Compressão de Arquivos Orientados a Colunas com PPM

Vinicius F. Garcia[1], **Sergio L. S. Mergen**[1]

[1]Universidade Federal de Santa Maria
Santa Maria – RS – Brasil

vfulber@inf.ufsm.br, mergen@inf.ufsm.br

Abstract. *Column oriented databases belong to a kind of NoSQL database in which the values of the same column are stored contiguously in secondary memory. This physical organization favors compression, mainly because the proximity of data of the same nature decreases the information entropy. With respect to high cardinality columns that store text, several compression methods can be used. One of them, called PPM, is usually good in obtaining high compression rates, but the execution time is poor for conventional files. The purpose of this paper is to analyze whether this compression method is able to explore the nature of column oriented data to obtain more expressive results in comparison with is main competitors.*

Resumo. *Bancos de dados orientados a colunas pertencem a um tipo de banco NoSQL em que os valores de uma mesma coluna são armazenados contiguamente em memória secundária. Essa organização física favorece a compressão, uma vez que a aproximação dos dados de mesma natureza diminui a entropia da informação. Considerando especificamente colunas de alta cardinalidade que armazenam texto, diversos tipos de compressores podem ser usados. Um deles, chamado PPM, costuma obter boas taxas de compressão, mas possui um tempo de processamento considerado alto para arquivos convencionais. O objetivo desse artigo é verificar se esse método consegue explorar a natureza dos dados orientados a coluna de forma a obter resultados mais expressivos em relação aos seus concorrentes.*

1. Introdução

Os bancos de dados NoSQL tem recebido muita atenção recentemente. Ao contrário dos bancos de dados relacionais, essa nova vertente utiliza diferentes formas de organização de arquivos. Utilizando arquiteturas baseadas na computação em nuvem, esse tipo de solução oferece um bom escalonamento para determinados tipos de aplicações que usam padrões de acesso aos dados bem específicos.

Um dos tipos de banco NoSQL que se popularizou é conhecido como orientados a colunas (Han et al., 2011). Diferentemente dos SGBDs convencionais que armazenam registros de tabelas consecutivamente em arquivos, os sistemas orientados a colunas armazenam todos os valores de uma mesma coluna consecutivamente, possivelmente em arquivos separados, conforme ilustrado na Figura 1.

Essa forma de organização é útil em alguns cenários específicos, como por exemplo, para acelerar a execução de consultas analíticas que acessam poucas colunas, uma vez que é possível delimitar os arquivos que o processador de consultas deve varrer. Além

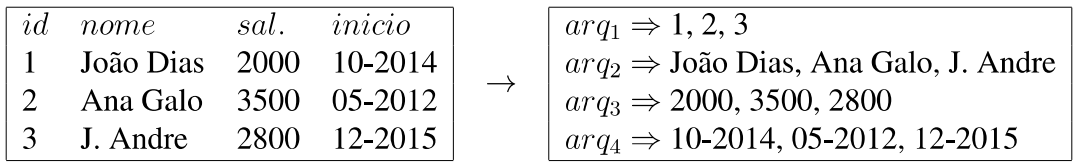


Figura 1. Orientação à registros e à colunas

disso, é possível representar de maneira mais eficiente as colunas que aceitam valores nulos, sem precisar recorrer a mapas de bits indicando campos nulos.

Outro ponto que merece destaque com relação aos bancos orientados a colunas é a sua capacidade de compressão de dados. Nota-se que com essa organização os arquivos passam a ser formados por valores que pertencem ao mesmo domínio e tipo de dados. Isso reduz a entropia da informação, e os algoritmos de compressão podem se beneficiar disso para obter uma taxa de compressão superior.

Caso uma coluna possua baixa cardinalidade (poucos valores distintos), um método de compressão bastante eficaz é a substituição do valor por um código que referencia uma entrada em um dicionário. Já para a compressão de colunas de texto de alta cardinalidade, os métodos que exploram padrões dentro do texto, como LZ (Ziv e Lempel, 1978) e BWT (Burrows e Wheeler, 1994), são mais adequados.

