Representação e Computação de Cubos de Dados Completos ou Parciais em Clusters de Computadores de Baixo Custo

Angélica Aparecida Moreira
Universidade Federal de Ouro Preto

Dissertação submetida ao
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto
como requisito parcial para obtenção do título de Mestre em Ciência da Computação

M838r Moreira, Angélica Aparecida.
Representação e computação de cubos de dados completos ou parciais em clusters de computadores de baixo custo [manuscrito] / Angélica Aparecida Moreira – 2012.
xx, 43 f.: il.; grafs.; tabs.

Orientador: Prof. Dr. Joubert de Castro Lima.

Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação. Programa de Pós-graduação em Ciência da Computação.
Área de concentração: Sistemas de computação

1. Computação de alto desempenho - Teses. 2. Banco de dados - Armazém de dados - Cubo de dados - Teses. 3. Processamento analítico online (OLAP) - Teses. I. Universidade Federal de Ouro Preto. II. Título.

CDU: 004.65

Catalogação: sisbin@sisbin.ufop.br

Ata da Defesa Pública de Dissertação de Mestrado

Aos 13 dias do mês de julho de 2012, às 14 horas na Sala Multimídia do Instituto de Ciências Exatas e Biológicas (ICEB), reuniram-se os membros da banca examinadora composta pelos professores: **Prof. Dr. Joubert de Castro Lima (presidente e orientador), Prof. Dr. Clodoveu Augusto Davis Junior e Prof. Dr. Ricardo Augusto Rabelo Oliveira**, aprovada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, a fim de arguirem a mestranda **Angélica Aparecida Moreira**, com o título **"Representação e Computação de Cubos de Dados Completos ou Parciais em Clusters de Computadores de Baixo Custo"**. Aberta a sessão pelo presidente, coube à candidata, na forma regimental, expor o tema de sua dissertação, dentro do tempo regulamentar, sendo em seguida questionada pelos membros da banca examinadora, tendo dado as explicações que foram necessárias.

Recomendações da Banca:

(X) Aprovada sem recomendações

( ) Reprovada

( ) Aprovada com recomendações: _____

Banca Examinadora:

Prof. Dr. Joubert de Castro Lima

Prof. Dr. Ricardo Augusto Rabelo Oliveira

Prof. Dr. Clodoveu Augusto Davis Junior

Prof. Dr. Fabrício Benevenuto de Souza
Coordenador do Programa de Pós-Graduação em Ciência da Computação
DECOM/ICEB/UFOP

Prof. Dr. Fabrício Benevenuto de Souza
Coordenador do Programa de Pós Graduação em
Ciência da Computação - DECOM/ICEB/UFOP

Ouro Preto, 13 de julho de 2012.

Dedico este trabalho aos meus pais, Walter e Sônia, pelo incentivo amor e carinho.

Representação e Computação de Cubos de Dados Completos ou Parciais em Clusters de Computadores de Baixo Custo

Resumo

A abordagem PnP (*Pipe 'n Prune*) é considerada uma das abordagens mais promissoras da literatura para computação de cubos em arquiteturas de computadores com memória distribuída. Infelizmente, a abordagem PnP gera uma enorme quantidade de dados redundantes. No geral, a PnP não considera a uniformidade nos dados, denominada *skew*. Não considerar o *skew* no particionamento da carga de trabalho impõe máxima redundância de dados, mesmo com dados uniformes. Diante deste cenário, foi desenvolvida a abordagem P2CDM (acrônimo de *Parallel Cube Computation with Distributed Memory*), que possui comunicação minimizada e gera redundância de dados sob demanda, dependendo do grau de uniformidade dos dados. Neste sentido, a abordagem P2CDM permite a computação de cubos completos a partir de um certo grau de uniformidade nos dados e cubos parciais quando o grau de uniformidade nos dados ultrapassar um limite predefinido. Os experimentos demonstram que as abordagens PnP e P2CDM possuem acelerações similares, porém a abordagem P2CDM é 20-25% mais rápida e consome 30-40% menos memória em cada nó do *cluster*, quando comparada com a abordagem PnP.

Full and Partial Data Cube Computation and Representation over Commodity PCs

Abstract

The PnP (Pipe 'n Prune) approach is considered one of the most promising approaches for cube computation over distributed memory computer architectures. Unfortunately, it generates a huge amount of redundant data. In general, PnP does not consider data uniformity, named skew, when partitioning its workload and, thus, it imposes a maximum data redundancy even with uniform data. Due to this scenario, we implement P2CDM (acronym for Parallel Cube Computation with Distributed Memory) approach which has minimized communication and low data redundancy, depending on the data skew. In this sense, P2CDM approach enables full cube computation from a input data with low skew and partial cube computation from high skew input data. Our experiments demonstrated that both approaches have similar speedup, but P2CDM approach is 20-25% faster and consumes 30-40% less memory at each host of the cluster, when compared to the PnP approach.

Declaração

Esta dissertação é resultado de meu próprio trabalho, exceto onde referência explícita é feita ao trabalho de outros, e não foi submetida para outra qualificação nesta nem em outra universidade.

Angélica Aparecida Moreira

Agradecimentos

Em primeiro lugar gostaria de agradecer a Deus por todas as oportunidades que colocou em meu caminho e pela força que me deu para concluir este trabalho. Agradeço também aos meus pais, Walter e Sônia, por terem me dado apoio para a realização dos meus sonhos, além do amor e carinho incondicional para comigo.

Meus sinceros agradecimentos ao meu orientador, o professor Dr. Joubert de Castro Lima, por ter me introduzido neste campo de pesquisa e por ter me dado o devido auxílio e orientação durantes estes últimos dois anos. O meu agradecimento especial ao professor Dr. David Menotti Gomes, por toda a orientação, atenção, auxílio e amizade que me proporcionou durante esta minha jornada, espero que conservemos esta amizade.

E finalmente, eu gostaria de agradecer aos Programa de Pós Graduação em Ciência da Computação, da UFOP, e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) por terem me contemplado com uma bolsa de estudos, o que permitiu minha dedicação exclusiva neste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

"The only place where success comes before work is in the dictionary."
— Albert Einstein

Capítulo 1

Introdução

Neste capítulo é apresentada uma visão geral do trabalho que foi realizado e também sua relevância científica.

1.1 Considerações iniciais

Como forma de contornar as limitações da programação sequencial e aumentar o desempenho de sistemas de computação, a programação paralela começou a ser vista como uma alternativa. Essa alternativa viabilizou os avanços das técnicas e arquiteturas de bancos de dados paralelos, possuindo ótimo desempenho ao processar consultas simultaneamente.

Mesmo que os sistemas de bancos de dados consigam processar consultas de forma paralela, são necessárias técnicas eficientes de extração de informação sumarizada a partir de banco de dados hierárquicos. Diante desse cenário, o operador relacional cubo de dados foi introduzido em (Gray, Chaudhuri, Bosworth, Layman, Reichart, Venkatrao, Pellow & Pirahesh 1997). O operador cubo de dados é considerado uma generalização do operador *group-by*, no qual a informação é organizada multidimensionalmente, possibilitando a exploração dos dados usando múltiplas perspectivas do processo decisório, chamadas dimensões, e múltiplas hierarquias em cada dimensão.

Seguindo este raciocínio, um cubo completo corresponde a todos os *group-bys* possíveis a partir de um conjunto de dimensões organizadas segundo múltiplas hierarquias. Um cubo parcial é um subconjunto de um cubo completo onde alguns *group-bys* não são

computados. Como exemplo de cubos parciais temos os chamado cubos iceberg. Um cubo iceberg é um tipo de cubo que computa apenas as porções de *group-bys* com valores agregados que satisfazem um certo limiar. Na linguagem de banco de dados SQL, isso corresponderia à cláusula HAVING.

A criação de cubos é um problema exponencial sob o ponto de vista do consumo de tempo de processamento e espaço de armazenamento. A computação paralela de cubos multidimensionais, em especial utilizando memória distribuída, tem sido estudada pela comunidade desde o artigo seminal de (Gray, Chaudhuri, Bosworth, Layman, Reichart, Venkatrao, Pellow & Pirahesh 1997) como forma de atenuar o impacto que a exponencialidade traz na resolução do problema. Dentre as abordagens distribuídas existentes podemos citar a RP (*Replicated Parallel BUC*), BPP (*Breadth-first writing, Partitioned, Parallel-BUC*), ASL(*Affinity Sip List*), PT (*Partitioned Tree*) apresentadas em (Ng, Wagner & Yin 2001), a *"Pipe 'n Prune"*(PnP) apresentado em (Chen, Dehne, Eavis & Rau-Chaplin 2008), e a abordagem Brown Dwarf apresentada em (Doka, Tsoumakos & Koziris 2011).

De uma forma geral, a abordagem PnP é a única a garantir aceleração linear na computação de cubos em arquiteturas com memória distribuída, sejam cubos completos quanto parciais, em especial cubos iceberg. Este resultado se deve em boa parte ao balanceamento de carga proposto pela abordagem PnP que, ao fim da geração de cada árvore na floresta PnP, gera uma nova base local, eliminando os dados redundantes. A base local é particionada em p outras, onde p é o número de nós de processamento no *cluster*. Em seguida cada partição p é enviada a um dos nós do *cluster*. Este processo é repetido até que o cubo seja computado. Cubos *skewed* também são computados eficientemente pela abordagem PnP. O *skew* indica a porcentagem de uniformidade do dado, ou seja, quando o *skew* é zero os atributos de uma relação estão uniformemente distribuídos na base e quando o *skew* é diferente de zero os atributos de uma relação possuem frequências distintas na base. De uma forma geral, bases reais são *skewed*.

