

Recomendando  
filmes com

# Apache Mahout e Hadoop

**Usando a API de aprendizagem de máquina da Apache junto com o sistema de processamento distribuído na implementação de um sistema de recomendação escalável em um cenário de BigData.**

Um cenário de BigData bastante comum é a necessidade de se analisar Tera, PetaBytes de informação para geração de recomendações para usuários de um determinado serviço. Cada vez mais, cenários de recomendação vêm sendo explorados por serviços de provisão de conteúdo na internet e portais de e-commerce para identificação automática dos gostos de seus usuários, tendo assim um melhor desempenho na sugestão de novos itens ou conteúdos a serem consumidos por seus clientes. Uma solução viável para este cenário é a combinação da ferramenta Mahout com o Hadoop para processamento dessa massa de dados, com o objetivo de se produzir recomendações.

O Apache Mahout é uma biblioteca de aprendizagem de máquina (machine learning), de código aberto que tem como principais objetivos operar como uma máquina de recomendação, clustering e classificação. Como o nome sugere, seu projeto é gerenciado sob as determinações da Apache Software Foundation. O Mahout começou em 2008 como um subprojeto do Apache Lucene, outra ferramenta também de código aberto e gerenciada pela Apache, muito utilizada em problemas de busca e recuperação de informação. Todos esses conceitos, juntamente com clustering e classificação são adjacentes às técnicas de aprendizagem de máquina e, por isso,

grande parte do trabalho dos comitadores desse projeto, que visava mais o problema de recomendação, foi separado em um subprojeto. Mais tarde, o Mahout absorveu o projeto Taste, um projeto código aberto de filtragem colaborativa e em 2010 o Mahout se tornou um projeto Apache independente. O Mahout foi desenvolvido não só visando a escalabilidade e eficiência, mas também convertendo seus algoritmos para trabalhar, também, junto com o Hadoop. O Mahout ainda é um projeto em desenvolvimento que, embora já possua muitas técnicas e algoritmos implementados, tem ainda alguns componentes em desenvolvimento ou em fase de testes.

A proposta deste artigo é abordar conceitos básicos de recomendação e demonstrar um exemplo prático da utilização do Mahout juntamente com Hadoop na implementação de um sistema de recomendação de filmes, a partir de um conjunto de dados contendo avaliações de filmes por usuários do site MovieLens, que contribui com o projeto GroupLens, da Universidade de Minnesota, o qual realiza pesquisas diversas em Ciência da Computação, incluindo sistemas de recomendação. Como o foco é cobrir as funcionalidades de Recomendação do Mahout, os conceitos de clustering e classificação não serão abordados neste artigo.




Wellington Ramos Chevreuil | wellington.chevreuil@gmail.com

Engenheiro de Computação, trabalhando atualmente em projetos relacionados a BigData pelo Instituto Nokia de Tecnologia.



Eliza Maria Azevedo Chevreuil | elizasantoro@gmail.com

Engenheira de Computação, especialista em Desenvolvimento de Software pelo IFAM, trabalhando atualmente em projetos relacionados a recomendações pelo Instituto Nokia de Tecnologia.



*Um problema bastante comum na área de BigData é a necessidade de geração de recomendações. Existem hoje recomendações para quase tudo: novos amigos, recomendações de músicas para baixar, de produtos para comprar, fotos para ver e muito mais. Este artigo apresenta um exemplo de implementação de um sistema de geração de recomendações de filmes, utilizando como ferramentas o Apache Hadoop para processamento distribuído, escalável para cenários de BigData, e o Apache Mahout, que já provê implementações em Java de diversas técnicas e algoritmos de recomendação.*

### Conceitos básicos sobre recomendação

O objetivo principal de qualquer sistema de recomendações é apresentar algum conteúdo de interesse e relevância para o seu usuário. Logo, é necessário se ter algum conhecimento prévio sobre as preferências desse usuário, para que se possa sugerir algo similar ou que tenha relação com esse gosto. Tal cenário resume os dois problemas centrais de qualquer mecanismo de recomendação:

1. A identificação do interesse do usuário, baseado na **análise do histórico de consumo** do mesmo (que itens foram comprados, quais textos ou filmes foram avaliados).
2. O cálculo da **similaridade entre os itens a serem recomendados**, para que se consiga recomendar algo dentro do conjunto de interesses do usuário.

Como solução para esses problemas, existem hoje duas abordagens distintas de como se definir o interesse do indivíduo e como se calcular a similaridade entre os itens, que constituem os dois modelos principais: recomendação por **filtragem colaborativa** ou **baseada em conteúdo**.

A filtragem colaborativa se caracteriza pela análise do histórico do usuário, levando em consideração algum tipo de avaliação de itens feito pelo usuário. A similaridade é então calculada levando em conta apenas essa informação, ignorando completamente detalhes sobre o que são esses itens.

Já a recomendação baseada em conteúdo é totalmente voltada para a análise de informações sobre os usuários, ou sobre os itens, ou sobre os dois. Envolvem a definição de perfis dos usuários e diversas características dos elementos a serem recomenda-

### Utilizações de Filtragem Colaborativa

O fato de não se importar com o conteúdo dos itens faz com que a abordagem de Filtragem Colaborativa seja bastante genérica. Pode ser aplicada para recomendar pessoas em uma rede social, considerando os amigos como os itens, e os relacionamentos entre as pessoas seriam as avaliações dos usuários. Também pode ser usada em uma loja de livros on-line, onde os livros seriam os itens, e as avaliações seriam as próprias notas atribuídas pelos clientes aos livros, ou/e também o fato de terem ou não comprado determinado livro.

dos, para se obter o entendimento sobre os interesses dos indivíduos e a similaridade dos itens. Assim, recomendações baseadas em conteúdo são claramente restritas a cada domínio específico, pois muitas vezes depende de atributos que só existem ou fazem sentido naquele cenário determinado.