Outro método que encontra grande utilidade na compressão de dados textuais de alta cardinalidade é conhecido como PPM (*Prediction by Partial Matching*) (Moffat, 1990). As taxas de compressão obtidas pelas diversas variações do PPM concorrem com os resultados obtidos pelos demais compressores. O que impede o seu uso mais disseminado é o elevado tempo de processamento.

As estatísticas de desempenho dos compressores PPM são oriundas de testes realizados sobre *benchmarks* conhecidos na área de compressão de dados, como o corpus CALGARY (Council, 2008), formado por arquivos de formatos variados, como textos, imagens e códigos em linguagens de programação. No entanto, bancos de dados orientados a colunas possuem características bem distintas no que diz respeito aos padrões que podem ser encontrados.

Assim sendo, o objetivo desse artigo é investigar o comportamento do PPM na compressão de dados textuais de alta cardinalidade e analisar se seu uso é viável para arquivos orientados a colunas. A avaliação envolve a análise do tempo de execução e a taxa de compressão do PPM, comparando os resultados àqueles que são obtidos pelo LZ e BWT.

O artigo está estruturado da seguinte forma: a seção 2 apresenta resumidamente algumas das principais estratégias de compressão de dados textuais propostas na literatura. Na seção 3 o método de compressão PPM é apresentado, afim de explicar porque esse método parece especialmente adequado para dados orientados a colunas. A seção 4 apresenta experimentos que foram feitos comparando o desempenho de diversos algoritmos de compressão em cenários compostos por arquivos orientados a colunas e arquivos convencionais pertencentes ao corpus *Calgary*. A seção 5 traz as considerações finais.

2. Algoritmos de Compressão de dados

A compressão de dados textuais sem perda recebeu muita atenção da comunidade científica em décadas passadas. Uma das primeiras ideias exploradas foram as chamadas técnicas de codificação estatística, como a codificação de Huffman (Huffman et al., 1952) e a codificação aritmética (Witten et al., 1987). Nos dois casos a compressão é obtida representando os caracteres (ou símbolos) mais frequentes do arquivo usando menos bits.

O código de Huffman emprega uma árvore binária que mapeia símbolos como cadeias de bits. Por sua vez a codificação aritmética emprega uma tabela que guarda a frequências de ocorrências dos símbolos já processados. Sabendo essas frequência, pode-se calcular a probabilidade de ocorrência de qualquer símbolo. Essa probabilidade é então codificada como um valor binário de ponto fixo. Os codificadores de Huffman e aritméticos podem ser estáticos, quando usam árvores/tabelas de frequência predeterminadas, ou dinâmicos, quando as árvores/tabelas são construídas à medida que a compressão ocorre.

Mais tarde surgiram técnicas de compressão que passaram a levar em consideração não apenas a frequência dos símbolos, mas o fato de que muitos símbolos costumam aparecer juntos. As técnicas que exploram essa característica são conhecidas como baseadas em dicionário. Os algoritmos dessa categoria mais usados hoje em dia derivam de uma ideia proposta por Ziv e Lempel (1978), e são referenciados pelas iniciais de seus autores (LZ). De modo geral, a sequência de símbolos já processada do arquivo de entrada forma o dicionário. Uma sequência de símbolos a codificar é representada através de um índice de deslocamento e uma largura. O índice indica um ponto no arquivo de entrada onde essa sequência já foi encontrada. A largura determina quantos símbolos a partir desse índice são equivalentes aos símbolos que se deseja comprimir. Essas duas informações são representadas através de um codificador estatístico, sendo o código de Hufmann mais comumente utilizado. Essa ideia levou à especificação de um padrão para codificação chamado DEFLATE (Deutsch, 1996) e serviu de base para a criação dos compressores de dados mais usados comercialmente, como gzip e lzip.