Infelizmente, a abordagem PnP adota uma estratégia de particionamento de cubos que não minimiza a comunicação entre os nós de processamento, e não se preocupa com a geração de dados redundantes na implementação do operador cubo. A abordagem PnP implementa a redundância máxima de dados, mesmo para os atributos com baixo *skew* na base. Diante deste cenário, é apresentada neste trabalho a abordagem P2CDM (acrônimo de *Parallel Cube Computation with Distributed Memory*), para a computação de cubos completos ou parciais, incluindo cubos iceberg, com dados uniformemente distribuídos ($0 < skew < x$) ou *skewed* ($skew > x$, onde x é um limiar predefinido,

seja da quantidade de memória, seja do tempo de processamento, entre outros), que apresenta desempenho satisfatório mesmo no uso de *clusters* de computadores de baixo custo.

A abordagem P2CDM adota redundância de dados apenas para valores *skewed* entre todas as dimensões de um *Data Warehouse* (DW). Ao contrário da abordagem PnP, a abordagem P2CDM considera a distribuição prévia dos atributos de forma a não haver redundância. Esta distribuição permite que a abordagem P2CDM gere agregações redundantes à medida que cada nó de processamento esgota, por exemplo, a capacidade de armazenamento para um determinado atributo. Ao contrário da abordagem PnP, a abordagem P2CDM efetua uma comunicação a todo *cluster* por nó de processamento, enquanto a PnP efetua d comunicações, onde d é o número de dimensões de um cubo, por nó.

Os experimentos demonstram que as abordagens PnP e P2CDM possuem acelerações similares, porém a abordagem P2CDM, além de permitir que cubos completos sem redundância sejam computados, também possui menor tempo de execução e menor consumo de memória nos nós de processamento, se comparada à abordagem PnP, possibilitando assim a computação de cubos massivos em *clusters* de computadores de baixo custo.

Os demais capítulos deste trabalho encontram-se organizados da seguinte maneira: O Capítulo 2 apresenta os conceitos básicos para uma correta compreensão do trabalho. O Capítulo 3 apresenta os trabalhos correlatos. O Capítulo 4 descreve a abordagem P2CDM, para computação de cubos para base de dados em ambiente com memória distribuída. O Capítulo 5 apresenta os experimentos e uma discussão dos mesmos. O Capítulo 6 conclui o trabalho e apresenta os trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este capítulo está dividido em dez seções, a seção 2.1 descreve o que é um armazém de dados. A seção 2.2 descreve como e por quem o termo OLAP foi introduzido, bem como o que vem a ser o mesmo. A seção 2.3 descreve o que são hierarquias. A seção 2.4 descreve as operações OLAP. A seção 2.5 descreve o tipo abstrato de dados cubo, o porquê de seu surgimento e os benefícios de seu uso. A seção 2.6 conceitua o que são células em um cubo de dados. A seção 2.7 descreve o que são medidas e seus tipos. A seção 2.8 descreve as estratégias de computação de cubo de dados. A seção 2.9 descreve os esquemas multidimensionais existentes. Por fim, a seção 2.10 conceitua os modelos de arquitetura de memória existentes.

2.1 Data Warehouse

Um Armazém de Dados ou *Data Warehouse* (DW) é um repositório estruturado, integrado, variado ou particionado em função do tempo e não volátil, que auxilia no gerenciamento do processo de tomada decisões (Inmon & Hackathorn 1994). As quatro expressões chaves: (1) estruturado, (2) integrado, (3) variado em função do tempo e (4) não volátil; diferenciam o DW de outros sistemas de repositórios, como os sistemas de bancos de dados relacionais, sistemas de processamento de transações e os sistemas de arquivos.

Um DW integra fontes de dados heterogêneas, como tabelas relacionais, arquivos de texto, objetos serializados e arquivos XML, em um único repositório analítico de dados. Técnicas de limpeza e integração de dados são aplicadas para garantir consistência na

base. Um DW deve armazenar dados históricos em um local fisicamente separado dos bancos de dados operacionais das organizações. Cada tópico em um DW deve conter, tanto explícita quanto implicitamente, a perspectiva tempo.

2.2 OLAP

O termo *On-line Analytical Processing* (OLAP) foi criado e tornado público por (Codd, Codd & Salley 1993). Este termo refere-se a um conjunto de ferramentas que são utilizadas para resumir, consolidar, visualizar, aplicar formulações e sintetizar dados de acordo com múltiplas dimensões.

Os dados utilizados pelas ferramentas OLAP normalmente estão armazenados em DWs. Cada ferramenta OLAP deve manipular um novo tipo abstrato de dados (TAD), chamado de cubo de dados. Cada uma destas ferramentas utilizam estratégias específicas, devido ao fato de considerarem o modo como os dados são armazenados, sendo classificadas em:

- ***Relational* OLAP (ROLAP):** ferramentas que utilizam Sistemas de Gerenciamento de Banco de Dados (*Database Management System* - DBMS) relacionais para o gerenciamento e armazenamento dos cubos de dados. Elas incluem otimizações para cada DBMS, implementação da lógica de navegação em agregações, serviços e ferramentas adicionais;

- ***Multidimensional* OLAP (MOLAP):** ferramentas que implementam estruturas de dados multidimensionais para armazenar cubo de dados em memória principal ou em disco. Não há utilização de repositórios relacionais para armazenar dados multidimensionais e a lógica de navegação já é integrada a estrutura proposta;

- ***Hybrid* OLAP (HOLAP):** ferramentas que combinam técnicas ROLAP e MOLAP, onde normalmente os dados detalhados são armazenados em base de dados relacionais (ROLAP), e as agregações são armazenadas em estruturas de dados multidimensionais (MOLAP).

2.3 Hierarquias

Hierarquias oferecem uma ordenação prévia nos atributos de uma dimensão, portanto uma dimensão normalmente possui inúmeras hierarquias. Os valores para a dimensão unidades federativas do Brasil são ao todo 27 dados que contemplam todos os vinte e seis estados e um Distrito Federal. As unidades federativas podem ser mapeadas em cinco regiões políticas (Centro-Oeste, Nordeste, Norte, Sul e Sudeste). Os mapeamentos formam um conceito de hierarquia para a dimensão unidades federativas do Brasil, mapeadas em regiões, o conjunto de regiões mapeadas em países, os países em subcontinentes, os subcontinentes em continentes e assim por diante.

Pode existir mais de uma hierarquia conceitual em uma dimensão, baseada nas diferentes perspectivas do usuário. Hierarquias conceituais podem ser fornecidas de maneira manual por usuários de sistemas, especialistas no domínio ou podem ser geradas de maneira automatizada com base na análise estatística da correlação dos dados.

2.4 Operações OLAP

No modelo multidimensional, os dados são organizados em múltiplas dimensões, cada dimensão contem múltiplos níveis de abstração definida conceitualmente como hierarquias. Esta organização fornece aos usuários a capacidade de visualizar dados de diferentes pontos de vista (Han, Kamber & Pei 2006).

Na Figura 2.1, são ilustradas algumas das operações OLAP típicas para dados multidimensionais. No exemplo utilizado tem-se um cubo com três dimensões, que são tempo, disciplina e departamento de uma escola qualquer, sendo a medida a nota e a função de agregação a média.

Algumas dimensões possuem diferentes hierarquias (ou diferentes níveis de sumarização). Se o usuário almeja navegar nos dados a partir dos níveis hierárquicos inferiores para níveis mais altos da hierarquia, dizemos que trata-se de uma operação *roll-up*. O contrário dizemos que trata-se de uma operação *drill-down*. Como exemplo, o usuário pode almejar saber sumarizações na seguinte ordem: dia $->$ mês $->$ trimestre. No exemplo anterior o usuário efetuou uma operação *roll-up*. Se o usuário navegasse trimestre $->$ mês $->$ dia, seria uma operação *drill-down*.

Operações de *slice* ("fatiamento") realizam seleções em uma dimensão de um dado

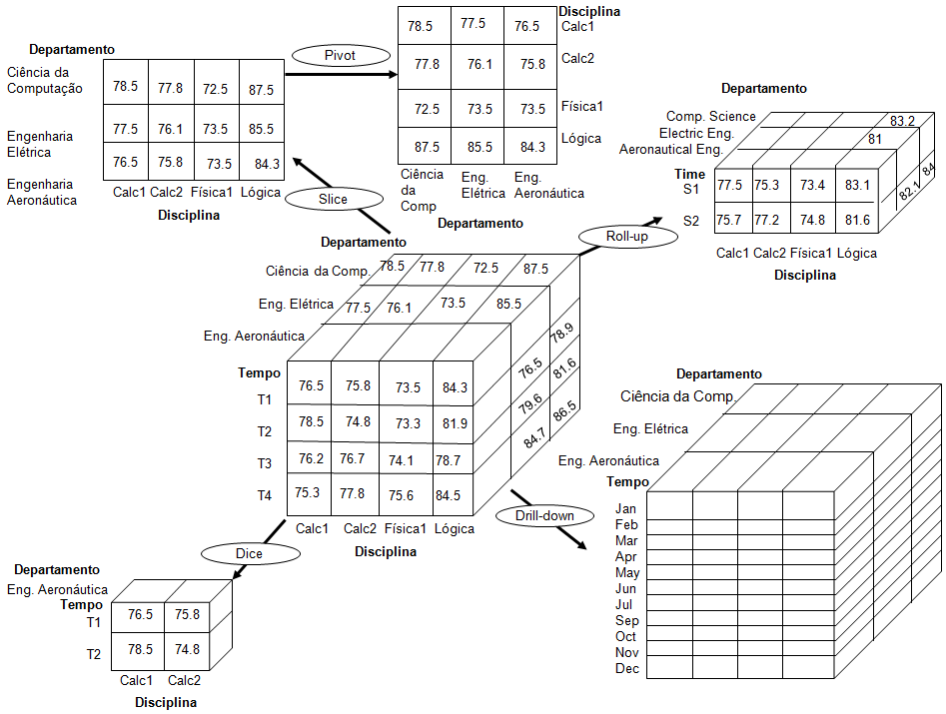

Figura 2.1: Exemplo de Operadores Relacionais

cubo, resultando em um subcubo. A Figura 2.1 mostra uma operação de *slice*, onde os dados (notas) são selecionados de um cubo central utilizando como critério o tempo="Sem1". A operação *dice* define um subcubo realizando uma seleção em duas ou mais dimensões. A Figura 2.1 apresenta uma operação de *dice* no cubo central baseada no seguinte critério de seleção, que incluem três dimensões: (departamento="Eng. Aeronáutica") e (tempo="T1"ou "T2") e (disciplina="Calc1"ou "Calc2").

