser processados em paralelo com o uso de vários computadores em um *cluster* de computação. Como saída, a função `Map` produz tuplas (chave, valor), enquanto a função `Reduce` é responsável por fornecer o resultado final da execução da aplicação.

O modelo MapReduce fornece uma interface entre a implementação de um sistema distribuído e os usuários do sistema, que geram os dados de entrada. Mesmo assim, o sucesso do modelo, de acordo com Pan, Cruz e Vasconcellos (2012), depende da implementação correta do sistema distribuído. A ideia é prover todos os requisitos necessários para o atendimento das expectativas e a execução das funções. Além disso, deve-se aplicar tolerância a falhas e permitir que o sistema de arquivos e todos os protocolos de comunicação e processos se comuniquem da maneira correta.

Para dar início às etapas estabelecidas pelo MapReduce, as funções são acionadas na ordem apresentada a seguir.

1. Os dados de entrada são divididos em segmentos.
2. O programa é inicializado no conjunto de máquinas que executam as funções atribuídas para a aplicação do MapReduce.
3. Uma das máquinas é selecionada como mestre e as outras máquinas são selecionadas como trabalhadores (escravos). O mestre escolhe trabalhadores livres e lhes envia tarefas de forma distribuída.
4. Os escravos que executam a função `Map` mantêm os valores intermediários na memória e os gravam periodicamente no disco local, notificando assim o computador principal (mestre) a respeito da localização dos pares intermediários de chave e valor.

5. Os escravos que executam a função `Reduce` obtêm a autorização para iniciar a tarefa por meio do mestre e também recebem informações sobre o local onde está escrito o valor relacionado à tarefa de redução.
6. A máquina de trabalho responsável pela redução de dados recebe os parâmetros de entrada e escreve seu valor de saída no final do arquivo de saída da redução.
7. O mestre desbloqueia o programa do usuário, retornando com o resultado final.

Você pode ver o processo na Figura 1, a seguir.

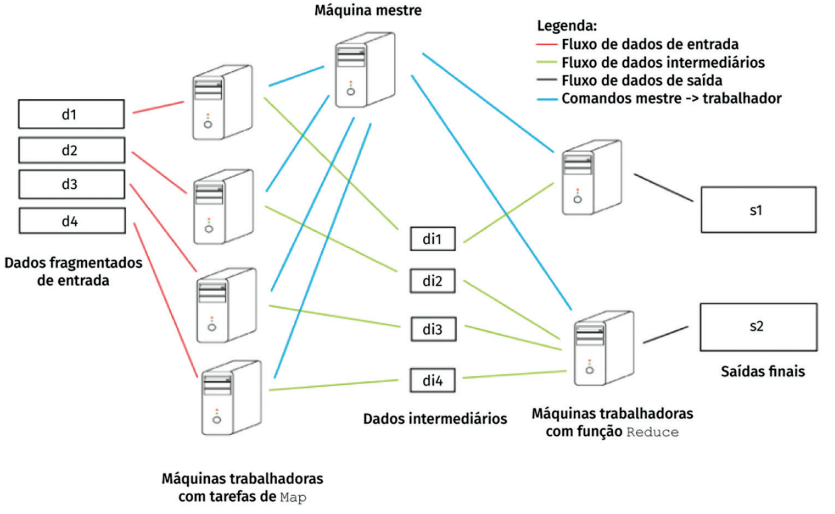

Figura 1. Modelo de funcionamento do paradigma do MapReduce.
Fonte: Adaptada de Pan, Cruz e Vasconcellos (2012).

O sistema MapReduce tolera falhas fazendo com que o mestre envie e receba notificações por meio dos escravos. Se alguma das máquinas passa muito tempo inativa, o mestre para de designar tarefas a ela e redireciona as tarefas para outra máquina. O mestre também é responsável por manter um arquivo sobre o estado da execução geral do MapReduce. Caso o mestre falhe, outra máquina pode ser eleita e iniciar a execução no ponto em que a anterior falhou.

Na Figura 2, veja a etapa inicial de entrada de dados e a distribuição das informações (os dados são armazenados no sistema de arquivos distribuído, sendo divididos em blocos menores, assim as tarefas são atribuídas a cada

um dos componentes). Note também as etapas de mapeamento (função `Map`) e agrupamento de tarefas, bem como a forma como as tarefas são disponibilizadas para a redução (função `Reduce`). Por último, atente à geração do resultado na saída do fluxo.