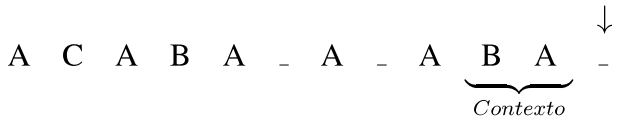
Outra técnica que se mostrou particularmente interessante para a compressão de texto foi proposta por Burrows e Wheeler (1994). Seu nome, BWT, é um acrônimo que remete aos nomes dos autores (*Burrows Wheeler Transform*). O passo inicial do algoritmo gera todas as permutações que se obtém ao rotacionar o texto a comprimir um símbolo de cada vez. Essas permutações são armazenadas em uma matriz onde as linhas são ordenadas. No próximo passo todas as colunas com exceção da última são descartadas. É possível reconstruir o texto original usando apenas essa última coluna e um índice que localiza a linha da matriz onde o texto original estaria armazenado. Esse método parte da constatação de que alguns símbolos são normalmente precedidos por determinados símbolos. Caso isso ocorra, a última coluna da matriz será composta por muitos símbolos repetidos. Isso abre espaço para a aplicação de uma técnica chamada MTF (*Move to Front*) que visa transformar essa saída em outra composta por valores de 0 a 255 com a predominância de valores baixos. O último passo (que é onde a compressão realmente ocorre) envolve codificar essa saída composta por valores numéricos usando algum codificar estatístico como *Huffman* ou codificação aritmética.

Também existem trabalhos voltados à bancos de dados orientados a colunas que exploram colunas que possuem determinadas características (Abadi et al., 2009). Por

exemplo, quando é comum que os valores sejam compostos por muitos caracteres em branco consecutivos, pode-se empregar técnicas de supressão de nulos, cujo objetivo é remover um símbolo de elevada ocorrência (como o espaço em branco, por exemplo), deixando em seu lugar a sua localização e quantidade (Westmann et al., 2000). Caso mais símbolos costumem aparecer de forma consecutiva, uma técnica simples e que encontra empregabilidade em diversas aplicações é a RLE (*Run Length Encoding*), onde símbolos consecutivos repetidos são substituídos por um par composto pelo símbolo e pelo número de repetições. Essa técnica pode ser útil em sistemas orientados a colunas que guardam os valores ordenados (Abadi et al., 2006). Já o uso de dicionários e vetores de bits (Wu et al., 2002) são indicados para casos em que a quantidade de valores distintos é baixa (baixa cardinalidade). De modo geral, esses trabalhos tem objetivos ortogonais aos propostos neste artigo, cujo foco é em dados textuais de alta cardinalidade e que não necessariamente estejam ordenados.

3. PPM

O método de compressão PPM codifica um símbolo de cada vez. Para gerar um código é levado em consideração o contexto, que são os símbolos que precedem o símbolo a ser codificado. Para compreender o funcionamento do PPM, considere o texto a comprimir indicado abaixo. A seta indica o próximo símbolo a ser codificado, e as chaves indicam o contexto a ser analisado.



Dado o símbolo a ser codificado ('_'), a codificação é determinada pela probabilidade de ocorrência desse símbolo dado o contexto que o precede. Para realizar esse cálculo é necessário analisar quais símbolos já ocorreram no passado quando esse contexto foi encontrado, e quais são as frequências de ocorrência desses símbolos. Quanto maior a frequência, maior é a probabilidade. A probabilidade pode ser computada usando codificação aritmética, de modo que símbolos mais prováveis gerem menos bits durante a codificação.

Pela Tabela 1 é possível observar quais símbolos ocorreram até então (e suas frequências) para cada ordem do contexto atual. Por exemplo, para o contexto de maior ordem ('BA') o histórico mostra que apenas um símbolo ocorreu ('_'), tendo ocorrido uma vez. Para cada contexto um símbolo especial é reservado, chamado de escape (ESC). A frequência desse símbolo depende da implementação do PPM. A implementação clássica considera que a frequência é equivalente ao número de símbolos distintos que já ocorreram naquele contexto (Moffat, 1990). O propósito desse símbolo especial será descrito mais adiante.

No caso em questão, o símbolo '_' tem uma probabilidade de ocorrência equivalente a 50% após o contexto 'BA'. Esse percentual é traduzido em um código binário de ponto fixo através de codificação aritmética. Em seguida a tabela de frequência dos contextos é atualizada com o símbolo recém processado e o codificador avança para processar o próximo símbolo.

