UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

GERÊNCIA DE DADOS NA WEB
TRABALHO PRÁTICO 1

Luiz Carlos A. M. Cavalcanti

Manaus, dezembro de 2013

Introdução

O programa descrito a seguir visa minerar conjuntos de itens frequentes em um ambiente distribuído baseado no modelo MapReduce.

O problema proposto neste trabalho visa encontrar relações de recorrência entre atributos de um conjunto de dados classificados. Ou seja, pretendemos descobrir que valores para os atributos de um conjunto de amostras é frequente o suficiente a ponto de ficar acima de um limiar de frequência informado pelo usuário do programa.

As seções a seguir descrevem, nesta ordem, os dados utilizados na modelagem e experimentação, o algoritmo de MapReduce escolhido como solução, as estruturas de dados e seus usos no programa, a lista de códigos-fonte e suas funções.

Dados

Para os experimentos, foram utilizados os dados da coleção *Mushroom* (http://archive.ics.uci.edu/ml/datasets/Mushroom), que consiste em 8124 amostras de cogumelos, classificados entre "venenoso" e “comestível”. Além da classificação, cada amostra também é composta por outros 22 atributos que vão desde cor e formato até o habitat onde os espécimes se encontram.

A coleção, embora relativamente grande, possui dados bastante repetitivos, de forma que é preciso agrupar muitos atributos a fim de encontrar grupos distintos entre os dados. Alguns atributos chegam a ter o mesmo valor em todas as amostras do conjunto de dados.

Para que pudesse facilitar o *parsing* e processamento dos dados durante a execução do MapReduce, foi criado um pequeno programa utilitário responsável por pré-processar o arquivo original de dados e gerar um novo arquivo já adequado ao uso pelo *framework* Hadoop. Os detalhes de implementação desse utilitário serão descritos em seção pertinente adiante neste documento. Um exemplo de registro do conjunto de dados em seu formato original é o seguinte:

p,x,s,n,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,u

Após a conversão, o mesmo registro é apresentado no arquivo a ser processado pelo programa principal da seguinte forma:

00:p,01:x,02:s,03:n,04:t,05:p,06:f,07:c,08:n,09:k,10:e,11:e,12:s,13:s,14:w,15:w,16:p,17:w,18:o,19:p,20:k,21:s,22:u

As marcações no formato *número-da-coluna:valor* são necessárias pois atributos diferentes das amostras podem possuir o mesmo valor nominal, como por exemplo *W* (white), que pode aparecer em diversos atributos de uma mesma amostra. O importante a ser entendido nesse caso, é que não se trata de um mesmo item repetido, e sim de itens diferentes. O valor *W* para o atributo *cap-color* não é o mesmo item que o valor W para o atributo *gill-color*, mesmo em amostras diferentes do conjunto de dados.

Algoritmo

O algoritmo Apriori em ambiente distribuído baseado no modelo MapReduce foi implementado para este trabalho. A implementação se baseou em dois *Mappers* e um *Reducer.*

O primeiro *Mapper* (AprioriMapPasso1) conta a ocorrência de atributos no primeiro passo do algoritmo, quando o número de itens na transação/registro é 1. Este mapper é bastante parecido com a implementação para o problema de contagem de palavras. Ao fim, o Reduce (AprioriReduce) é executado, eliminando todos os itens cuja soma total de ocorrências é inferior ao suporte mínimo (limiar) estipulado pelo usuário. O suporte mínimo é informado em forma de percentual, e serve para determinar quais registros permanecem como candidatos para a próximo passo do algoritmo. Os candidatos cujo percentual de incidência no total de registros fica abaixo do estipulado pelo suporte mínimo são eliminados e não serão considerados em execuções posteriores.

Os passos ou execuções subsequentes do algoritmo utilizam um segundo *Mapper* (AprioriMapPassoK), que difere do primeiro por utilizar como entrada, além do conjunto de dados original, o resultado do Reduce do passo anterior. A cada passo, a dimensão do conjunto de itens aumenta em um, de forma que o passo *k* possui como candidatos uma coleção de registros com *k itens*.