O módulo de recomendação do Mahout implementa somente algoritmos baseados em Filtragem Colaborativa, pois não há muito sentido em tentar abstrair em um *framework* cenários predeterminados de geração de recomendação baseado em conteúdo.

Com filtragem colaborativa, existem ainda duas variações principais da forma de se gerar a recomendação. Uma delas consiste em se descobrir a semelhança entre os diferentes usuários do sistema, para se recomendar os itens bem avaliados por pessoas semelhantes ao usuário ao qual a recomendação se destina. Esse modelo de filtragem colaborativa é chamado de **baseado em usuário**. O outro tipo de filtragem colaborativa busca filtrar os itens que o usuário avaliou melhor, para então identificar no sistema itens similares a esses, que então serão recomendados. Esse conceito de filtragem colaborativa é chamado de **baseado em itens**.

## Recomendações com Mahout

A versão 0.7 da API do Mahout disponibiliza tanto implementações de recomendação baseada em usuário, quanto baseada em itens. Possui uma série de classes de cálculo de similaridade entre itens e usuários, cada uma abordando um algoritmo diferente. Existe ainda um conjunto de classes voltadas para a importação dos dados no Mahout, ou seja, como os dados serão disponibilizados para os objetos que contêm a lógica específica de similaridade e recomendação em si.

Para implementações baseada em usuário, o Mahout disponibiliza a interface **UserSimilarity**. Esta interface possui várias implementações de algoritmos de cálculo de similaridade, mas o detalhamento da lógica de cada uma dessas classes não seria viável ser coberto neste artigo. Ainda para se definir a similaridade entre usuários é preciso se indicar a vizinhança, ou seja, o limite de usuários similares para se buscar itens a serem recomendados. Essa informação é representada pela interface **UserNeighborhood**. Já os dados contendo as informações a serem analisadas pelo Mahout devem ser fornecidas através da interface **DataModel**. O Mahout exige um formato específico de entrada dessas informações, o qual é mostrado na Listagem 1. Uma vez obtida as devidas implementações das interfaces anteriores, é possível criar uma instância da interface **Recommender**, a qual é usada para se obter a recomendação em si. Um exemplo simples de código para gerar recomendação baseada em usuário em um ambiente não distribuído

pode ser visto na Listagem 2. O exemplo da Listagem 2 tem como entrada o arquivo `intro.csv` que contém dados semelhantes aos da Listagem 1. Este código recomenda uma lista de itens para um usuário específico, no caso, `userId 1`. As variáveis `totalRecomendacao` usadas para escolher a quantidade de itens a serem recomendados e a variável `vizinhos` é usada para escolher quantos vizinhos serão usados para verificação da similaridade do usuário.

**Listagem 1.** Formato de dados de entrada para recomendação com Mahout: `UsrId`, `ItemId`, `Avaliação`.

1,	101,	5
1,	102,	3
1,	103,	2.5
2,	101,	2
2,	102,	2.5
2,	103,	5
2,	104,	2
3,	101,	2.5
3,	104,	4
3,	105,	4.5
3,	107,	5
4,	101,	5
4,	103,	3
4,	104,	4.5
4,	106,	4
5,	101,	4
5,	102,	3
5,	103,	2
5,	104,	4
5,	105,	3.5
5,	106,	4

**Listagem 2.** Exemplo de código de recomendação baseada em usuário.

```
public void recommendation() throws IOException {

    FileDataModel dataModel;
    long userId = 1;
    int vizinhos = 10;
    int totalRecomendacao = 5;

    try {

        URL url = this.getClass().getClassLoader().
            getResource("intro.csv");
```

```

dataModel = new FileDataModel(new File(
    url.getFile()));

UserSimilarity similarity =
    new PearsonCorrelationSimilarity(dataModel);

UserNeighborhood neighborhood =
    new NearestUserNeighborhood(vizinhos,
        similarity, dataModel);

GenericUserBasedRecommender
    itemRecommender = new
        GenericUserBasedRecommender(dataModel,
            neighborhood, similarity);

List<RecommendedItem> recommendations =
    itemRecommender.recommend(userId,
        totalRecomendacao);

for (RecommendedItem recommendedItem :
    recommendations) {
    System.out.println(recommendedItem);
}

} catch (Exception e) {
    e.printStackTrace();
}
}

```

O código da Listagem 2 terá como saída uma lista onde cada linha representa um item e o valor da similaridade, respectivamente, de itens recomendados para o usuário em questão. O valor mencionado significa o grau de similaridade de outros usuários em relação a este item, ou seja, todos os usuários que possuem os mesmos itens que ele também possuem o item 104. O exemplo desta saída segue na Listagem 3.