Tabela 1. Contextos de ordens de zero a dois e seus respectivos símbolos/frequências

Ordem	Contexto	Símbolo (Frequência)
2	B A	ESC(1), _(1)
1	A	ESC(3), B(2), C(1), _(2)
0		ESC(4), A(5), B(2), C(1), _(2)

Caso o símbolo a codificar fosse 'C' em vez de '_' (supondo o texto 'ACABA_A_ABAC'), observa-se que em nenhum momento no passado esse símbolo foi encontrado depois do contexto 'BA'. Nesse caso deve ser codificado o símbolo de escape, com probabilidade de 50%. Esse símbolo sinaliza que o contexto deve ser reduzido (de 'BA' para 'A') e a busca feita novamente. Dessa vez, três símbolo ocorreram após 'A'. Um deles é aquele que se deseja codificar, com probabilidade de 12,5% (uma ocorrência dentre as oito existentes). A existência de probabilidades iguais (ex. a probabilidade de aparecer 'B' ou '_' depois de 'A') é resolvida pela codificação aritmética através da divisão da escala de probabilidades em intervalos.

O pseudo-código do Algoritmo 1 mostra como o contexto diminui de tamanho a medida que a busca avança. No pior dos casos o contexto é reduzido à ordem 0 (zero). Nesse caso leva-se em consideração a frequência total dos símbolos, independente de onde eles apareceram. Todos os símbolo possíveis são contemplados nessa lista, então a busca sempre será bem sucedida nesse nível. A descompressão segue o caminho inverso. Símbolos decodificados alimentam o contexto e servem de indício para a decodificação do próximo símbolo.

Algoritmo 1: CODIFICAÇÃO USANDO PPM

```
   Entrada: simbolos_lidos, simbolo_a_codificar
 1 início
 2     contexto ← ultimos n simbolos_lidos
 3     para cada ordem de n a 0 faça
 4         no ← busca_simbolo(contexto, simbolo_a_codificar)
 5         se no não for nulo então
 6             codifica_simbolo(no)
 7             retorna
 8         fim
 9         senão
10             codifica_escape()
11             encurta_contexto()
12         fim
13     fim
14 fim
```

O algoritmo original e muitas de suas variações utilizam um tamanho máximo de contexto. Experimentos indicam que melhores taxas de compressão são obtidas ao utilizar contextos cujo tamanho máximo (n) está compreendido no intervalo de três a sete (Moffat, 1990). Também existem variações que não limitam o tamanho do contexto (Cleary e Teahan, 1997). No entanto, seu consumo de memória é elevado, apesar da preocupação

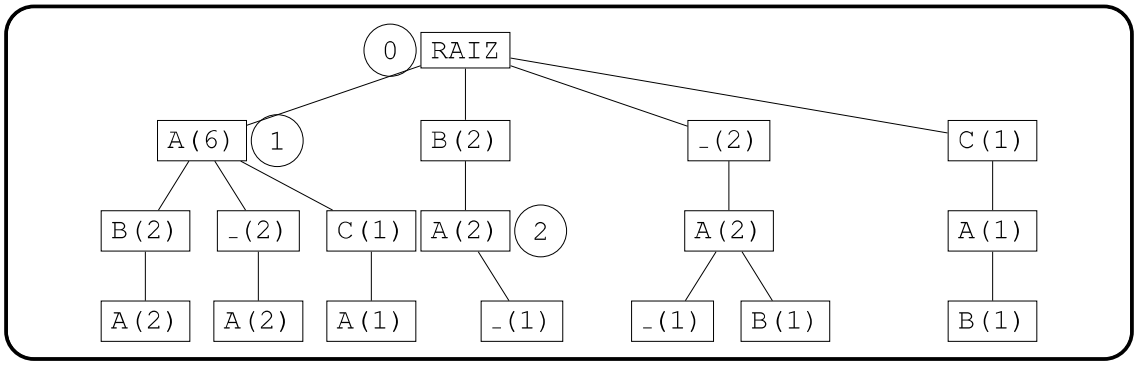


Figura 2. Árvore de contexto (ordem máxima = 2)

no uso de estruturas de dados que minimizem esse custo.