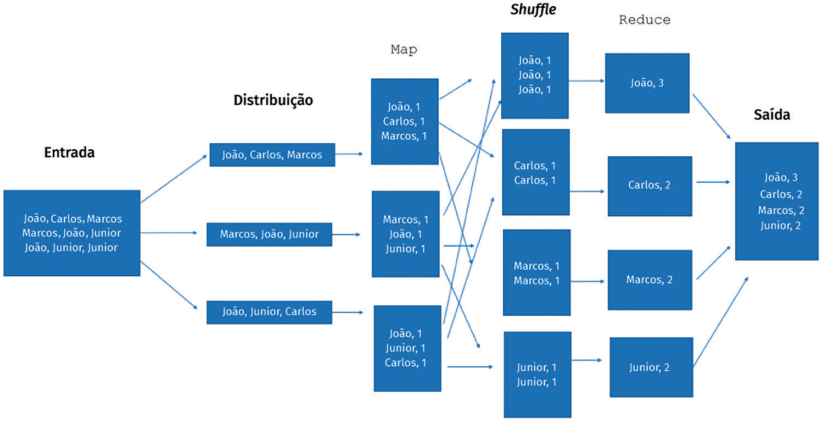

Figura 2. Exemplo de um processo MapReduce para contagem de palavras.

Para entender a Figura 2, confira a seguir a descrição das fases do paradigma do MapReduce.

Entrada

A entrada para um trabalho MapReduce é um conjunto de arquivos de armazenamento distribuídos com o uso de HDFS, por exemplo. Na entrada, a forma como os dados serão distribuídos é definida, e os arquivos são separados e carregados posteriormente por uma tarefa que ocorre no `Map`.

Distribuição

Nessa etapa, um leitor de registro é utilizado para converter a divisão dos dados gerada pelo formato de entrada em registros que possam ser analisados e repassados para o mapeador. Para cumprir sua função, o conjunto de dados mapeados se baseia em tuplas no formato de chave e valor. Nesse contexto, a chave é a informação posicional e o valor representa os dados que compõem um registro (ANDRADE, [2012?]).

A tupla utilizada pelo MapReduce recebe um valor para que ele possa ser agrupado ou filtrado, produzindo uma saída de processamento como resultado. Isso serve para abstrair a complexidade da paralelização de uma aplicação com o uso de suas funções principais: `Map` e `Reduce`.

`Map`

Na função `Map`, cada nó atuante sob a carga de trabalho a ser processada aplica a função de mapeamento aos dados locais e grava a saída em um dispositivo ou local de armazenamento temporário, garantindo que apenas uma cópia dos dados de entrada seja processada. Nessa etapa, diversas tarefas de mapeamento são realizadas de forma paralela.

No mapeador, o código fornecido pelo usuário, que representa a entrada dos dados, é executado em cada par de chave e valor, produzindo zero ou mais novos pares de chave e valor, chamado de "valor intermediário". A chave define quais são os dados a serem agrupados, e o valor define a informação que será analisada pelo redutor, retornando somente os dados pertinentes.

Essa função faz a leitura de arquivos contendo os dados armazenados como entrada, sendo capaz de ler o arquivo e realizar a filtragem e a geração de tuplas em seu formato padrão (chave e valor). Após receber o arquivo de entrada, aplica a ele a função e gera uma nova lista entregue na saída dos dados. Os tipos de dados de entrada e de saída da função `Map` podem ser (e geralmente são) diferentes uns dos outros.

Em um contador de palavras, por exemplo, a função `Map` quebra cada linha em palavras, e a saída de um par formado por chave e valor é gerada para cada palavra encontrada. Além disso, o número de ocorrências de cada palavra na linha aparece como valor.

Exemplo

Quando uma lista é recebida e um fator multiplicador é aplicado a ela, dobra o valor de cada elemento:

```
Map ({2,4,8,16}, (x2)) > {4,8,16,32}
```

Observe que o multiplicador 2 foi aplicado à lista completa, gerando os valores dobrados como saída.