**Listagem 3.** Exemplo de saída do código de recomendação da Listagem 2.

```

RecommendedItem[item:104, value:5.0]
RecommendedItem[item:106, value:4.0]

```

No caso de recomendações baseadas em item, o Mahout provê a interface **ItemSimilarity**. Essa interface, assim como a **UserSimilarity**, também possui várias implementações específicas. No entanto, o conceito de vizinhança não faz sentido para recomendações baseadas em itens, logo, a respectiva instância de **Recommender** (como **GenericItemBasedRecommender**), só precisa receber um objeto **ItemSimilarity** e **DataModel**. A Listagem 5 apresenta um exemplo simples de código para geração de recomendação baseada em item, utilizando o **GenericBooleanPrefItemBasedRecommender** que estende o **GenericItemBasedRecommender** já mencionado

acima. Este código mostra uma técnica de recomendação por item utilizando a preferência booleana. Este método pode ser usado em casos em que a preferência do usuário não é relevante, sendo considerado assim, apenas a relação entre um usuário e um item. A entrada para este tipo de implementação é semelhante à mostrada na Listagem 1, porém removendo a última coluna que representa a avaliação do usuário. Um trecho desta entrada está exemplificada na Listagem 4.

**Listagem 4.** Formato de dados de entrada para recomendação booleana com Mahout: **UsrId**, **ItemId**.

1,	101
1,	102
1,	103
2,	101
2,	102
2,	103
2,	104

**Listagem 5.** Exemplo de código de recomendação baseada em item.

```

public void recommendation() throws IOException {

    long userId = 21;
    int totalRecomendacao = 5;
    FileDataModel dataModel;

    try {
        URL url = this.getClass().getClassLoader().
            getResource("input.csv");
        dataModel = new FileDataModel(new
            File(url.getFile()));

        TanimotoCoefficientSimilarity similarity =
            new TanimotoCoefficientSimilarity(dataModel);
        GenericBooleanPrefItemBasedRecommender
            itemRecommender = new
                GenericBooleanPrefItemBasedRecommender(
                    dataModel, similarity);

        List<RecommendedItem> recommendations =
            itemRecommender.recommend(userId,
                totalRecomendacao);

        for (RecommendedItem recommendedItem :
            recommendations) {
            System.out.println(recommendedItem);
        }
    } catch (Exception e) {
    }

}

```



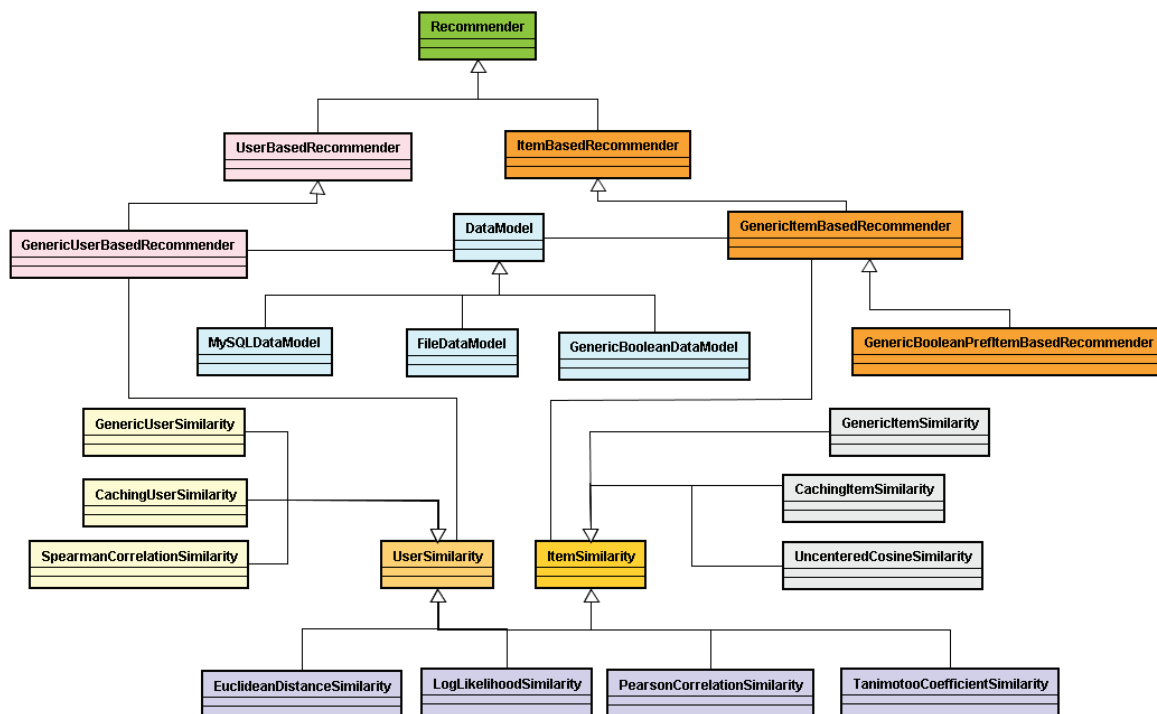
**Listagem 6.** Exemplo de saída de recomendação baseada em item.

```
RecommendedItem[item:104,
value:1.8]
RecommendedItem[item:106,
value:1.15]
RecommendedItem[item:105,
value:0.85]
RecommendedItem[item:107,
value:0.2]
```

A saída do código da Listagem 5 é demonstrada na Listagem 6. Nos dois exemplos, os resultados foram bem parecidos, apenas as métricas e a forma de calculá-las é que divergiram. No fim, os itens 104 e 106 foram os mais recomendados, embora o critério de peso tenha sido diferente. Comparando os códigos em si, fica clara a semelhança entre ambos, com o baseado em itens aparentemente mais simples, já que dispensa o conceito de vizinhança. Apesar de não ter sido o caso desse exemplo, o processamento pode levar a resultados e desempenhos distintos, dependendo de como os dados estão organizados. O tempo de execução de um algoritmo de recomendação baseado em item tende a piorar conforme o aumento do número de itens no sistema aumenta. O contrário se aplica

para o modelo baseado em usuário. Essa pode ser uma característica determinante na escolha de um modelo que tenha um melhor desempenho. No caso de recomendações baseadas em itens, um fator importante que pode contar a favor de tal modelo é que a similaridade entre itens tende a ser estável, podendo ser pré-processada.