A operação de pivotação (*pivot*) realiza uma rotação nos eixos dos dados, a fim de mudar o modo de apresentação de um certo dado. Na Figura 2.1 é possível visualizar a operação de pivotação, onde os eixos da dimensão disciplina estão rotacionados e "fatiados"(isso porque o pivotação fora aplicado sobre o resultado da operação de *slice*).

2.5 Cubo de Dados

Dados são extraídos de bancos de dados por meio de consultas, e eles são analisados e visualizados por ferramentas de análise de dados onde são realizados 4 passos:

1. Formulação da consulta que recupera dados relevantes;
2. Extração de dados agregados de uma base de dados em um arquivo ou tabela;
3. Visualização do resultado de forma gráfica;
4. Análise dos resultados e formulação de uma nova consulta.

Para tal processo de extração de dados comumente são utilizados operadores relacionais de agregação, tais como o operador *group-by*, que combinam valores de uma dada coluna em um único valor escalar. Este operador combina atributos de uma relação R, formadas pelos atributos A, B ..., sendo A composto por (a1, a2, ...), B composto por (b1, b2, ...), etc., e um conjunto de valores escalares obtidos a partir do cálculo de funções estatísticas, sejam estas SUM, COUNT, MIN, MAX, AVG, entre outras.

A extração de conhecimento de banco de dados é muito utilizada no processo de descoberta de conhecimento (ou *Knowledge Discovery in Databases*), especificamente na fase de mineração de dados (*data mining*), sendo considerada o gargalo do processo de descoberta de conhecimento. Muitas vezes é necessária a geração de histogramas, relatórios com totais e sub-totais, além de tabelas com dados cruzados, conforme ilustra a Figura 2.2 com um exemplo de dados cruzados. Infelizmente, o operador *group-by* não consegue suprir a necessidade de relacionar dados multidimensionais de maneira eficiente (Gray, Chaudhuri, Bosworth, Layman, Reichart, Venkatrao, Pellow & Pirahesh 1997).

Palio	2010	2012	Total(ALL)
Prata	50	85	135
Vermelho	40	115	155
Total(ALL)	90	200	290

Modelo	Ano	Cor	Unidades
Palio	2010	Prata	50
Palio	2010	Vermelho	40
Palio	2010	ALL	90
Palio	2012	Prata	85
Palio	2012	Vermelho	115
Palio	2012	ALL	200
Palio	ALL	ALL	290
Palio	ALL	Prata	135
Palio	ALL	Vermelho	155

Figura 2.2: Exemplo de Cross-Tabulation

Diante destas limitações, (Gray, Chaudhuri, Bosworth, Layman, Reichart, Venkatrao, Pellow & Pirahesh 1997) conceberam o conceito de cubo de dados (*data cube*), ou simplesmente cubo, e o definiram como um operador relacional que gera todas as combinações possíveis de seus atributos de acordo com uma medida. Para gerar tais

combinações ele introduz o conceito do valor ALL, para ser o valor que representa todas as combinações possíveis de um universo de atributos.

O operador cubo de dados é o componente mais importante na modelagem multidimensional de dados e é definido por dimensões e medidas. Medidas (ou fatos) são atributos numéricos, que representam informações a serem analisadas, normalmente relacionadas a medidas estatísticas, mas também podem ser medidas espaciais. Dimensões são perspectivas do processo decisório, permitindo que fatos possam ser analisados. Para realizar tais análises o operador cubo pode ser utilizado em conjunto com outros operadores, a fim de satisfazer diferentes necessidades de visualização, ou com o intuito reduzir o tamanho do cubo a ser computado.

Um cubo de dados é composto por células e cada célula possui valores para cada dimensão, incluindo ALL, e valores numéricos para as medidas. O valor de uma medida é computado para uma determinada célula utilizando níveis de agregação inferiores para gerar os valores dos níveis de agregação superiores. Esta estratégia de computação de cubos é denominada *Top-down* e a ordem inversa é denominada *Bottom-up*, e as mesmas são explicitadas mais a frente, na seção 2.8.

A computação do cubo de dados é considerada um problema exponencial em relação ao tempo de execução e ao consumo de memória. Para uma dada entrada de tamanho n a saída é 2^n, onde n é o número de dimensões de um cubo. O operador cubo é ilustrado na Figura 2.3, onde é calculada a quantidade de carros vendidos, representada pela coluna vendas. Cada tupla possui três valores de atributos e um de medida. Suponha a tupla de entrada (Palio, 2010, Prata), onde n = 3. Isto significa que existem 8 tuplas de saída, isso porque $2^3 = 8$. As tuplas são: [(Palio, 2010, Prata), (Palio, 2010,ALL), (Palio, ALL, Prata), (ALL, 2010, Prata), (Palio, ALL, ALL), (ALL, 2010, ALL), (ALL, ALL, Prata), (ALL, ALL, ALL)].

2.6 Células de um Cubo

Um cubo de dados é composto por diversos subcubos e cada subcubo é composto por diversas células base e células agregadas. Deste modo uma célula em um subcubo base é uma célula base. E uma célula em um subcubo não base é uma célula agregada. Uma célula agregada agrega sobre uma ou mais dimensões, onde cada dimensão agregada é indicada pelo valor especial ALL ("*") na notação da célula.

Modelo	Ano	Cor	Vendas
Palio	2010	Prata	100
Palio	2009	Prata	230

Cubo Completo

Modelo	Ano	Cor	Vendas
Palio	2010	Prata	100
Palio	2009	Prata	230
Palio	2010	ALL	100
Palio	2009	ALL	230
Palio	ALL	Prata	330
Palio	ALL	ALL	330
ALL	2010	Prata	100
ALL	2009	Prata	230
ALL	2010	ALL	100
ALL	2009	ALL	230
ALL	ALL	Prata	330
ALL	ALL	ALL	330

Figura 2.3: Cubo de Dados

Por exemplo, suponha que existe um cubo de dados n-dimensional. Seja a = (a_1, a_2, a_3, ..., a_n, medidas) uma célula de um dos subcubos que constituem um cubo de dados qualquer. A célula a é uma célula m-dimensional, se exatamente m (m $\leq$ n) valores entre (a_1, a_2, a_3, ..., a_n) não são "*". Se m = n, então a é uma célula base, caso contrário, ela é uma célula agregada.

Considere o cubo de dados da Figura 2.1, com as dimensões tempo, departamento e disciplina, e a medida nota. As células (T1, *, *, 78.9) e (*, Ciência da Comp., *, 81.3) são células de 1 dimensão, (T1,*,Calc1, 76.3) é uma célula de 2 dimensões, e (T1, Ciência da Comp., Calc1, 78.8) é uma célula de 3 dimensões. Aqui todas as células base possuem 3 dimensões, enquanto que as células com 1 e 2 dimensões são células agregadas.

Um relacionamento de descendente-ancestral pode vir a existir entre células. Em um cubo de dados n-dimensional, uma célula a = (a_1, a_2, a_3, ..., a_n, medidas) de dimensão i é um ancestral de uma célula b = (b_1, b_2, b_3, ..., b_n, medidas) de dimensão j, e b é um descendente de a, se e somente se i $<$ j e 1 $\leq$ m $\leq$ n, onde $a_m = b_m$ sempre que $a_m \neq$ *. Em particular, uma célula a é chamada de pai de uma célula b, e b de filho de a, se e somente se j = i+1 e b for um descendente de a (Han, Kamber & Pei 2006).

Tomando como base o exemplo anterior, uma célula a = (T1, *, *, 78.9) com uma dimensão e uma célula b (T1, *, Calc1, 76.3) com duas dimensões são ancestrais da célula c = (T1, Ciência da Comp., Calc1, 78.8) que possui três dimensões, e c é descendente de a e b, onde b é pai de c.

2.7 Medidas

Cada célula de um cubo é definida como um par <(d_1, d_2, ..., d_n), medidas>, onde (d_1, d_2, ..., d_n) representam as combinações possíveis de valores de atributos sobre as dimensões. A medida de um cubo de dados é em geral uma função numérica que pode ser avaliada em cada célula na grade de células.