Agora veja como essa função poderia ser atribuída como "dobro" de forma expressa:

```
Map ({2,4,8,16}, dobro) > {4,8,16,32}
```

Shuffle

O objetivo dessa etapa, de acordo com Miner e Shook (2012), é agrupar chaves equivalentes para que seus valores possam ser iterados facilmente na tarefa de redução, gerando a tupla de chave e listando os valores recebidos por ela. O nó intermediário redistribui os dados recebidos pelo mapeador na etapa anterior, de acordo com a chave de saída (gerada pela função `Map`), de forma que todos os dados pertencentes a uma chave estejam localizados no mesmo nó. Assim, os dados são agrupados e depois disponibilizados para a função `Reduce`.

A etapa *shuffle* (ou "fase de *shuffle*"), ilustrada no contador de palavras que você viu anteriormente, é iniciada a partir do momento em que cada tarefa mapeada e dividida entre os nós passa a produzir tuplas e chaves intermediárias e é dividida novamente, considerando os dois passos a seguir.

- **Particionamento:** as chaves e tuplas geradas são divididas em partições para serem enviadas para a próxima fase (redução).
- **Ordenação:** as chaves e tuplas pertencentes à mesma partição são ordenadas para que sejam processadas mais facilmente na etapa posterior, em que várias chaves serão processadas em uma única chamada. Assim, são processados todos os valores associados a determinada chave.

Fique atento

Na fase de *shuffle*, os valores são agrupados produzindo um conjunto de tuplas. Os valores de uma chave determinada são associados para que uma lista seja criada. Essa fase realiza a troca de dados entre as outras, entrada/saída (E/S).

`Reduce`

O redutor tem como objetivo tratar os dados agrupados como entrada e executar uma função de redução através do agrupamento de chaves. Na fase representada pela função `Reduce`, cada chave é recebida em conjunto com um objeto de iteração que representa um valor e é associado a essa chave, dando origem à tupla (MINER; SHOOK, 2012).

Nessa fase, é possível que os dados sejam agregados, filtrados ou combinados de várias maneiras para sumarizar um grande volume de dados. Apesar de funcionar de forma semelhante à função anterior, essa função

retorna um único valor como saída em resposta à lista recebida, reduzindo seu resultado e aplicando algumas funções, como `mínimo`, `máximo` e `média`, todas contendo um resultado baseado nos valores apresentados e em sua combinação. O resultado coletado por `Map` é recebido pela função `Reduce` com o objetivo de sumarizar os valores, uma vez que eles são recebidos e agrupados pela chave (ANDRADE, [2012?]).

A função `Reduce` fornece um valor como resultado final da execução de uma aplicação. Inicialmente, a implementação aplica uma função `Map` a um conjunto de valores e, a partir do seu resultado, aplica a função `Reduce`, gerando a saída final para o sistema de computação distribuída, que é controlada por um *framework* que utiliza seu sistema de arquivos em conjunto com protocolos de comunicação. Tais protocolos permitem a troca de mensagens entre as máquinas que compõem o *cluster*.

Exemplo

Analise um exemplo que mostra como a função `Reduce` pode ser aplicada a um cenário simples, retornando em sua saída os valores que se referem ao resultado específico, garantindo valores correspondentes ao que se busca na base de dados:

`>` representa a saída;

`Reduce ({1,2,3,4}, mínimo) > 1` retorna o menor valor;

`Reduce ({6,8,12,18}, máximo) > 18` retorna o maior valor;

`Reduce ({2,4,5,8}, média) > 4,75` retorna a média, somando as entradas e dividindo pelo número de valores declarados.

Saída

O formato de saída traduz o par chave e valor final gerado pela função de redução e grava o resultado em um arquivo final, separando a chave e o valor. Como exemplo, considere uma loja que necessita calcular os lucros realizando a contagem do dinheiro entrante em determinando espaço de tempo. Esse tipo de análise permite avaliar os valores, extraindo somente os dados que requerem atenção.

O MapReduce pode ser utilizado para solucionar qualquer problema de otimização que exija boas respostas em situações que normalmente são difíceis de serem resolvidas com aplicações tradicionais em uma única máquina.