Embora o Mahout proveja uma grande variedade de algoritmos de recomendação, não existe um algoritmo que atenda a todos os problemas. No entanto, é possível avaliar se o algoritmo escolhido é eficiente para o problema em questão. O Mahout disponibiliza para isso a interface **RecommendedEvaluator** que auxilia na avaliação dos resultados. Essa interface possui algumas implementações específicas, como, por exemplo, o **AverageAbsoluteDifferenceRecommenderEvaluator** e o **RMSRecommenderEvaluator** que provêm métodos diferentes de calcular o desempenho da recomendação (*score*). O método responsável pela avaliação é o **evaluate()**, que compara as estimativas de preferência com dados de teste. Ainda que sejam usados dados aleatórios para a avaliação de escolha de dados, o resultado é consistente graças ao **RandomUtils.useTestSeed()**, que garante que os mesmos dados aleatórios sejam usados toda vez que o teste é executado. Este método deve ser usado apenas em testes e exemplos para garantir re-



**Figura 1.** Principais classes e interfaces do mecanismo de recomendação do Mahout.

sultados repetidos, nunca para código em produção. Como resultado, o `evaluate()` gera um *score*, e quanto menor o valor desse *score*, melhor é o resultado, pois há pouca diferença entre a recomendação gerada e a esperada. É difícil diferenciar uma recomendação boa de uma ruim, por isso, é possível prover ao Mahout um *threshold* que determina esse limiar. Caso não seja passado nenhum valor, o próprio framework ficará responsável por definir um valor por usuário.

A figura 1 exibe um diagrama com as principais classes envolvidas no mecanismo de recomendação do Mahout.

## Combinando Mahout com Hadoop

Os exemplos de códigos das Listagens 2 e 5 e as classes da figura 1 representam a API de recomendação do Mahout para um sistema onde os dados das avaliações podem ser processados de maneira não distribuída. Essas classes, cada uma implementando diferentes algoritmos de similaridade, podem ser utilizadas diretamente em cenários onde todo o conjunto de dados pode ser carregado na memória do sistema para análise, de forma que ainda sobre recursos para execução do mesmo. Se não há previsão de demanda de crescimento da base de dados contendo as informações relevantes para a geração da recomendação desejada, esta é uma abordagem bem viável, pois o Mahout provê diversas variações de técnicas para cálculo de recomendações, cada qual apropriada ao tipo do modelo dos dados, sendo sua aplicação trivial, como mostrado nos exemplos.

No entanto, muitos dos serviços que se beneficiam de algum tipo de recomendação não se enquadram no cenário descrito acima. A quantidade de avaliações a ser contabilizada por usuários das principais redes sociais ou clientes de grandes lojas de varejo na internet para se calcular recomendações com certeza superam o limite de espaço da heap de um único processo Java. Nesse caso, faz-se necessária a implantação de um sistema de processamento distribuído que possa dividir a análise da massa dos dados, e o Hadoop pode ser usado como alternativa. Porém, simplesmente copiar os arquivos para o HDFS (Hadoop Distributed File System) e executar MapReduces usando algumas das classes do Mahout descrita anteriormente não funciona, pois é necessário organizar e agrupar essas informações de forma que possam ser computadas paralelamente, com relativa independência.

Na versão 0.7 do Mahout, as implementações de recomendação distribuída integrada ao Hadoop disponíveis são baseadas em item, slopeone e Alternating Least Square (als). Neste artigo será explorada a técnica de recomendação baseada em item, com utilização de cálculo de similaridade entre os itens por meio de matriz de co-ocorrência. Nesse modelo, a sugestão

de novos itens é basicamente obtida através de operações em matrizes. A similaridade entre os itens é baseada na quantidade de vezes que cada item está presente em par com outro item (co-ocorrência), ao longo das avaliações dos usuários do sistema. Como exemplo, na Listagem 7, os itens 1 e 2 foram avaliados pelo usuário 10, pelo usuário 11 e pelo usuário 15. Logo, a co-ocorrência do item 1 com o 2 nesse cenário é de 3. A co-ocorrência dos itens organizada no formato de uma matriz quadrada, como mostrado na Listagem 8, será a entrada desse mecanismo de recomendação distribuída do Mahout com o Hadoop.

**Listagem 7.** Exemplo de avaliações de itens por usuários, os quais podem ser contabilizados em uma matriz de co-ocorrência.

USUÁRIOS	ITEM 1	ITEM 2	ITEM 3	ITEM 4	ITEM 5
10	5	5		3	
11	3	5	4		
12			1	5	5
13	3				5
14		3	4	2	
15	4	1		1	

**Listagem 8.** Matriz de co-ocorrência dos itens, gerada a partir das avaliações dos usuários.

	ITEM 1	ITEM 2	ITEM 3	ITEM 4	ITEM 5
Item 1	4	3	1	2	1
Item 2		4	2	3	0
Item 3			3	2	1
Item 4				4	1
Item 5					2

O primeiro passo para gerar a recomendação para cada usuário é representar as avaliações de cada usuário como um vetor e então multiplicar esse vetor pela matriz de co-ocorrência. A Listagem 9 exibe o vetor resultante para o usuário 10.