Medidas numéricas podem ser organizadas em três categorias: distributiva, algébrica e holística. A categoria mais simples é a distributiva. Para ilustrá-la, suponha que os dados são particionadas em n conjuntos. A função de agregação é aplicada a cada partição, resultando em n valores agregados. Se o resultado obtido através da aplicação da função aos n valores agregados for o mesmo que o resultado obtido aplicando a função a todo o conjunto sem particionamento, a função pode ser computada de maneira distributiva. Como exemplo temos: count(), sum(), min(), e max().

Uma função de agregação é algébrica se ela pode se computada por meio de uma função algébrica com M argumentos, onde M é um inteiro positivo finito. Cada argumento é obtido através da aplicação de uma função de agregação distributiva. Como exemplo temos a média (avg()) que é calculada através da divisão do resultado da função soma pela frequência ($\frac{sum()}{count()}$), onde ambas são funções de agregação distributivas. Uma função de agregação é holística se não existe uma função algébrica com M argumentos, onde M é uma constante, que caracteriza a computação. Exemplos comuns de função holística incluem mediana, moda e classificação, representadas respectivamente na SQL como median(), mode() e rank().

2.8 Computação de Cubos

A computação do cubo de dados é uma tarefa essencial, uma vez que a pre-computação de parte ou de todo o cubo de dados pode reduzir significamente o tempo de execução e melhorar o desempenho de sistemas OLAP. No entanto, tal computação é um dos problemas mais relevantes e difundidos na área de DW. Devido ao fato do problema possuir complexidade exponencial em relação ao número de dimensões, a materialização completa de um cubo envolve uma grande quantidade de células e uma quantidade substancial de tempo para sua geração.

Dado um cubo base, existem três métodos de se gerar as agregações remanescentes:

a não materialização, a materialização completa e a materialização parcial.

Na não materialização, cubos base não são pre-computados, o que leva a uma computação imediata extremamente custosa, que pode ser extremamente lenta.

A materialização completa pre-computa todas as agregações possíveis de um dado cubo, tendo como resultado um cubo completo. Este método possibilita que o tempo de resposta a uma dada consulta seja extremamente rápido, uma vez que o cubo completo está previamente computado. No entanto, isso pode exigir uma grande quantidade de espaço em memória.

Finalmente, tem-se o método de materialização parcial ou seletiva, que computa um sub-conjunto específico de um universo de possibilidades. Alternativamente, é possível computar um sub-conjunto de um cubo de dados que contém somente células que satisfazem um dado critério especificado pelo usuário. Este tipo de cubo de dados é chamada de cubo iceberg (Beyer & Ramakrishnan 1999a). Existe uma outra técnica, chamada de *shell fragment*, onde cubos pequenos (com 3 a 5 dimensões) são computados para formar cubos completos. As lacunas (junções de dois ou mais cubos pequenos) são computadas a medida que forem necessárias. Este tipo de cubo é chamada cubo *shell* (*shell cube*) (Li, Han & Gonzalez 2004). Finalmente, temos a sumarização semântica de cubos de dados, chamada de cubos *closed* (*closed cubes*) (Xin, Shao, Han & Liu 2006) ou *quotient cubes* (Lakshmanan, Pei, U & Han 2002), onde um conjunto de células de um cubo com medidas idênticas são colapsadas em uma única abstração, chamada *closed cell* ou células de classe.

A materialização parcial representa um interessante equilíbrio entre o espaço de armazenamento e o tempo de resposta. No entanto, a computação do cubo completo continua sendo importante. Os avanços alcançados na computação de cubos completos são normalmente adotados na computação de cubos parciais.

Dado um cubo base, a computação do operador cubo pode utilizar a estratégia *Top-down* ou *Bottom-up* para a geração dos subcubos remanescentes. A Figura 2.4 ilustra a geração de um cubo de dados de 4 dimensões por meio da estratégia *Top-down*. Seja ABCD o cubo base, os subcubos de 3 dimensões são: ABC, ABD, ACD e BCD; que podem utilizar os resultados do cubo base para serem computados. Os resultados da computação do subcubo ACD pode ser utilizado para computar AD, que consequentemente pode ser utilizado para computar A. Essa computação compartilhada permite que a estratégia *Top-down* compute agregações em múltiplas dimensões. Os valores agregados intermediários podem ser reutilizados para a computação de subcubos

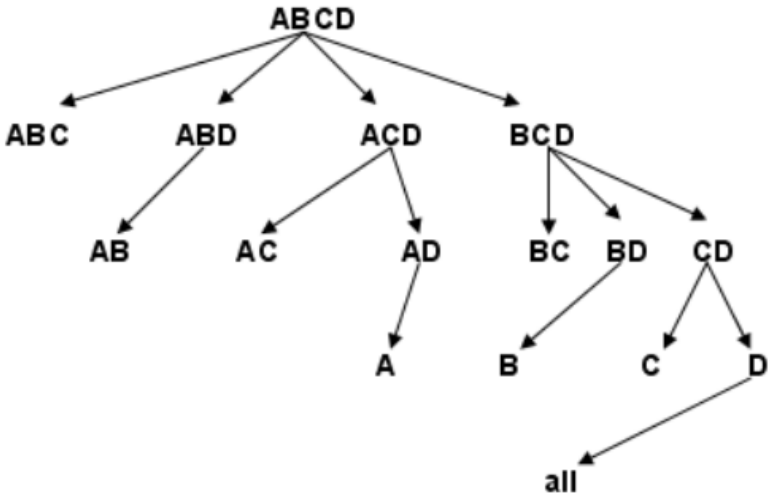

Figura 2.4: Estratégia Top-down de computação de cubos

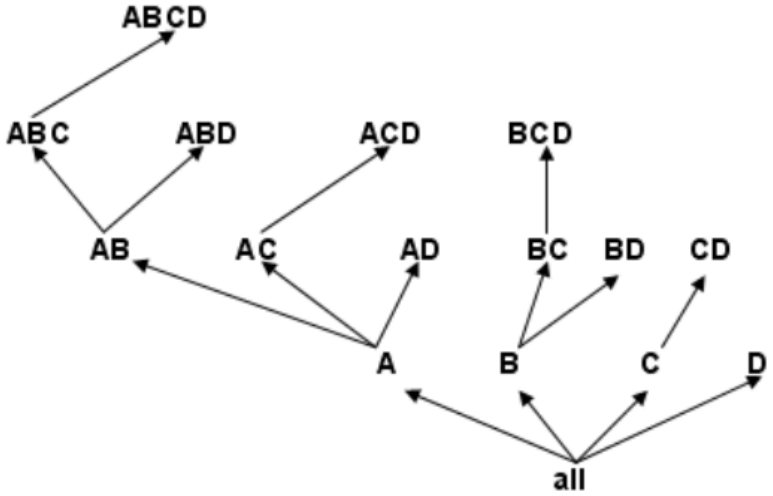

Figura 2.5: Estratégia Bottom-up de computação de cubos

descendentes sucessivos.

A Figura 2.5 ilustra a geração de um cubo de dados de 4 dimensões por meio da estratégia *Bottom-up*. Subcubos de poucas dimensões tornam-se pais de subcubos com mais dimensões. Infelizmente, a computação compartilhada, utilizada na estratégia *Top-down*, não pode ser aplicada quando utilizada a estratégia *Bottom-up*, então cada subcubo descendente necessita ser computado a partir do zero.

2.9 Esquemas Multidimensionais

O modelo multidimensional pode existir na forma de um esquema estrela (*star*), floco de neve (*snowflake*), ou como uma constelação de fatos (*fact constellation*). O esquema estrela apresenta as tabelas de dimensões num padrão radial ao redor da tabela de fatos central. Nele cada dimensão é representada por uma tabela e cada tabela contém um conjunto de atributos. No entanto, aplicações sofisticadas podem exigir múltiplas tabelas de fatos para compartilhar dimensões. Este tipo de esquema pode ser visualizado como uma coleção de estrelas, e portanto é chamado de constelação de fatos (*galaxy schema* ou *fact constellation schema*).

O esquema floco de neve é uma variante do esquema estrela, onde algumas tabelas de dimensões são normalizadas, criando-se um conjunto de tabelas e não uma única tabela para representar uma dimensão. A forma normalizada reduz redundâncias, economizando espaço de armazenamento para as dimensões normalizadas. Infelizmente, a economia de espaço pode ser insignificante se comparada com a magnitude da tabela de fatos. Além disso, a estrutura floco de neve pode reduzir a eficácia na navegação, uma vez que mais junções são necessárias para executar uma pesquisa. Consequentemente, o desempenho do sistema pode degradar.

2.10 Tipos de Memória em Arquiteturas Multiprocessadas

Na arquitetura multiprocessada temos dois modelos de memória, o modelo de memória distribuída e a compartilhada. No modelo de memória distribuída os processadores acessam somente sua própria memória, conforme ilustrado na Figura 2.6. Cada processador possui sua própria memória e uma mudança nos dados de um sistema de memória não afeta os demais sistemas.

Já no modelo de memória compartilhada, ilustrado na Figura 2.7, existe somente um barramento de acesso a memória. Cada processador utiliza a mesma memória de forma compartilhada. Apesar de se ter um barramento de altíssima velocidade, como todos os processadores alteram a mesma memória, podem haver muitos conflitos, o que requer habilidades do desenvolvedor para manter recursos compartilhados sendo acessados de forma segura.