Aplicações do MapReduce

Quando os dados são tratados em grandes volumes, em sistemas em que informações são adicionadas e atualizadas a todo momento, é mais complexo

olhar para um conjunto de dados e localizar um registro manualmente. O uso de MapReduce facilita esse processo. Diversas ações são viáveis por meio do uso desse paradigma, como calcular valores, avaliar dados e analisar um conjunto de dados para extrair um resultado específico cuja obtenção seria complexa ou até mesmo impossível de forma manual.

A tecnologia MapReduce pode ser aplicada a muitos campos, como agrupamento de dados, aprendizado de máquina, visão computacional, inteligência computacional e, especialmente, algoritmos genéticos. Neste último campo, grandes bancos de dados devem ser processados para que se possa localizar o valor de decisão.

Essas aplicações geralmente envolvem altos custos de processamento para serem executadas em uma máquina, o que torna o MapReduce uma escolha ideal. A seguir, veja algumas aplicações permitidas pelo paradigma de MapReduce (ANDRADE, [2012?]).

Exemplo

Um *site* registra informações cada vez que um usuário faz *log-on* em sua interface. Para analisar o *log* de uso e as contas que utilizam o serviço, seria inviável ler uma quantidade de arquivos definida em terabytes utilizando um leitor comum de texto.

Para facilitar o processo, os *log-ins* podem ser agrupados por hora e dia, e a contagem do número de acessos pode ser realizada para cada grupo. Assim, fica mais fácil reconhecer os resultados, determinando a melhor segmentação para categorizá-los e filtrá-los. Isso pode ser feito com o uso do MapReduce.

A seguir, veja outras aplicações e funcionalidades do MapReduce.

- **Agrupamento de dados:** em alguns casos, é necessário agrupar informações que contenham um mesmo valor de função armazenadas em um arquivo e processar essas informações em outro programa que requer que esses dados sejam processados como um único grupo. O MapReduce pode ser utilizado nesse cenário.

Exemplo

Considere como exemplo a construção de índices reversos, recurso utilizado para buscar padrões no final do texto em bancos de dados relacionais. Esse recurso é implementado de forma nativa na maioria dos bancos. Aplicando MapReduce, as tarefas de mapeamento calculam uma função para cada um dos itens e emitem o valor dessa função em uma só chave, reduzindo

todos os itens envolvidos no agrupamento a somente um valor para a função específica, processando e salvando o resultado. No caso de índices reversos, a função é identificar um termo baseado em dados armazenados com o uso de códigos reversos, facilitando a busca de valores.

- **Filtro, análise e validação:** o MapReduce é utilizado em sistemas em que existe um grande registro de informações para análise de *logs*, consulta e validação de dados. Nesses sistemas, é necessário coletar todos os registros que condizem com o que se busca, com a intenção de transformar esses registros em um único arquivo. A função `Map` obtém os registros, e a função `Reduce` entrega os itens recebidos após serem transformados.
- **Execução de tarefas distribuídas:** o MapReduce é útil quando uma grande quantidade de dados precisa ser tratada em um recurso computacional. Sendo assim, pode ser dividido em múltiplas partes, armazenando todos os dados de entrada, mapeando-os e executando os cálculos necessários para emitir o resultado esperado.

Exemplo

Uma grande empresa financeira trata muitos dados em diversas cidades. Para isso, trata os dados de forma distribuída, combinando todas as partes emitidas para obter o resultado final.

O MapReduce possibilita o uso de suas funções, agregadas à flexibilidade e à funcionalidade, em diversos contextos computacionais. Seus elementos geralmente são utilizados em bases de grande volume de dados para realizar análise de informações, coletar dados que interessam a determinado sistema e filtrar entradas a fim de que os resultados sejam resumidos em saídas menores.

Com o uso do MapReduce, entradas são tratadas pelas funções, realizando o agrupamento correspondente ao que é esperado para garantir que se possa exibir somente informações relevantes. A ideia é manter o desempenho da operação e coletar dados que podem estar distribuídos em diversas bases com a mesma funcionalidade.

MapReduce e uso de pseudocódigos

Agora você vai estudar alguns pseudocódigos que podem ser utilizados em casos mais comuns. Programas escritos em MapReduce geralmente possuem uma estrutura pequena, composta por códigos breves e com poucas variações. Por isso, de acordo com Gasparotto (2014), é possível escrever programas MapReduce utilizando um modelo já existente e somente modificando partes específicas, de acordo com a função desejada pelo programador. Em síntese, a ideia é usar como base um programa simples e moldar novas funcionalidades em sua estrutura.