**Listagem 9.** Vetor de preferências do usuário 10.

	ITEM 1	ITEM 2	ITEM 3	ITEM 4	ITEM 5
Usuário 10	5	5	0	3	0

O vetor resultante da multiplicação indica o peso que cada elemento representa para o usuário. Isso quer dizer que, descartando os elementos para os quais o usuário em questão já manifestou sua preferência, aqueles que possuem maior valor são os que possuem maior importância na recomendação. Só

para relembrar multiplicação de matrizes quadradas por vetores, cada elemento do vetor resultante é obtido através da soma do produto de cada elemento da linha correspondente da matriz por cada elemento do vetor. Por exemplo, o peso estimado do item 5 para o usuário 10 pode ser obtido da seguinte forma:  $(1 \times 5) + (0 \times 5) + (1 \times 0) + (1 \times 3) + (2 \times 0) = 8$ . A Listagem 10 exibe o vetor de estimativas de preferências para o usuário 10, resultante da multiplicação descrita anteriormente.

**Listagem 10.** Vetor resultante da multiplicação do vetor das avaliações do usuário 10 pela matriz de co-ocorrência.

	ITEM 1	ITEM 2	ITEM 3	ITEM 4	ITEM 5
Estimativa	41	44	21	37	8

Descartando as estimativas dos itens 1, 2 e 4, que o usuário 10 já avaliou, o item mais recomendado para esse usuário seria o 3, que é o que contém maior valor estimado. Essa técnica de operação de matriz com vetor torna muito viável o processamento distribuído porque permite que as estimativas de cada item, de acordo com a preferência de cada usuário, seja calculada de maneira independente. A partir da matriz de co-ocorrência, as multiplicações de cada linha da matriz pelo vetor de estimativas dos usuários podem ser feitas em paralelo, como tarefas de um programa MapReduce em um cluster Hadoop.

Com o Mahout, todo esse processo descrito acima é implementado em uma série de MapReduces que podem ser executados a partir da classe `org.apache.mahout.cf.taste.hadoop.item.RecommenderJob`. Ela é disponibilizada no arquivo `mahout-core-0.7-job.jar` da API do mahout, e deve ser executada, como todo programa MapReduce, usando o comando `hadoop jar`. Seu arquivo de entrada deve possuir o formato indi-

cado na Listagem 1. Mais detalhes sobre sua execução serão exibidos a seguir, na demonstração do exemplo prático.

## Exemplo Prático: Recomendando Filmes para Usuários do MovieLens

Para demonstrar o exemplo prático mostrado neste artigo, utilizamos uma base de dados para os experimentos que foram obtidos no site do GroupLens. Esse conjunto é apenas uma amostra dos dados do MovieLens, disponibilizada gratuitamente para fins de estudos (o link do MovieLens com a base usada pode ser encontrado nas referências do artigo). No link indicado, é possível baixar um arquivo .zip contendo alguns arquivos, com informações sobre os filmes e com as avaliações de alguns usuários sobre filmes. O arquivo contendo as avaliações é o `ratings.dat`, o qual deve ser copiado para o HDFS, usando o comando `put` ou `copyFromLocal` do Hadoop. O formato desse arquivo diverge um pouco do formato de entrada do Mahout, descrito na Listagem 1. Logo, é necessário converter o arquivo do MovieLens, o que pode ser implementado como um mapper para ser executado no Hadoop. O código deste mapper é exibido na Listagem 12, e o comando com a sua execução pode ser visto na Listagem 13. A implementação da classe driver foi omitida neste artigo, pois a mesma trata apenas de configuração do mapper e das suas respectivas entradas e saídas. Mais detalhes sobre implementação de classes drivers podem ser obtidos nos artigos anteriores sobre Hadoop e MapReduces, indicados na caixa “Para Saber Mais”.

O formato do MovieLens consiste em quatro valores, sendo eles: id do usuário, id do item, preferência e timestamp, conforme mostrado na Listagem 11.

**Listagem 11.** Formato do arquivo `ratings.dat` do MovieLens.

```
1::150::1.0::1245326
1::1233::1.0::124532
1::533::2.0::1244326
1::543::5.0::1345734
```

Com o arquivo original do movielens formatado para o padrão da API do Mahout (um novo arquivo com o formato do Mahout foi criado pelo *mapper* da Listagem 12), já é possível gerar as recomendações com o Mahout. Isso é feito com a execução do *mapreduce* disponível no *jar mahout-core-0.7-job.jar*, usando como entrada o arquivo gerado pelo mapper da Listagem 12. Essa execução é realizada com o comando mostrado na Listagem 14.

É possível gerar tanto recomendações para cada usuário a partir da classe `org.apache.mahout`.

## Problemas com versões do Hadoop

A versão 0.7 do Mahout utiliza a versão 0.20 do Hadoop. Tentar executar o Mahout com versões mais recentes do Hadoop, como o CDH4, apresentará alguns erros de incompatibilidade do Hadoop. Para solucionar esse tipo de problema, foi necessário baixar o código-fonte do Mahout, alterar o `pom.xml` e recompilá-lo com a mesma versão do Hadoop a ser utilizado, gerando um novo `mahout-core-0.7-job.jar`.