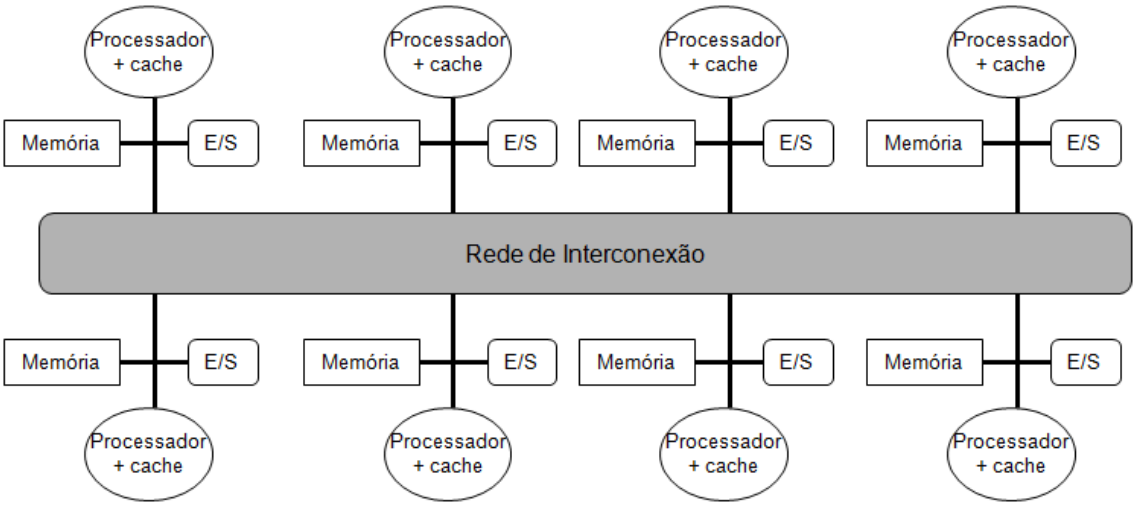

Figura 2.6: Esquema do Modelo de Memória Distribuída

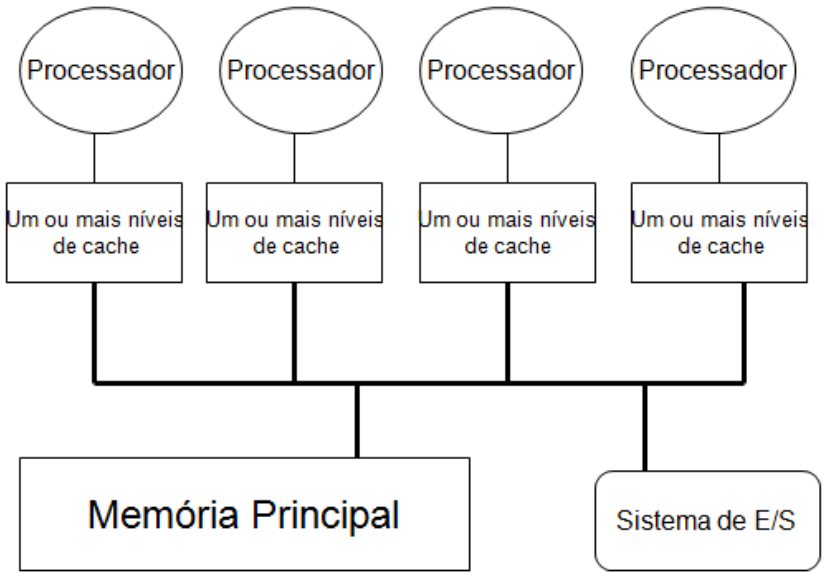

Figura 2.7: Esquema do Modelo de Memória Compartilhada

Capítulo 3

Trabalhos Correlatos

Diante do desafio em computar um cubo completo, surgiram inúmeras abordagens para computá-lo sequencialmente como a MultiWay (Zhao, Deshpande & Naughton 1997), Star (Xin, Han, Li & Wah 2003), *Bottom-Up Computation* (BUC) (Beyer & Ramakrishnan 1999b), Dwarf (Sismanis, Deligiannakis, Roussopoulos & Kotidis 2002), *Multidimensional Direct Acyclic Graph Cubing*(MDAG) (Lima & Hirata 2007), *Multidimensional Cyclic Graph* (MCG) (Lima & Hirata 2011).

Infelizmente, as abordagens citadas acima não foram desenhadas para usufruírem do poder de computação das máquinas com múltiplos núcleos de processamento, ou mesmo um conjunto de máquinas interligadas por uma rede de dados. De uma forma geral, as abordagens sequenciais não conseguem obter desempenho satisfatório quando cubos são computados a partir de bancos de dados massivas.

Neste contexto, diversas abordagens para representação e computação de cubos paralelos foram desenvolvidas. Em um primeiro momento, o foco foi para o uso de memória distribuída, como visto em (DeWitt & Gray 1992), isso porque a arquitetura da solução foi baseada na arquitetura de hardware *shared-nothing*, onde cada processo se comunica com o outro somente através do envio de mensagens pela rede. Nesses trabalhos introdutórios as tuplas são particionadas entre as unidades de armazenamento de cada processo.

Seguindo a mesma direção, (Ng, Wagner & Yin 2001) discutiram o uso de arquiteturas com memória distribuída para computação de cubos. Foi desenvolvida uma abordagem de computação de cubos icebergues utilizando *clusters* de computadores pessoais. Nos algoritmos apresentados, os autores assumem que o cubo completo não foi previamente

computado e, para facilitar a implementação, consideraram somente o contador (*count*) como medida. Além disso não incluíram a agregação ALL para simplificar o projeto dos algoritmos. Utilizaram uma estrutura baseada em ordenação para a implementação de quatro algoritmos, que são: RP (*Replicated Parallel* BUC), BPP (*Breadth-first writing, Partitioned, Parallel-BUC*), ASL (*Affinity SkipList*) e PT (*Partitioned Tree*). Tal estrutura foi utilizada devido ao menor consumo de memória e suporte a computação compartilhada.

Nos testes apresentados em (Ng, Wagner & Yin 2001), foi demonstrado que RP somente é aconselhável para cubos com pouquíssimas dimensões. O mesmo foi constatado para o BPP, porém, de acordo com os autores, é visível nos testes que os tempos diminuíram com o uso do *breadth-first writing*. Apesar do PT ser uma extensão do BPP, ele foi considerado o algoritmo mais indicado para a maioria das situações, perdendo somente quando os cubos são densos e quando é necessário fazer amostragem e refinamento, sendo que nestes casos o ASL é o mais recomendado.

No entanto, as versões distribuídas obtiveram somente um ganho de 50% em relação à versão sequencial. Tais resultados foram obtidos quando utilizado um pouco mais que meio milhão de registros por processador. Infelizmente, a aceleração com 8 a 16 processadores é abaixo do linear.

Desta forma, motivados pelo trabalho (Ng, Wagner & Yin 2001), foi desenvolvido o *Pipe'nPrune* (PnP), apresentado em (Chen, Dehne, Eavis & Rau-Chaplin 2008), como um método híbrido, baseado em ordenação, que integra *Top-down pinping* (Sarawagi, Agrawal & Gupta 1996) para a agregação de dados com a poda *Apriori* proposta na abordagem BUC (Beyer & Ramakrishnan 1999a). A abordagem PnP obteve aceleração próxima do linear para uma grande quantidade de processadores, e em alguns casos até aceleração super linear.

A abordagem PnP foi desenvolvida e adaptada para os seguintes cenários:

- **Ambiente sequencial com uso de memória primária**: foi a base de desenvolvimento dos demais algoritmos dos demais cenários. Este algoritmo mostrou-se o melhor para ser utilizado, quando comparado ao BUC e ao Star-Cubing;
- **Ambiente sequencial com uso de memória externa**: a abordagem PnP não requer estruturas de dados complexas em memória, portanto é aceitável a implementação do método de memória externa para consultas de cubos icebergues massivos. O algoritmo que faz uso de memória externa foi em média duas vezes

mais lento do que a versão em memória principal;

- **Ambiente distribuído com uso de memória externa**: este algoritmo foi o que obteve um melhor desempenho, mesmo em ambientes com bancos de dados massivos, com muitas dimensões e cardinalidade extremamente grande. Foi obtida aceleração próxima do linear, e em alguns casos aceleração superlinear.

Os ótimos resultados obtidos no trabalho de (Chen, Dehne, Eavis & Rau-Chaplin 2008) devem-se à forma de construção dos *group-bys*. Num primeiro momento, são construídos todos os *group-bys* v' que são prefixos de v através de uma única operação de varredura da base previamente ordenada, combinada com a poda do cubo icebergue. Em seguida é utilizada a propriedade de antimonotonia da relação para a computação eficiente dos novos *group-bys*, que são pontos de início de outras operações de *piping* (Chen, Dehne, Eavis & Rau-Chaplin 2008). Este processo é realizado d vezes, onde d é o número de dimensões. O resultado é um cubo de dados parcial com todos os *group-bys* que satisfazem um limiar predefinido pelo usuário.