Pode-se acrescentar que somente uma classe define completamente os *jobs* do MapReduce. Essa classe será executada na máquina cliente, enquanto as máquinas que compõem o *cluster* rodam todas as outras — denominadas "`Mapper`" para a função `Map` e "`Reducer`" para a função `Reduce` —, que estarão executando em vários nós diferentes.

Além disso, outra grande vantagem dessa abordagem é que tudo cabe em apenas um arquivo, simplificando a administração do código. Algumas das aplicações citadas anteriormente são exemplificadas nesta seção.

Contador de palavras

Esse tipo de aplicação é utilizado para buscar palavras-chave em grandes sistemas de arquivo ou em bases de dados complexas, contabilizando os valores de forma simplificada. Em um contador de palavras, a aplicação produz um documento com a chave `1` como valor, agrupando as palavras encontradas e, em seguida, aplicando a fase de redução para produzir cada palavra única. Posteriormente, as palavras únicas encontradas são somadas, e é contabilizado o número de vezes que uma palavra aparece em um conjunto de documentos de textos.

A funcionalidade pode ser resumida da seguinte forma:

- cada vez que uma palavra é armazenada, o contador é incrementado (+1);
- a função `Map` filtra todas as palavras que chegam como valor, convertendo o texto em *string* e gerando chave e valor;
- a filtragem é realizada até que existam palavras para serem contabilizadas;
- a função `Reduce` recebe como entrada chave e valor e escreve todas as ocorrências encontradas como resultado.

A seguir, veja um exemplo de `WordCount`.

Exemplo

```
public class WordCount
   {
      public static class Map extends Mapper<LongWritable,
Text, Text, IntWritable>
       {
        /* Gera o numero 1 em IntWritable que será escrito
mais à frente na <chave,valor>)  */
        private final static IntWritable one = new
IntWritable(1);

        /* Irá armazenar as palavras coletadas */
        private Text word = new Text();
        public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException
        {
              /* Converte o tipo Text para String */
              String line = value.toString();

              /* Fragmenta a linha em palavras */
              StringTokenizer tokenizer = new
StringTokenizer(line);

              // FILTRAGEM

              while (tokenizer.hasMoreTokens())
              {
                    /* Tipo de dado String armazena a pa-
lavra encontrada no StringTokenizer */
                    word.set(tokenizer.nextToken());
                    context.write(word, one);
              }
        }
        }

      /* Implementação da classe (static) Reduce que deve
(estender) Reducer */
        public static class Reduce extends Reducer<Text,
IntWritable, Text, IntWritable>
        {
        /* METODO REDUCE */
        public void reduce(Text key, Iterator<IntWritable>
values, OutputCollector<Text, IntWritable> output, Reporter
reporter) throws IOException
        {
              // Somatório
```

```
            int sum = 0;

            // CLASSIFICAÇÃO

            /* Enquanto existirem valores realiza a con-
tagem */
            while (values.hasNext())
            {
                  sum += values.next().get();
            }

            output.collect(key, new IntWritable(sum));
        }
        }

       public static void main(String[] args) throws
Exception
        {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");

        /* Tipos de dados do output */
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Vincula os métodos
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

           /* Formatos de entrada de dados */
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

           /* Caminhos no HDFS */
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
        }

}
```

Contador de *hashtags* (#)

O contador de *hashtags* pode ser utilizado para coletar as palavras-chave mais utilizadas em dado ambiente. Essa tarefa pode ser realizada com o uso de ferramentas tradicionais, como o Excel, que é mais utilizado para tratar

bases menores. Por meio do MapReduce, o programa é capaz de receber uma sentença de texto como entrada, convertendo-a em uma lista de palavras-chave e calculando a quantidade de vezes que elas foram utilizadas.