A Figura 2 mostra a árvore de contextos que seria gerada para o texto usado como exemplo, considerando um tamanho máximo de contexto igual a 2 e um universo de símbolos composto por 'A', 'B', 'C' e '_'. O contexto de cada nó é identificado pela concatenação do símbolo do nó atual e dos símbolos dos nós ascendentes. Para cada nó é também armazenada a frequência de ocorrência do contexto correspondente. Dentro de um nó os filhos são ordenados pela frequência. Essa organização visa obter menores tempos na busca de um símbolo (linha 4 do pseudo-código). Caso esse símbolo seja muito frequente, poucos nós deverão ser visitados até que ele seja encontrado.

O nível zero é reservado para o contexto vazio. Todos os símbolos possíveis aparecem como filhos do raiz com uma frequência não nula, para que sua probabilidade de ocorrência seja superior a zero. Isso garante que símbolos que nunca sucederam um contexto específico possam ser codificados quando o contexto for encurtado até ficar vazio.

Os círculos indicam os nós que fazem parte do contexto atual, ou seja, os nós que identificam os símbolos presentes em 'BA', em cada uma das ordens, de zero a dois. A operação que encurta contexto (linha 11 no pseudo-código) pode ser vista como a navegação de um nó de ordem maior para um nó de ordem menor.

A taxa de compressão dos algoritmos PPM depende fortemente da existência de padrões de repetição nos símbolos a processar. Isso é bastante comum em textos, onde as mesmas palavras (ou compostas pelo mesmo radical) costumam aparecer com frequência. Isso leva à geração de árvores em que cada pai possua poucos filhos muito frequentes e muitos filhos pouco frequentes. Nesses casos, as sequências de caracteres que costumam aparecem juntas são codificadas com poucos bits, uma vez que a probabilidade de ocorrência dos caracteres dentro dessas palavras comuns será alta.

Consideração uma organização de arquivos orientado a coluna, em que os valores de cada coluna são armazenados de forma consecutiva, a expectativa é que sejam encontrados ainda mais padrões do que o que se costuma encontrar em arquivos de texto convencionais. A próxima seção investiga como a natureza dos dados encontrados em arquivos orientados a colunas se relaciona com o PPM e com outros compressores.

4. Experimentos

Os experimentos desta seção mostram como diversos compressores de texto se comportam ao comprimir dados orientados a colunas. Os compressores testados são o PPM, codificação aritmética, BWT e LZ. Todos foram implementados em C. Os dois primeiros foram criados como parte deste trabalho. Para os dois últimos foram utilizados os compressores BZIP2 e GZIP, respectivamente. Nenhuma *flag* de otimização foi utilizada na compilação/execução dos códigos-fontes.

Os dados das colunas foram gerados a partir do TPC-H, um conhecido *benchmark* usado para avaliar a performance de processamento de transações de bancos de dados (Council, 2008). O modelo de dados do TPC-H é composto por oito tabelas (PART, SUPPLIER, PARTSUPP, CUSTOMER, NATION, LINEITEM, REGION, ORDERS). O gerador de dados foi configurado com um fator de escala igual a 1, o que resultou em um volume de dados de aproximadamente 1 GB.

Após a geração dos dados, as tabelas foram segmentadas de modo a se aproximar da organização física de arquivos empregada em SGBDS orientados a colunas. Para isso, uma tabela com x colunas foi dividida em x arquivos distintos. Em cada arquivo os valores da coluna respectiva foram adicionados consecutivamente, separados um do outro por um símbolo de uso reservado.

Foram selecionadas para compressão apenas colunas que armazenassem tipos de dados textuais e tivessem alta cardinalidade. Diversas colunas que satisfaziam os critérios foram avaliadas. De modo geral, em todas elas o resultado foi semelhante. Desse modo, foi escolhida a coluna COMMENT da tabela CUSTOMER como referência.