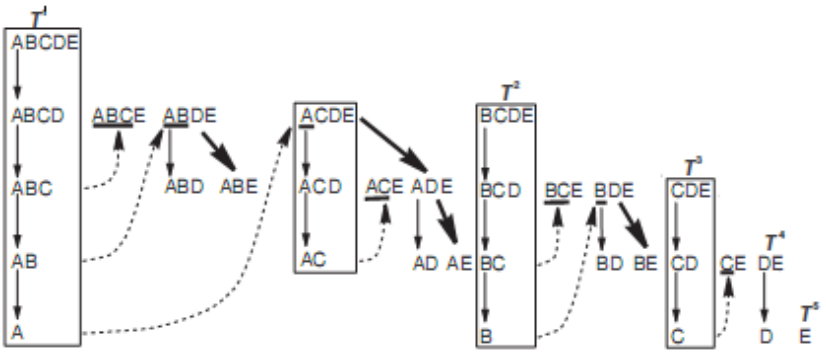

Figura 3.1: Floresta do PnP

Um outro ponto importante da abordagem PnP é a estratégia de balanceamento das cargas de trabalho. Basicamente, ao final da geração de cada árvore na floresta PnP é gerada uma tabela local com base na entrada de dados anteriormente gerada. Esta tabela é utilizada para o cálculo do sub-cubo, removendo-se a primeira dimensão e eliminando dados duplicados a partir de uma operação de ordenação sequencial. Em seguida, uma ordenação global é realizada de forma que cada nó de processamento fique com uma quantidade equilibrada de tuplas. A Figura 3.1, extraída do artigo (Chen, Dehne, Eavis & Rau-Chaplin 2008), ilustra uma floresta PnP, representando um cubo PnP com 5 dimensões. As caixas ilustram os *group-bys* v' gerados pela operação se

varredura/ordenação. As setas representam os pontos de início de outras operações de *piping*.

Recentemente, foi publicado um sistema distribuído para computar, armazenar e atualizar cubos de dados por meio de pares de nós em uma rede, utilizando P2P (*Peer to Peer*) não estruturado. Este sistema, denominado Brown Dwarf (Doka, Tsoumakos & Koziris 2011), computa cubos completos e garante níveis de redundância mínima de dados. O Brown Dwarf utiliza supressão de prefixos e alguns sufixos repetidos, uma vez que utiliza a abordagem Dwarf (Sismanis, Deligiannakis, Roussopoulos & Kotidis 2002) em um conjunto de nós de processamento. Ainda não realizamos testes comparativos com a abordagem Brown Dwarf, porém já é possível afirmar que o fator de aceleração não é diretamente proporcional ao número de nós participantes da computação do cubo, e o cálculo do cubo continua sequencial. Por fim, comunicação na rede introduz enorme latência à medida que cubos massivos são computados, como descrito em (Doka, Tsoumakos & Koziris 2011).

Ainda na literatura temos algumas abordagens que utilizam aplicações com estilo *map-reduce* para computação de cubos em arquiteturas de computadores com memória distribuída. Podemos citar os trabalhos: (You, Xi, Zhang & Chen 2008), (Sergey & Yury 2009), (Wang, Song & Luo 2010) e (Nandi, Yu, Bohannon & Ramakrishnan 2011); como as contribuições mais significativas da área. Infelizmente, tais abordagens também não possuem aceleração linear e são mais usadas para computação em lotes, ao contrário da abordagem P2CDM, Brown Dwarf e PnP.

O uso de computação paralela com memória compartilhada também vem ganhando força. No entanto, o foco não é somente no uso de CPU (*Central Processing Unit*), mas também algoritmos que utilizam GPGPU (*General Purpose Graphical Processing Unit*), devido à promessa de alto desempenho a baixo custo. Essa alternativa foi explorada em (Lauer, Datta, Khadikov & Anselm 2010), onde foi obtido um desempenho superior ao dos algoritmos tradicionais do estado da arte na computação sequencial de cubos completos e/ou parciais.

Uma comparação entre CPU e GPGPU foi feita em (Kaczmarski 2011), mostrando a eficiência da unidade na computação de cubos. Segundo o autor, alguns cubos são computados dez vezes mais rápido do que uma versão paralela simples para computação de cubos usando CPU. O interessante no uso de GPGPU foi o completo aproveitamento da vazão de instruções e da largura de banda, auxiliando na otimização da tarefa de ordenação dos dados, que é muito custosa na CPU. Porém, devido à falta de detalhes

da implementação em CPU, não ficou claro se a implementação paralela foi mesmo otimizada. Sendo assim, até o momento, falta uma comparação entre uma implementação paralela com memória compartilhada para CPU, que seja otimizada, e a implementação descrita no artigo (Kaczmarski 2011), para constatar o efeito real do impacto do uso de GPGPU para computação de cubos.

3.1 A Abordagem MCG e as Restrições Impostas

Para a geração das agregações, a abordagem P2CDM utiliza a abordagem *Multidimensional Cyclic Graph* (MCG), descrita em (Lima & Hirata 2011). A MCG é uma abordagem para computação de cubos completos ou icebergues que utiliza grafos para representação dos cubos. Cada nível da árvore representa uma dimensão, e cada nó um valor do atributo. As tuplas são inseridas uma a uma no cubo base, e uma célula no cubo é representada pelo caminho partindo da raiz até um nó. Cada nó possui quatro campos: ponteiros para os possíveis descendentes, um conjunto de valores de medida, um identificador associado e um valor de casamento (valor único que identifica o nó no cubo). Na Figura 3.2 é possível visualizar um fragmento de um grafo de um cubo base ABC, e como os nós são representados no mesmo.

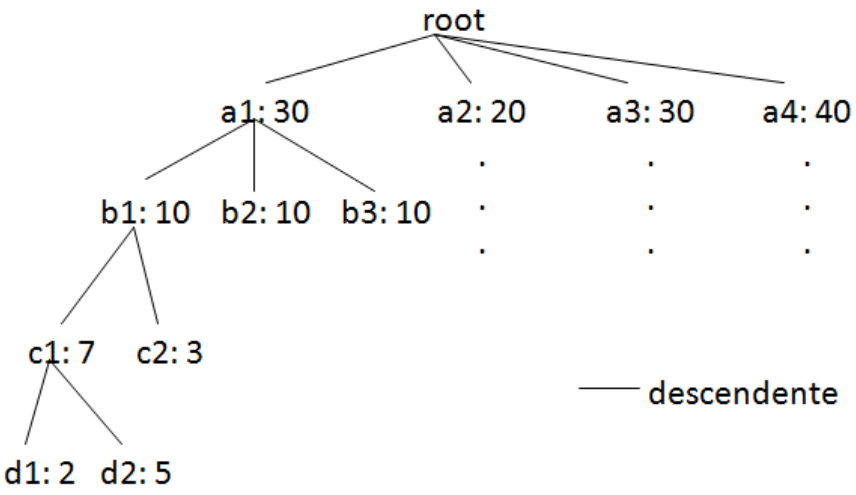

Figura 3.2: Um Fragmento de um Grafo de Cubo Base

A abordagem MCG reduz o tamanho de um cubo de dados através da fusão de grafos que possuem valores da função *graph-path* idênticos. A função *graph-path* gera um identificador único para um sub-grafo, possibilitando assim que sub-grafos sejam fundidos

e o consumo de memória seja reduzido. Infelizmente, o cálculo do *graph-path* assume a criação do sub-grafo redundante, o que acarreta computação desnecessária. Outra desvantagem de gerar o *graph-path* ocorre na atualização de cubos MCG. Novas tuplas podem exigir a separação de sub-grafos anteriormente fundidos. Este procedimento é extremamente custoso computacionalmente.

Além da função *graph-path*, a abordagem MCG traz a inovação do método para geração das sumarizações sob demanda. Na Figura 3.3 é possível visualizar um exemplo do funcionamento do método de geração de agregações sob demanda. Existem quatro grafos iniciados pelo nó *root*. O primeiro grafo representa um cubo base, ou em outras palavras, a relação de entrada sem redundância de prefixos. Os grafos restantes representam passo a passo o processo de geração de agregações MCG e os cubos resultantes. Inicialmente, a agregação (Palio, ALL, Prata) é gerada apontando diretamente Palio a Prata. O resultado é ilustrado no segundo grafo da Figura 3.3. O terceiro e quarto grafos ilustram o conflito quando uma segunda tupla (Palio, ALL, Prata) é encontrada, quando o caminho root->Palio->2010->Prata é percorrido. O conflito implica em um segundo nó com o rótulo Prata no quarto grafo da Figura 3.3. Note que, uma associação entre Palio->Prata:230 foi substituída por Palio->Prata:330. O último grafo representa o cubo completo gerado pelo MCG sem o problema da redundância de sufixos. Nenhum nó novo foi necessário para a geração das agregações remanescentes depois da criação do nó Prata com medida 330. De forma resumida novos nós só são criados caso existam conflitos de pais diferentes, caso contrário é adicionada uma referência ao nós existentes, de maneira a reaproveitá-los, diminuindo assim o consumo de memória. No exemplo da Figura 3.3, o cubo completo possui somente 7 nós. Caso não houvesse a eliminação dos sub-grafos redundantes o cubo completo possuiria 12 nós.

Na Figura 3.4 é ilustrado como seria o cubo completo sem o uso de agregação sob demanda. Como pode ser observado na Figura 3.4, existe uma redundância de nós intermediários que possuem informação idêntica, como os nós 2009, 2010 e Prata, filhos de Palio, e os nós 2009, 2010 e Prata, filhos de *root*. Neste cenário, um cubo completo possui 12 nós, sendo 5 nós a mais do que o cubo completo ilustrado na Figura 3.3. Em ambas as figuras foram utilizadas a mesma entrada de dados, o que significa que, com o uso da técnica de geração de agregações sob demanda, obtivemos uma redução de $\approx$58% no tamanho do cubo completo.