**cf.taste.hadoop.item.RecommenderJob**, usando o comando da Listagem 14, quanto obter apenas as similaridades entre os itens, com o processo implementado por **org.apache.mahout.cf.taste.hadoop.similarity.item.ItemSimilarityJob**, mostrado na Listagem 15. Em ambos os jobs mencionados, foi utilizada a classe do Mahout que implementa o algoritmo de similaridade por matriz de co-ocorrência, representada pela constante **SIMILARITY\_COOCURRENCE** para o cálculo de similaridade entre os itens. Essa é a implementação que utiliza a matriz de co-ocorrência explicada anteriormente.

**Listagem 12.** Mapper para formatação do arquivo do MovieLens para o Mahout.

```
public class MovieLensFormatMapper extends
Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context
context) throws InterruptedException, IOException{
        String[] values = value.toString().split("::");
        context.write(new Text(values[0] + ";" + values[1] + ";"
+ values[2]), new Text());
    }
}
```

**Listagem 13.** Comando para executar mapper de conversão do arquivo do MovieLens para o formato de entrada do Mahout.

```
hadoop jar movielensformatter.jar org.mundoj.mahout.
hadoop.MovieLensFormatDriver \
/mundoj/ratings.dat /mundoj/output-formatted/output-03
```

**Listagem 14.** Comando para iniciar o job para geração de recomendação com Mahout.

```
hadoop jar mahout-core-0.7-job.jar \
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob \
-Dmapred.input.dir=/mundoj/output-formatted/output-03/
part-r-00000 \
-Dmapred.output.dir=/mundoj/output-recomm/users-
recommendations-01 \
--similarityClassname SIMILARITY_COOCURRENCE
```

**Listagem 15.** Comando para iniciar o job para geração de similaridade de itens com Mahout.

```
hadoop jar mahout-core-0.7-job.jar \
org.apache.mahout.cf.taste.hadoop.similarity.item.
ItemSimilarityJob \
-Dmapred.input.dir=/mundoj/output-formatted/output-03/
part-r-00000 \
-Dmapred.output.dir=/mundoj/output-recomm/
similarities-01 \
--similarityClassname SIMILARITY_COOCURRENCE
```

O comando mostrado na Listagem 14 irá, na realidade, executar vários jobs MapReduce, conforme pode ser visto no console web do JobTracker, mostrado na figura 2. Os primeiros três MapReduces da tabela, cujos nomes começam com o prefixo **PreparePreferenceMatrixJob**, tratam de transformar o arquivo de entrada em uma matriz de preferências, agrupando os itens relacionados em vetores que representam a co-ocorrência de cada item, com todos os outros itens, ao longo das avaliações dos usuários. Os três processos seguintes, começando com **RowSimilarityJob**, contabilizam os vetores resultantes da etapa anterior, de forma a obter o valor de co-ocorrência de cada par de itens. O resultado dessa fase seria a matriz de co-ocorrência. Os quatro últimos, contendo o prefixo **RecommenderJob**, executarão a

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed
job_201208301332_0001	NORMAL	tester	PreparePreferenceMatrixJob-ItemDIndexMapper-Reducer	100.00%	2	2	100.00%	1	1
job_201208301332_0002	NORMAL	tester	PreparePreferenceMatrixJob-ToltemPrefsMapper-Reducer	100.00%	2	2	100.00%	1	1
job_201208301332_0003	NORMAL	tester	PreparePreferenceMatrixJob-ToltemVectorsMapper-Reducer	100.00%	1	1	100.00%	1	1
job_201208301332_0004	NORMAL	tester	RowSimilarityJob-VectorNormMapper-Reducer	100.00%	1	1	100.00%	1	1
job_201208301332_0005	NORMAL	tester	RowSimilarityJob-CooccurrencesMapper-Reducer	100.00%	2	2	100.00%	1	1
job_201208301332_0006	NORMAL	tester	RowSimilarityJob-UnsymmetrifyMapper-Reducer	100.00%	5	5	100.00%	1	1
job_201208301332_0007	NORMAL	tester	RecommenderJob-SimilarityMatrixRowWrapperMapper-Reducer	100.00%	1	1	100.00%	1	1
job_201208301332_0008	NORMAL	tester	RecommenderJob-User/VectorSplitterMapper-Reducer	100.00%	1	1	100.00%	1	1
job_201208301332_0009	NORMAL	tester	RecommenderJob-MapMapper-Reducer	100.00%	4	4	100.00%	1	1
job_201208301332_0010	NORMAL	tester	RecommenderJob-PartialMultiplyMapper-Reducer	100.00%	2	2	100.00%	1	1

Figura 2. Lista dos MapReduces que o RecommenderJob executa.



File: [/mundoj/output-recomm/users-recommendations-01/part-r-00000](#)

Goto: [/mundoj/output-recomm/u](#)

[Go back to dir listing](#)

[Advanced view/download options](#)