A primeira análise é voltada exclusivamente ao método de compressão PPM. A intensão é descobrir o melhor tamanho de contexto para o domínio de dados escolhido de modo a obter melhores taxas de compressão sem que isso acarrete em perdas significativas de desempenho. A relação entre esses dois fatores (compressão x desempenho) se dá basicamente pelo tamanho da árvore gerada. Contextos curtos geram árvores menores. Assim, gasta-se menos tempo na manutenção da árvore. Por outro lado, a probabilidade de ocorrência de um símbolo qualquer tende a ser menor, o que diminui a taxa de compressão.

A Figura 3 apresenta resultados empíricos relacionando o tamanho do contexto aos fatores desempenho (gráfico da esquerda) e taxa de compressão (gráfico da direita). O desempenho é medido como o tempo necessário em milissegundos para realizar a compressão. A taxa de compressão é medida em bits por código(bpc), que indicam quantos bits são necessários para compactar cada byte do arquivo de entrada. Foram testadas diversas versões do PPM, variando o tamanho máximo do contexto de dois (PPM-2) até sete (PPM-7). Os gráficos permitem ver como os resultados variam conforme parcelas maiores do arquivo COMMENT são processadas.

Como pode-se ver, o PPM-2 obteve o pior desempenho nos dois fatores analisados. Apesar de pouco tempo ser gasto na geração da árvore, muito tempo é investido na busca de nós a partir de um pai. Como os nós filhos são ordenados pela frequência, a busca por nós com baixa frequência provoca um acesso a um maior número de filhos até que se encontre o nó correto. Além disso, como o arquivo comprimido é maior, perde-se mais tempo em operações de gravação do arquivo de saída.