O contador de *hashtags* é bastante útil em redes sociais, em especial no Twitter. O contador funciona com base na identificação feita pela função `Map()`, ao final contabilizando as *hashtags* mais utilizadas. Os usuários utilizam as *hashtags* para categorizar determinado assunto, tornando o conteúdo pesquisável por meio de rótulos que agregam significado a ele. As *hashtags* são bastante utilizadas em publicações que coletam grande quantidade de informações, relacionadas a grandes eventos, política, desastres, esportes e entretenimento. Por meio do MapReduce, esses dados podem ser analisados.

O MapReduce faz a contagem de todas as *hashtags* utilizadas contabilizando o número de vezes que cada uma delas foi usada na rede. As funções são utilizadas para criar combinações entre campos, para que eles sejam contabilizados com a utilização das chaves implementadas. A aplicação desses filtros resulta em novas bases contendo apenas as informações importantes para determinadas análises. Na saída, as *hashtags* são listadas de forma crescente, após a redução dos resultados (MARQUESONE, 2016).

Exemplo

```
public class ContaHashtagsMapper
     extends Mapper <OBJECT, Text, Text, IntWritable>{

  private final static IntWritable numeroUm = new
IntWritable(1);
  private final Text palavra = new Text();

  @Override
  public void map(Object key, Text value, Context context)
                              throws IOException, Inter-
ruptedException {

             StringTokenizer tk = new StringTokenizer(value.
toString());
                while (tk.hasMoreTokens()) {

                 String token = tk.nextToken();
                  if(token.startsWith("#")){
                      palavra.set(token.toLowerCase()
                                  .replaceAll("[^a-Za-Z#
]", ""));
                          contexto.write(palavra, numeroUm);
                }
```

```
   public class ContaHashtagsReducer
      extends Reducer <Text, IntWritable, Text, IntWrita-
ble>  {
   private IntWritable resultado = new IntWritable();

   @Override
   public void reduce(Text key, Iterable values,
               Context context )
               throws IOException, InterruptedException {
        int soma = 0;
        for (IntWritable val : values) {
                soma += val.get();
          }
           resultado.set(soma);
           context.write(key, resultado);
       }

}
```

Redução de resultados

O desafio do modelo MapReduce não é somente armazenar e gerenciar um vasto volume de dados especificado pelo estudo de *big data*, e sim analisar e extrair um resultado de um valor gigantesco de informações, utilizando formas específicas para cada caso.

Lidando com esses desafios, o MapReduce atua no armazenamento, na recuperação e na análise correta de dados, inclusive de dados não estruturados, que, de acordo com Petry (2016), representam informações que não têm um modelo de dados específico. Todavia, essas são as informações que têm o maior crescimento atualmente.

A seguir, veja um exemplo em que a função `Reduce` é utilizada para encontrar o valor mínimo e o máximo e somar ambos. Como você viu, um contador de palavras também funciona assim.

- A função `Map` recebe todos os valores presentes na base de dados a ser analisada.
- A função `Reduce` procura os valores na base para identificar o valor mínimo e o valor máximo existentes.
- Todos os valores são identificados na entrada, e na saída o resultado define os valores conforme o mapeamento e a redução aplicada.
- Após identificar os valores, ambos são somados para resultar na contagem final.

Exemplo

```
public static class MinMaxCountReducer extends
   Reducer<Text, MinMaxCountTuple, Text, MinMax-
CountTuple> {

   // Our output value Writable
   private MinMaxCountTuple result = new MinMaxCountTuple();

   public void reduce(Text key, Iterable<MinMaxCountTuple>
values,
               Context context) throws IOException, Inter-
ruptedException {

   // Initialize our result
   result.setMin(null);
   result.setMax(null);
   result.setCount(0);
   int sum = 0;

   // Iterate through all input values for this key
   for (MinMaxCountTuple val : values) {
        // If the value's min is less than the result's min
       // S et the result's min to value's
       if (result.getMin() == null ||
                        val.getMin().compareTo(result.get-
Min()) < 0) {
                result.setMin(val.getMin());
   }

   // If the value's max is more than the result's max
   // Set the result's max to value's
   if (result.getMax() == null ||
                    val.getMax().compareTo(result.getMax())
> 0) {
                result.setMax(val.getMax());
            }

            // Add to our sum the count for value
           sum += val.getCount();

   }

           // Set our count to the number of input values
           result.setCount(sum);
           context.write(key, result);
       }

}
```

O modelo MapReduce oferece uma forma facilitada de lidar com problemas utilizando os recursos da computação distribuída. Ele é eficiente justamente por ser capaz de separar os dados. Assim, é possível livrar-se da complexidade e operar com as diversas divisões do sistema como se fossem um só recurso, utilizando a paralelização. Além disso, o MapReduce vem resolvendo um dos maiores desafios contemporâneos no âmbito da Tecnologia da Informação (TI): o armazenamento de dados distribuídos em grande escala.