De uma forma geral, a abordagem MCG consome 70-90% de memória se comparada a abordagem Star-cubing original, descrita em (Xin, Han, Li & Wah 2003). No mesmo cenário, a abordagem Star-cubing proposta em (Xin, Han, Li, Shao & Wah 2007), reduz o

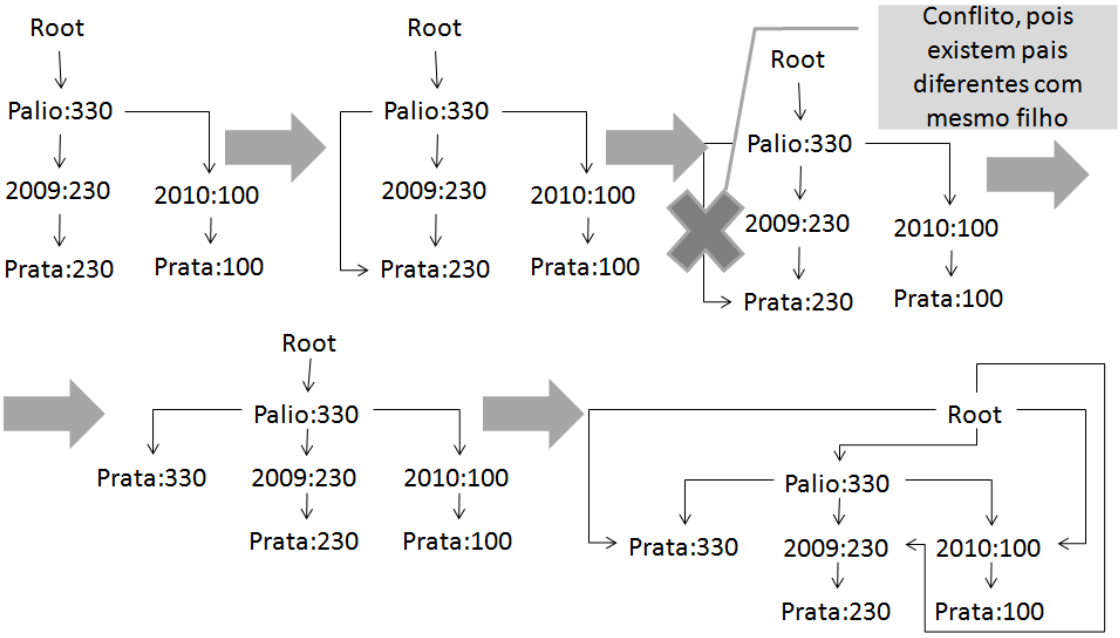

Figura 3.3: Exemplo da Geração de um Cubo Completo com o uso de Agregação Sob Demanda

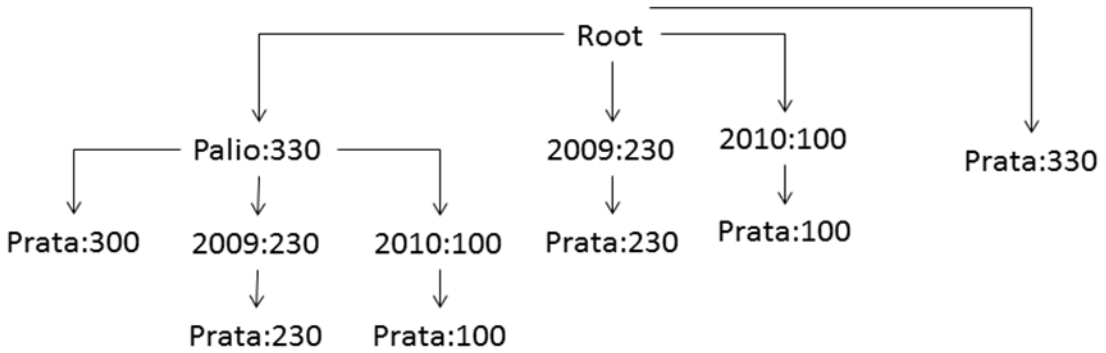

Figura 3.4: Exemplo da Geração de um Cubo Completo sem o uso de Agregação Sob Demanda

consumo de memória de somente 10-30%, Dwarf em 30-50% e MDAG em 40-60%, quando comparadas com a abordagem Star-cubing original. Cabe ressaltar que a abordagem MCG é, em média, 20-40% mais rápida que as abordagens Dwarf, MDAG e Star no cálculo de cubos esparsos.

Capítulo 4

A Abordagem P2CDM

A abordagem P2CDM foi implementada utilizando a linguagem Java, o mecanismo de comunicação utilizado para a comunicação entre os nós de processamento é o *Socket*, no modo orientado a conexão, o que significa que é utilizado o protocolo TCP/IP. Neste trabalho, a seguinte nomenclatura é considerada: um conjunto de árvores é chamado de floresta e representa um cubo completo e cada uma das árvores pertencentes à floresta representa um subcubo.

A abordagem P2CDM computa um cubo completo em quatro passos e com uma única operação de varredura na base de dados R. Inicialmente, é gerado um mapa de prefixos únicos, indicando onde os atributos da relação R devem ser computados. Os atributos de cada dimensão podem ser facilmente mapeados a nós de processamento usando estratégias diversas, dentre elas round-robin ou roleta. Na Figura 4.1, ilustra-se o mapa de prefixos únicos e seus respectivos endereços, que indicam onde devem ser computados.

Após o primeiro passo, são criadas d relações a partir da relação R' recebida, sendo R' uma relação derivada de R que possui $\frac{n}{p}$ linhas, onde n é o número de linhas da relação R e p o número de nós de processamento utilizados para a computação do cubo. Ainda no segundo passo, é necessária a fragmentação das d relações em k outras relações, onde k é o número de prefixos distintos de cada relação pertencente a d. Na Figura 4.1 ilustra-se o resultado do primeiro passo, que é um conjunto de tabelas com todos os prefixos distintos de cada dimensão, utilizado por cada nó de processamento para computar o cubo.

Uma vez finalizado o segundo passo, inicia-se o envio e recebimento de um conjunto

de relações com os prefixos únicos com base no mapa de endereços criado no primeiro passo. Por fim, no quarto passo computa-se o subcubo completo ou parcial recebido, com base nas relações recebidas no passo anterior. Na Figura 4.1, após a última seta, é possível visualizar os passos três e quatro combinados em uma única representação, isso porque a quantidade de processadores considerada foi igual a um. Caso fosse superior a um, os passos seriam separados. No restante desta seção, são descritos em detalhes os algoritmos que compõem a abordagem P2CDM.

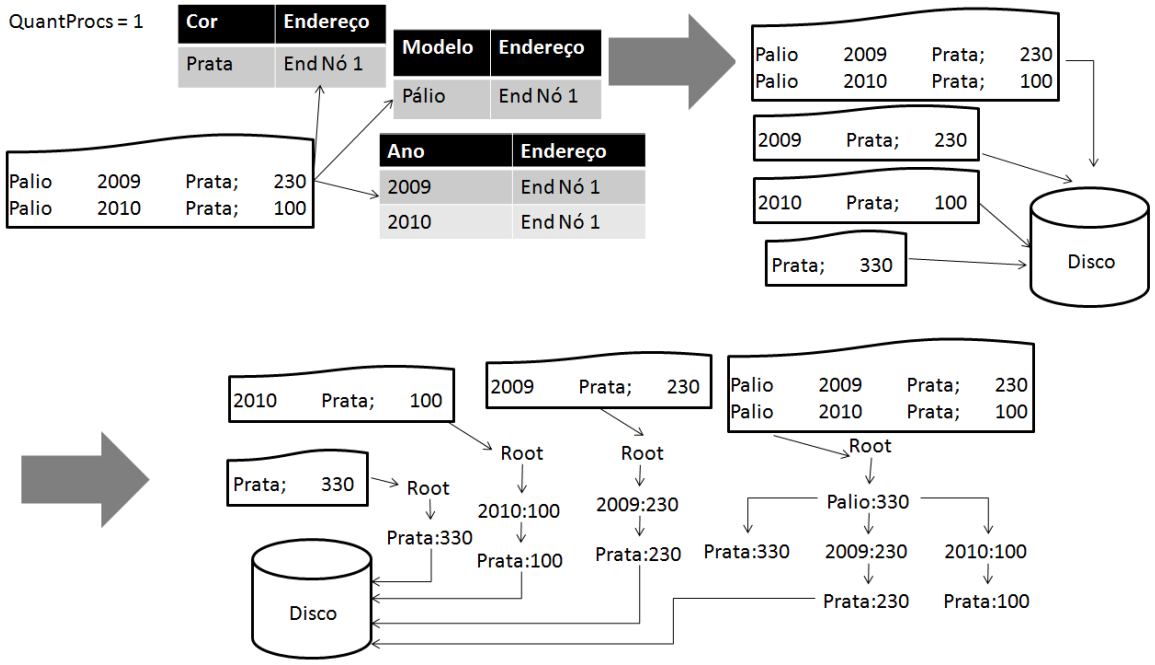

Figura 4.1: Exemplo do Funcionamento da Abordagem P2CDM para 1 nó de Processamento

4.1 O Algoritmo P2CDM

O primeiro algoritmo assume como entrada de dados um arquivo representando uma relação d-dimensional R, o qual consiste de n linhas, que aqui são denotadas por R[i], onde i $\in$ N e i $\subset$ [1,n]. Também é assumido como entrada de dados um arquivo contendo um conjunto de atributos únicos de cada dimensão, que é utilizado de forma a auxiliar na criação de d relações de atributos únicos. Os atributos são os prefixos das tuplas sem repetição e d é a quantidade de dimensões do cubo a ser computado.