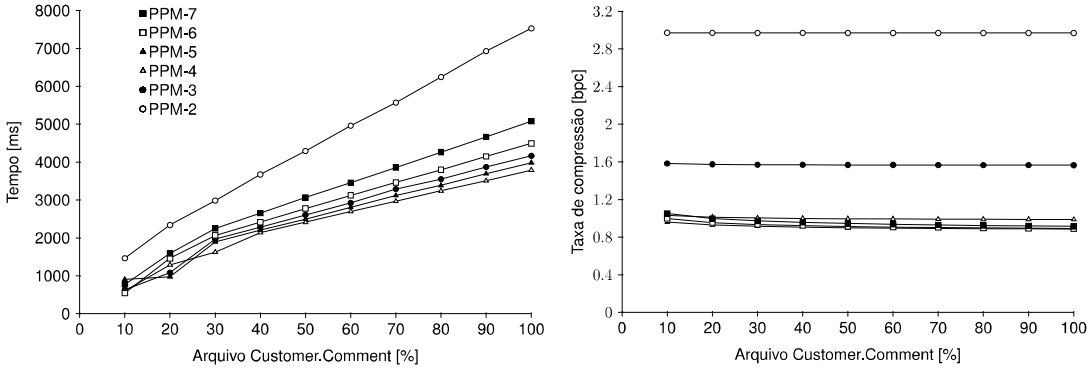


Figura 3. Diferentes versões do PPM variando o tamanho máximo do contexto

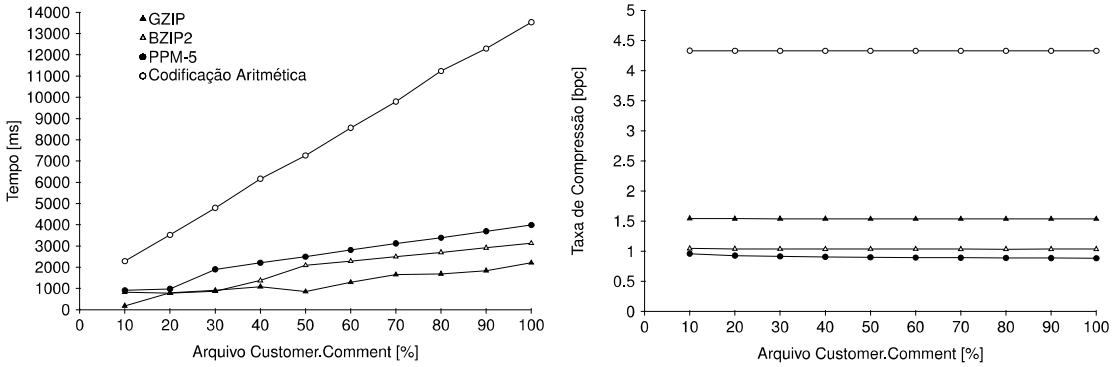


Figura 4. Algoritmos de compressão processando um arquivo orientado a colunas

O PPM-7 também gerou resultados insatisfatórios quanto ao tempo de processamento. Isso ocorre em boa parte porque a manutenção da árvore requer muito trabalho. Por outro lado, o PPM-7 obteve uma boa taxa de compressão. No entanto, a taxa é semelhante aos resultados obtidos por versões do algoritmo que usaram tamanhos máximos de contexto menores. Analisando os dois fatores em conjunto, percebe-se que os PPM-4 e PPM-5 apresentam uma boa relação custo-desempenho, sendo que o PPM-4 é levemente superior no quesito desempenho enquanto o PPM-5 é melhor na taxa de compressão. Como o objetivo é atingir boas taxas de compressão sem perdas significativas de desempenho, a versão PPM-5 foi utilizada no demais experimentos.

Os próximos gráficos(Figura 4) comparam o desempenho e a taxa de compressão de todos os algoritmos avaliados. Novamente a medição foi feita considerando parcelas do arquivo COMMENT. Os resultados mostram que a compressão aritmética pura perde nos dois fatores. Os outros três algoritmos apresentam um custo benefício semelhante, sendo que o GZIP apresenta o menor tempo de execução enquanto o PPM-5 apresenta a maior taxa de compressão. Caso a intenção seja otimizar a ocupação de espaço em disco, a alternativa que implementa o PPM seria preferível.

Para finalizar, os gráficos da Figura 5 incluem na comparação com a coluna COMMENT o corpus CALGARY. Esse corpus possui uma coleção de arquivos de variados formatos e tamanhos, e é bastante utilizado como *benchmark* de compressão de dados (Arnold e Bell, 1997). A tabela CUSTOMER, de onde foi extraída a coluna COMMENT, também foi adi-

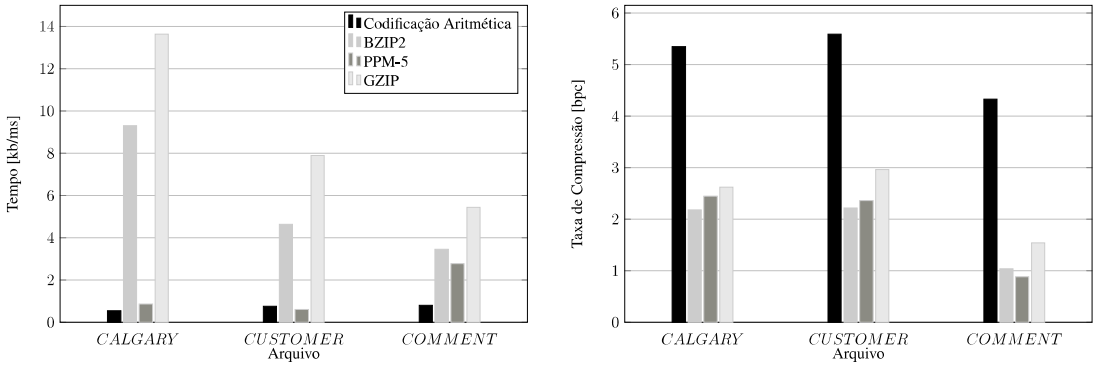


Figura 5. Algoritmos de compressão processando arquivos variados

cionada. Como essa tabela é orientada a registros, cabe fazer uma análise referente ao comportamento dos algoritmos de acordo com a organização física dos dados. Os arquivos CALGARY, CUSTOMER e COMMENT ocupam respectivamente 3.2, 23.9 e 10.8 mbytes.