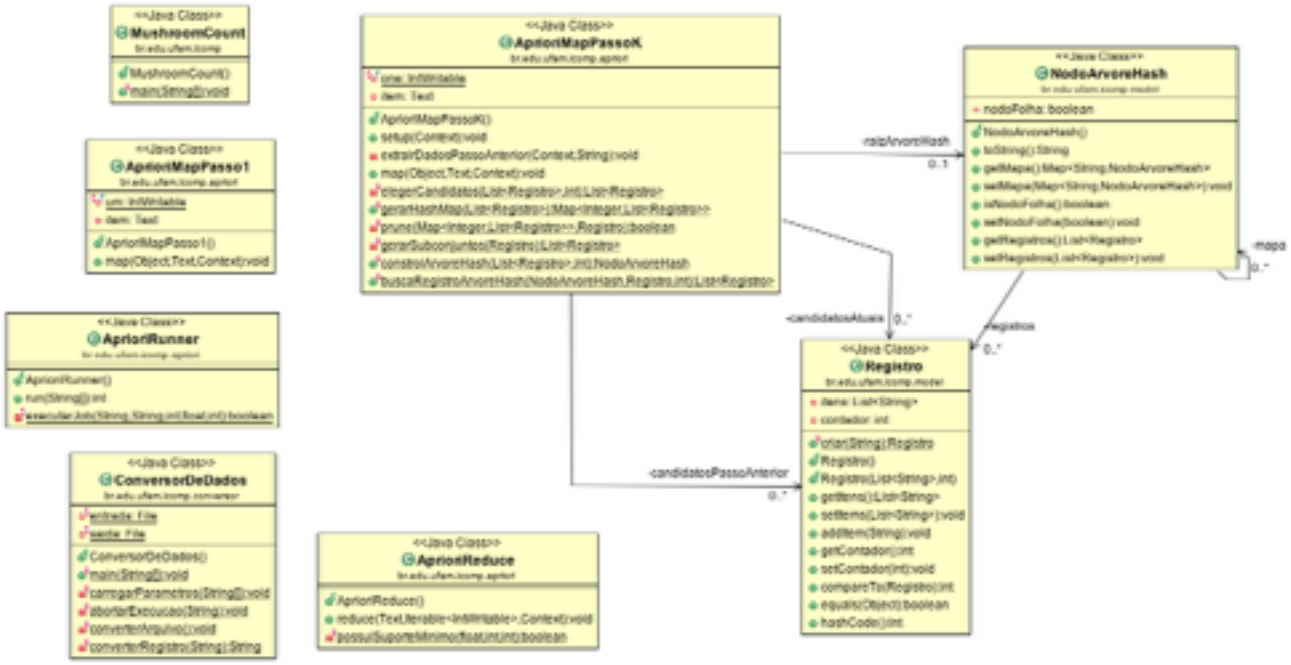
Os dados entre uma execução e outra do Apriori MapReduce são compartilhados através do recurso de cache distribuído (DistributedCache) fornecido pelo framework hadoop, que garante que antes do próximo passo, cada nodo que executará o Map obtenha uma cópia dos resultados do passo anterior.

A eleição de candidatos, que ocorre durante o Map, é feita primeiramente gerando todos os subconjuntos de itens possíveis a partir do resultado do passo anterior. Depois uma leitura do conjunto de dados completa é feita, realizando a contagem de ocorrências desses subconjuntos de itens. Por fim, o Reduce elimina todos os candidatos que se encontram abaixo do suporte mínimo informado pelo usuário.

O número de passos é determinado pelo usuário no começo da execução do programa. Mesmo após um passo não encontrar mais conjuntos de itens comuns que se encontrem acima do suporte mínimo, o algoritmo continua sua execução. Tendo como entrada uma coleção de

candidatos anteriores de tamanho zero, esses passos também produzirão, obrigatoriamente, uma saída de tamanho 0.

A modelagem de classes do programa é apresentada no diagrama abaixo:



Mais detalhes sobre cada função e seu funcionamento interno podem ser encontrados em forma de comentários no código-fonte.

Estruturas de dados

Para representar a lista de candidatos em memória, uma Árvore de Hash (variação da Árvore de Merkle) é criada a cada passo k quando k > 1 (AprioriMapPassoK) em memória. Esta abordagem possui algumas vantagens de desempenho como, por exemplo, a possibilidade de fazer junção e *prune* na coleção de candidatos sem ter que passar por toda a coleção de candidatos do passo anterior. A estrutura de um nodo da árvore é representada no programa pela classe NodoArvoreHash:

```
public class NodoArvoreHash {
    private Map<String, NodoArvoreHash> mapa;
    private boolean nodoFolha;
    private List<Registro> registros;
}
```

As demais estruturas de dados utilizadas no programa são de uso comum da plataforma Java, tais como HashMap, ArrayList e demais estruturas do pacote Collections. Estas estruturas foram utilizadas levando em consideração sua facilidade de uso (ordenação através de Comparables, por exemplo), bom desempenho e serialização disponível.

Experimentos

Utilizando toda a coleção de dados fornecida, alguns testes foram realizados, variando dois parâmetros de execução do algoritmo: suporte mínimo e número de passos.

Os dados abaixo exibe um levantamento empírico de desempenho e eficácia do algoritmo, e considera qual a quantidade de registros encontrados no último passo onde o tamanho da saída é maior que zero.

Número de passos	Suporte mínimo (limiar)	Tempo de execução (ms)	Conjunto de itens encontrados
4	70%	64.499	5
4	75%	63.542	5
4	80%	62.539	2
4	85%	62.578	1

O alto valor percentual utilizado no atributo de suporte se faz necessário por conta da grande repetição dos atributos do conjunto de dados explorado. Como é pouco interessante para o problema proposto que hajam muitos registros a serem analisados, preferimos dar ênfase a configurações que gerem rapidamente poucos resultados. Com isso em mãos, é possível inferir a relação entre seus atributos de forma mais objetiva e factível, em comparação a uma situação onde dezenas ou centenas de combinações fossem encontradas, tornando inviável tal análise em tempo razoável.

Conclusão

Neste trabalho foi possível implementar um algoritmo Apriori em ambiente distribuído sob o modelo MapReduce, e entender melhor como técnicas de mineração em grandes massas de dados podem ser utilizadas para resolver problemas além daqueles vastamente descritos na literatura. Foi possível, mesmo que sem confirmação através de outros métodos (como cálculo de ganho de informação e técnicas de aprendizagem de máquina), descobrir a relação entre atributos de um conjunto de dados.

Para trabalhos futuros no tema, podem ser abordadas melhorias no mecanismos de controle, especialmente no que tange a parametrização manual do número de passos, substituindo-o por um parâmetro de número máximo de passos, visto que em muitos casos explorados nesse trabalho, o algoritmo continuou sua execução até o último passo, mesmo sem candidatos novos em relação ao passo anterior.

Há também espaço para melhoria de desempenho nos algoritmos de seleção de candidatos e criação de subconjuntos, uma vez que as técnicas utilizadas neste trabalho foram rudimentares, privilegiando a simplicidade em detrimento do desempenho.