A simplicidade do MapReduce é atraente para os usuários, mas é importante observar que ele tem algumas limitações em relação ao grande processamento dos mesmos dados diversas vezes, enquanto alguns algoritmos iterativos leem os dados apenas uma vez, mas requerem várias interações. Por isso, o paradigma do MapReduce representa uma enorme sobrecarga, e esse é o grande desafio da sua atuação em relação às soluções de *big data*. Dessa forma, o MapReduce resolve parte dos problemas, podendo compreender diversas máquinas que trabalham como um só nó e sendo capaz de tratar uma grande quantidade de dados com uma taxa alta de processamento.

Em síntese, o modelo de programação MapReduce é útil para os propósitos mais diversos. Isso se relaciona à sua facilidade de uso — mesmo para programadores que não estejam muito habituados a sistemas que utilizam paralelização ou trabalham com distribuição de dados. Afinal, detalhes da camada de *hardware*, como *clusters*, balanceamento de carga e tolerância a falhas, não precisam ser tratados diretamente pelo modelo.

Referências

ANDRADE, T. P. da C. de. *MapReduce:* conceitos e aplicações. Campinas: UNICAMP, [2012?]. Disponível em: https://www.ic.unicamp.br/~cortes/mo601/trabalho_mo601/tiago_cruz_map_reduce/relatorir.pdf. Acesso em: 23 set. 2020.

ARAÚJO, G. *Mapreduce:* conceitos e aplicações. 2014. Disponível em: https://pt.slideshare.net/GuilhermeArajo/mapreduce-37085946. Acesso em: 24 set. 2020.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. *In:* SYMPOSIUM ON OPERATING SYSTEM DESIGN AND IMPLEMENTATION, 6., 2004, San Francisco. *Anais* [...]. Berkeley: USENIX, 2004. p. 137-150. Disponível em: https://www.usenix.org/legacy/events/osdi04/tech/full_papers/dean/dean.pdf. Acesso em: 24 set. 2020.

GASPAROTTO, H. M. Hadoop Mapreduce: como criar um programa MapReduce base. *DevMedia*, Rio de Janeiro, 2014. Disponível em: https://www.devmedia.com.br/hadoop-mapreduce-como-criar-um-programa-mapreduce-base/30080. Acesso em: 23 set. 2020.

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: 11 set. 2020.

MARQUESONE, R. *Big Data:* técnicas e tecnologias para extração dos valores dos dados. São Paulo: Casa do código, 2016.

MINER, D.; SHOOK, A. *MapReduce design patterns.* Sebastopol, CA: O'Reilly Media, 2012.

PAN, H.; CRUZ, P.; VASCONCELLOS, P. *MapReduce:* (Big Data). Rio de Janeiro: UFRJ, 2012. Disponível em: https://www.gta.ufrj.br/grad/12_1/mapreduce/. Acesso em: 23 set. 2020.

PETRY, M. E. *Big Data e a implementação de um sistema distribuído com Apache Hadoop:* um estudo exploratório. São Leopoldo, RS: UNISINOS, 2016. Disponível em: https://www.researchgate.net/publication/318600685_Big_Data_e_a_implementacao_de_um_Sistema_Distribuido_com_Apache_Hadoop_um_estudo_exploratorio. Acesso em: 23 set. 2020.

WHITE, T. *Hadoop:* the definitive guide. Sebastopol, CA: O'Reilly, 2009.

Leituras recomendadas

DEVMEDIA. Big Data: MapReduce na prática. *DevMedia*, Rio de Janeiro, 2015. Disponível em: https://www.devmedia.com.br/big-data-mapreduce-na-pratica/32812. Acesso em: 23 set. 2020.

TUTORIALS POINT. *Hadoop:* HDFS Overview. 2020. Disponível em: https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm. Acesso em: 23 set. 2020.

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