O gráfico da esquerda exibe a velocidade de processamento, medida em kbytes processados por milissegundo. Aqui pode-se ver que BZIP2 e GZIP são visivelmente mais rápidos do que o PPM e a codificação aritmética na compressão de arquivos convencionais ou da tabela orientada a registros. No entanto, essa relação de desempenho é menos impactante na compressão do arquivo orientado a colunas. Além disso, a velocidade do GZIP e BZIP2 diminui na compressão do arquivo COMMENT, enquanto a velocidade do PPM aumenta. Isso mostra que a proximidade de valores de um mesmo domínio impulsiona o PPM e causa um efeito contrário em seus principais concorrentes.

Já o gráfico da direita exibe a taxa de compressão. Todos os compressores obtiveram melhores resultados no arquivo orientado a colunas. Isso demostra que a redundância desse tipo de arquivo é bem explorada pelos algoritmos. O que vale a pena destacar aqui é a distinção que existe na compressão do CALGARY e CUSTOMER em comparação à COMMENT. Nos dois primeiros, BZIP2 e GZIP foram superiores ao PPM. Já no arquivo COMMENT essa relação se inverteu. Esse resultado sugere que métodos baseados em contexto como o PPM são mais eficazes na compressão de informações textuais cujo universo de valores aceitos pertence a um domínio de dados mais restrito.

5. Conclusões

Arquivos orientados a colunas são realmente bem explorados por compressores de dados baseados em padrões. Os experimentos realizados mostraram que as taxas de compressão são maiores quando se lida com dados bem comportados cujos valores pertencem a um domínio de dados bem definido, uma vez que a entropia tende a ser menor.

Outro ponto importante levantado nos experimentos foi a descoberta de que o método de compressão PPM se mostra particularmente viável para esse tipo de dados, apresentado taxas de compressão superiores aos métodos concorrentes e um tempo de processamento não muito superior. Aqui cabe ressaltar que também foram realizados experimentos medindo tempo de execução na descompressão. Esses resultados foram omitidos por serem análogos aos resultados obtidos na compressão, posicionando o PPM como método de compressão com desempenho razoavelmente competitivo.

Os resultados obtidos servem de motivação para a investigação de formas de melhorar o desempenho do PPM. Nessa linha de pesquisa, um dos pontos a serem explorados surgiu de uma constatação feita durante os experimentos. Conforme a Figura 3 indica, a taxa de compressão do PPM permanece constante ao longo do processamento do arquivo orientado a colunas. Isso sugere que a árvore de contextos existente após o processamento do trecho inicial do arquivo apresenta uma relação de probabilidades similar à arvore de contextos existente após o processamento de todo o arquivo. Assim, a ideia a investigar é a interrupção na manutenção da árvore de contexto quando alguns critérios forem atingidos, na expectativa de que a árvore existente seja um modelo de predição bom o suficiente para a codificação dos próximos símbolos.

Referências

Abadi, D., Madden, S., e Ferreira, M. (2006). Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM.

Abadi, D. J., Boncz, P. A., e Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665.

Arnold, R. e Bell, T. (1997). A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. DCC'97. Proceedings*, pages 201–210. IEEE.

Burrows, M. e Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm (relatorio tecnico).

Cleary, J. G. e Teahan, W. J. (1997). Unbounded length contexts for ppm. *The Computer Journal*, 40(2 and 3):67–75.

Council, T. P. P. (2008). Tpc-h benchmark specification. [Online; accessado em 19 de janeiro de 2016].

Deutsch, L. P. (1996). Deflate compressed data format specification version 1.3.

Han, J., Haihong, E., Le, G., e Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE.

Huffman, D. A. et al. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Moffat, A. (1990). Implementing the ppm data compression scheme. *Communications, IEEE Transactions on*, 38(11):1917–1921.

Westmann, T., Kossmann, D., Helmer, S., e Moerkotte, G. (2000). The implementation and performance of compressed databases. *ACM Sigmod Record*, 29(3):55–67.

Witten, I. H., Neal, R. M., e Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540.

Wu, K., Otoo, E. J., e Shoshani, A. (2002). Compressing bitmap indexes for faster search operations. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 99–108. IEEE.

Ziv, J. e Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536.