[View Next chunk](#)

```
1 [648:5.0,553:5.0,552:5.0,551:5.0,1101:5.0,543:5.0,541:5.0,1097:5.0,4993:5.0,2762:5.0]
2 [1213:4.679751,1221:4.676387,750:4.535784,1208:4.527122,7153:4.518246,1961:4.5131717,3996:4.512747,2396:4.50915,420:4.4460497,160:4.4056487]
3 [6333:5.0,3114:4.9515476,5816:4.949171,5502:4.944991,2324:4.93199,8636:4.9318614,8360:4.927317,33794:4.927222,7147:4.8300676,1917:4.801166]
4 [593:5.0,553:5.0,551:5.0,1101:5.0,541:5.0,1097:5.0,539:5.0,4993:5.0,2762:5.0,1089:5.0]
5 [913:5.0,555:5.0,1307:5.0,551:5.0,111:5.0,6:5.0,1101:5.0,1097:5.0,539:5.0,4993:5.0]
6 [919:5.0,1617:5.0,10:5.0,1101:5.0,3897:5.0,541:5.0,1097:5.0,539:5.0,4993:5.0,4993:5.0]
7 [969:5.0,1617:5.0,2012:5.0,2959:5.0,1101:5.0,3897:5.0,541:5.0,1097:5.0,4993:5.0,4993:5.0]
8 [750:5.0,1544:5.0,2000:5.0,3623:5.0,1101:5.0,3948:5.0,541:5.0,1097:5.0,1653:5.0,4993:5.0]
9 [750:5.0,551:5.0,6:5.0,3897:5.0,541:5.0,1097:5.0,4993:5.0,4993:5.0,2762:5.0,1089:5.0]
10 [750:5.0,1288:5.0,2268:5.0,3147:5.0,11:5.0,6:5.0,1101:5.0,1097:5.0,539:5.0,4993:5.0]
11 [1035:5.0,2000:5.0,2797:5.0,11:5.0,1101:5.0,541:5.0,4993:5.0,2762:5.0,1089:5.0,2791:5.0]
12 [329:4.569371,6:4.551476,185:4.541648,786:4.538912,208:4.538913,2:4.536939,1393:4.5335727,95:4.5305347,292:4.5300035,434:4.5291333]
13 [919:5.0,5010:5.0,1617:5.0,2000:5.0,2640:5.0,6:5.0,1101:5.0,541:5.0,1097:5.0,4993:5.0]
14 [3408:5.0,44191:5.0,3977:5.0,3481:5.0,8665:5.0,4246:5.0,3897:4.9188676,3751:4.8893933,33493:4.886429,1080:4.8691072]
16 [590:5.0,1617:5.0,1732:5.0,1101:5.0,541:5.0,1097:5.0,1653:5.0,4993:5.0,2762:5.0,1089:5.0]
17 [898:5.0,1304:5.0,2000:5.0,6:5.0,1101:5.0,541:5.0,1097:5.0,539:5.0,4993:5.0,4993:5.0]
18 [7361:4.739416,7438:4.726931,474:4.673625,1394:4.6660924,329:4.6617537,434:4.6583014,185:4.6574993,161:4.6517715,253:4.6507573,300:4.650599]
19 [555:5.0,553:5.0,552:5.0,551:5.0,1250:5.0,2918:5.0,11:5.0,6:5.0,1101:5.0,4306:5.0]
22 [1221:4.630735,1193:4.5213385,1387:4.4915276,2997:4.455606,1213:4.4438734,4226:4.385389,111:4.365153,5952:4.3575845,1961:4.354601,1089:4.328192]
23 [750:5.0,555:5.0,1247:5.0,551:5.0,11:5.0,541:5.0,1097:5.0,539:5.0,4993:5.0,2762:5.0]
24 [750:5.0,1265:5.0,1721:5.0,1101:5.0,541:5.0,1097:5.0,539:5.0,4993:5.0,2762:5.0,1089:5.0]
26 [553:5.0,509:5.0,551:5.0,786:5.0,2:5.0,1101:5.0,2762:5.0,1097:5.0,1704:5.0,541:5.0]
27 [1391:4.9108152,1127:4.8093467,1544:4.771478,2916:4.670823,1917:4.6334324,1573:4.6147323,4306:4.606866,3793:4.5332313,2:4.4387383,329:4.43714]
28 [1080:4.0,1288:4.0,2470:4.0,1101:4.0,541:4.0,539:4.0,4993:4.0,2762:4.0,1089:4.0,527:4.0]
29 [1653:5.0,6539:5.0,5378:5.0,4896:5.0,3408:4.9068174,2355:4.8868,586:4.8715773,3623:4.8153634,3147:4.7555504,364:4.6769376]
```

[Download this file](#)

[Tail this file](#)

Chunk size to view (in bytes, up to file's DFS block size):

Total number of blocks: 1

-5545507186198429236: [127.0.0.1:50010](#) [View Block Info](#)

Figura 3. Exemplo de saída do job de recomendação.

multiplicação da matriz pelo vetor de preferências de cada usuário, e já ordenará os itens mais relevantes.

Como saída do último MapReduce, é gerado um arquivo que demonstra as recomendações para cada usuário, exemplificado na figura 3. Como pode ser visto, todo esse processamento requer uma abordagem assíncrona. Assim, para tornar os dados disponíveis em um sistema on-line, como uma aplicação web, é necessário importar o conteúdo gerado e armazená-lo em alguma estrutura de cache (em memória, tabela temporária em base de dados etc.). O formato de saída é: id do usuário [id do item:peso de importância do item para o usuário]. Vale lembrar que as recomendações geradas utilizando o Hadoop são todas recomendações off-line, visto que o fluxo completo de MapReduces pode durar alguns minutos para executar completamente. Também nem sempre uma recomendação local (não distribuída) é on-line. A diferença entre as duas recomendações é que as recomendações on-line são processadas em tempo real, ou seja, se você fez uma alteração no seu perfil ele será considerado na próxima recomendação solicitada por você, enquanto que na recomendação off-line, como a mesma é pré-processada, essa alteração pode não ser considerada quando solicitada uma nova recomendação.

No arquivo de recomendações resultante mostra-

do como exemplo na figura 3, a primeira linha apresenta os *Ids* dos dez filmes mais recomendados para o usuário de Id 1: [648, 553, 552, 551, 1101, 543, 541, 1097, 4993, 2762]. No arquivo também está disponível o peso da recomendação de cada item, o que no caso do usuário 1, foi 5 (peso máximo) para todos os 10 itens. Analisando, por curiosidade, o arquivo movies.dat do conjunto de dados baixado do GroupLens (arquivo zip citado anteriormente), pode-se descobrir quais são esses filmes recomendados. A Listagem 16 apresenta uma amostra desse arquivo, com cada um dos filmes recomendados para o usuário 1. Como no caso do usuário 1 todos os filmes continham o mesmo peso, a Listagem 16 exhibe os detalhes dos filmes ordenados por Id.

**Listagem 16.** Amostra do arquivo movies.dat, com detalhes de cada filme recomendado para usuário 1.

```
...
541::Blade Runner (1982)::Adventure|Drama|Film-Noir|Sci-Fi|Thriller
...
543::So I Married an Axe Murderer (1993)::Comedy|Romance|Thriller
...
551::Nightmare Before Christmas, The (1993)::Animation|Children|Fantasy|Musical
```

```

552::ThreeMusketeers,The(1993)::Action|Adventure|Comedy
553::Tombstone (1993)::Action|Drama|Western
...
648::Mission: Impossible (1996)::Action|Adventure|Mystery|
Thriller
...
1097::E.T. the Extra-Terrestrial (1982)::Children|Drama|Sci-Fi
...
1101::Top Gun (1986)::Action|Romance
...
2762::Sixth Sense, The (1999)::Drama|Mystery|Thriller
...
4993::Lord of the Rings: The Fellowship of the Ring, The
(2001)::Action|Adventure|Fantasy

```

## Considerações Finais

Neste artigo, foram abordados com exemplos práticos, uma aplicação de recomendação de filmes usando parte da base de dados do MovieLens, explorando o Hadoop como solução de BigData e o Mahout para processamento e geração de recomendações. Conseguimos mostrar a diferença no uso do Mahout em um sistema distribuído usando o Hadoop de forma assíncrona de um sistema não distribuído, onde o Mahout oferece uma gama maior na variedade da implementação de seus algoritmos. Foram abordados também conceitos básicos de recomendação e filtragem colaborativa com o intuito de contextualizar a ferramenta no problema proposto. Não foram abordados detalhes dos algoritmos de recomendações que o Mahout implementa visto que tornaria o artigo mais complexo e mais de cunho acadêmico do que técnico, por isso foi focado apenas o uso da ferramenta e o que ela provê.

Foi exibido, também, o funcionamento em detalhes dos *jobs* MapReduce que o Hadoop executa para a geração das recomendações, demonstrando o flow com as principais classes e o que é processado nas etapas mais relevantes do processo de geração das matrizes, vetores e, por fim, na recomendação de novos filmes para os usuários propostos.

Como possibilidade de mais testes com o exemplo proposto, o leitor pode fazer algumas alterações no arquivo de entrada do Mahout usado neste artigo, adicionando dados para um novo usuário, e tentar gerar a recomendação para o usuário inserido, ou alterar a preferência de alguns dos usuários e checar se houve alguma mudança. Às vezes, quando a quantidade de dados é muito extensa, uma só alteração não faz efeito, sendo necessário alterar dados de vários usuários.

Como continuação desse trabalho, existem vários assuntos que podem ser explorados como extensão ao tema de recomendação com grande volume de da-

dos. Uma delas é explorar a integração entre a camada de apresentação de uma aplicação web (*frontend*), ou seja, o módulo do sistema com que os usuários interagem em tempo real e produz os dados, os quais precisam ser exportados para o sub-sistema de geração de recomendação e processamento de BigData (no exemplo deste artigo, Mahout com Hadoop).

## /para saber mais

> Nas edições 52, 53 e 54 da *MundoJ* foram apresentados artigos detalhando o funcionamento do Hadoop, o qual pode ser definido como um sistema de persistência e processamento distribuído, útil para tratar grande volume de dados.

## /referências

> *Mahout in Action* - Sean Owen, Robin Anil, Ted Dunning e Ellen Friedman - Manning Publications, 2011.

> *Algorithms of the Intelligent Web* - Haralambos Marmanis e Dimitry Babenko - Manning Publications, 2009.

> <https://cwiki.apache.org/confluence/display/MAHOUT/Mahout+Wiki> - Página inicial da Wiki oficial do Mahout.

> <https://cwiki.apache.org/confluence/display/MAHOUT/Itembased+Collaborative+Filtering> - Página da Wiki do Mahout que descreve as implementações MapReduce de filtragem colaborativa do Mahout.

> <http://www.grouplens.org/node/12> - Site do projeto GroupLens, contendo vários conjuntos de dados para experimentos sobre recomendação, inclusive o do MovieLens, usado neste artigo.

> <http://www.movielens.org> - Site do MovieLens, serviço de recomendação de filmes gratuito.

> <https://www.ibm.com/developerworks/java/library/j-mahout/index.html> - Introducing Apache Mahout - Grant Ingersoll - Intro to Apache Mahout focused on clustering, classification and collaborative filtering.

> <http://www.slideshare.net/sscdotopen/mahoutcf-Apresentação sobre filtragem colaborativa distribuída usando Mahout>.