Além das entradas de dados anteriores, tem-se o número máximo de tuplas que pode ser materializado em memória para a geração do subcubo completo, e um limiar que indica a partir de quando uma distribuição de frequência de um atributo é considerada *skewed*. Ambas as entradas são predefinidas pelo usuário, e são usadas para controlar o consumo de memória e/ou CPU. Note que seria possível utilizar outros critérios para expressar *skew*, como o número máximo de agregações que podem ser geradas por um determinado atributo, tempo de execução para gerar agregações a partir de um determinado atributo no grafo P2CDM, entre outros.

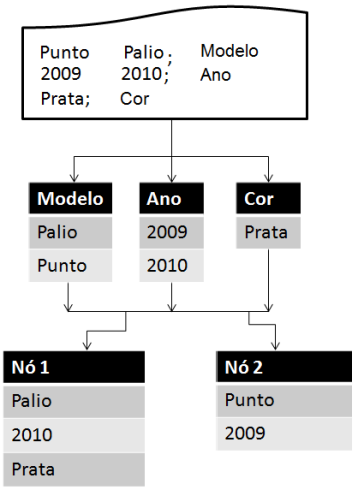

Figura 4.2: Exemplo de Geração das p Tabelas de Prefixos Únicos

Após a geração das d relações de prefixos únicos, cada uma delas é dividida em p outras relações, sendo p o número de núcleos de processamento disponíveis para a computação do cubo. Em conjunto com a criação destas p relações é criado um mapa de prefixos únicos com seus respectivos endereços. Nas linhas 1 a 9 do Algoritmo 1 é descrito este passo. Este esquema de particionamento pelo prefixo da dimensão é utilizado para auxiliar no balanceamento de carga, como ilustrado na Figura 4.2. Nesse exemplo foi considerada a existência de dois nós de processamento, além de ter sido adicionada a tupla (Carro=Punto; Ano=2009; Cor=Prata; Vendas=120), para melhor ilustração desta fase. Essa tupla é considerada nos demais exemplos desta seção.

No exemplo da Figura 4.2, as relações foram carregadas para as tabelas correspondentes aos nomes dos atributos da relação original. Os valores de atributos Palio e Punto formam a tabela modelo de carros, 2009 e 2010 a tabela ano de fabricação e assim por diante. Após a criação de cada uma destas tabelas, seus atributos são distribuídos para os mapas de prefixos únicos de cada nó de processamento. Ao fim dessa distribuição,

Algoritmo 1 Algoritmo *P2CDM_master*

Entrada: Uma base de dados bruta R, com n linhas (R[i..n], onde i∈ N e i ⊂ [1,n]); $T_{slaveAdress}$ uma tabela com a localização dos escravos; up é um conjunto com todos os prefixos únicos de cada dimensão; p é o número de nós de processamento que será utilizado; *threshold* indica a quantidade máxima de tuplas será materializada em memória; x é um limiar para verificação de atributos *skewed*.

Saida: O cubo de dados completo distribuído, e persistido entre os p nós de processamento escravo.

1: **for** $i = 1 \rightarrow d$ **do**
2: $aux \leftarrow up[i].poll()$
3: $ps_i \leftarrow aux$
4: **if** $aux \notin mp$ **then**
5: mp.put<aux, $T_{slaveAdress}$[j]>
6: $j \leftarrow j + 1$
7: $j \leftarrow (j > p)?1 : j$
8: **end if**
9: **end for**
10: **for** $i = 1 \rightarrow p$ **do**
11: Call $P2CDM\_Slave(R'_i, ps_i$, mp, threshold, x); onde R'_i é uma base derivada de R que possui $\frac{n}{p}$ linhas; ps_i é um conjunto com os prefixos específicos daquele processador, mp é um mapa com todos os prefixos e a localização respectiva onde deve ser computado.
12: **end for**

são geradas p tabelas de prefixos únicos.

Na linha 11 do Algoritmo 1, é chamado o método $P2CDM\_Slave$ para que cada nó de processamento receba uma partição equivalente da base de dados bruta R, que possui $\frac{n}{p}$ linhas, onde n é a quantidade de linhas da base R e p é a quantidade de nós de processamento disponíveis. Além dessa entrada, cada nó recebe um limiar que indica o limite para um atributo ser considerado *skewed*. Devido a esse limiar, conseguimos tratar os dados com distribuição não uniforme de forma diferente dos dados com distribuição uniforme. Este tratamento diferenciado é muito importante para a garantia de um correto balanceamento das cargas de trabalho, e no auxílio da diminuição do tamanho das mensagens trafegadas na rede. Isso porque os atributos *skeweds* iguais são agrupados e re-divididos em p porções, e cada uma destas porções é colocada no conjunto de envio.

Graças à estratégia de geração das p tabelas de prefixo único, é possível garantir a geração de florestas de subcubos completos sem dados redundantes. Cada nó também recebe um conjunto que contém os prefixos que devem ser computados por ele, a quantidade de tuplas que torna possível a materialização do subcubo completo na memória

principal, e um mapa com todos os prefixos e seus respectivos endereços. Os endereços indicam onde os prefixos devem ser computados para a geração da floresta de subcubos completos.

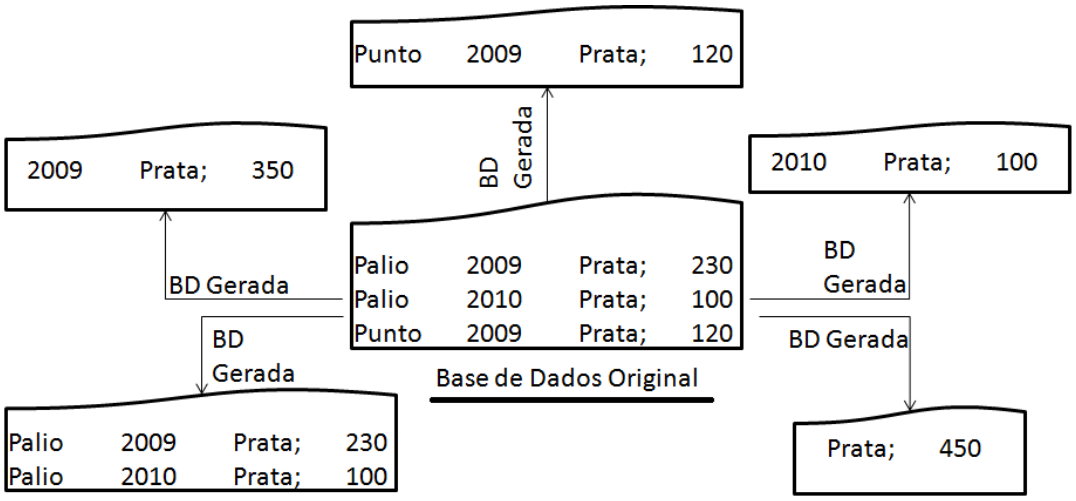

Figura 4.3: Exemplo de Geração das k bases a partir de uma Porção da Base de Dados Recebida

A geração da floresta de subcubos completos é realizada em 3 passos, sendo o primeiro descrito nas linhas de 1 a 4 do Algoritmo 2. Neste trecho de código, são criadas d relações a partir da relação de entrada com as tuplas iguais fundidas. Nesse passo é utilizado o algoritmo de ordenação externa da API *SmallText* [1]. Cada uma destas d bases é dividida em k outras bases, onde k é o número de prefixos distintos presentes no conjunto de d bases, como ilustrado na Figura 4.3. No exemplo, a base de dados original é fragmentada em 5 base de dados, correspondentes ao número de prefixos únicos encontrados na base de dados original que são Palio, Punto, 2009, 2010 e Prata, com seus respectivos valores de medida acumulados.

Após a geração das k relações, como apresentado no Algoritmo 2 nas linhas 5 a 8, cada uma das relações é agrupada de acordo com o nó destino. Este agrupamento é obtido utilizando o mapa mp de prefixo e localização. Após o agrupamento, o conjunto é compactado e enviado ao endereço correspondente ao nó destino de cada grupo, com uma ressalva: as relações de prefixo não pertencentes ao conjunto de prefixos de dimensão *skewed* são enviadas de forma imediata para serem adicionadas ao conjunto de envio. Isso porque as relações de prefixos *skewed*, no momento que são detectadas através do limiar x, são divididas em p outras relações. Cada uma das p relações é adicionada a

[1] http://zola.di.unipi.it/smalltext/

Algoritmo 2 Algoritmo *P2CDM_slave*

Entrada: R'_i: Uma base relacional derivada de R armazenada em disco, com $\frac{n}{p}$ linhas; um conjunto com os prefixos específicos daquele processador (ps_i); um mapa com todos os prefixos e a localização respectiva onde deve ser computado (mp), a quantidade máxima de tuplas será materializada em memória (*threshold*), x o limiar máximo que indica se um atributo pertence ao conjunto *skewed* ou não.

Saida: O subcubo completo armazenado em disco.

//Cria os lotes de bases de mesmo prefixo

1: **for** $j = 1 \rightarrow d$ **do**

2: Cria T_i^{j+1} de T_i^j via ordenação sequencial em disco, com junção de tuplas iguais. (T_i^1)

3: particiona a base T_i^j em k outras base de dados que terá apenas tuplas com o mesmo prefixo.

4: **end for**

//Troca de conjunto de bases particionados

5: **for all** para cada base de dados gerada no passo anterior **do**

6: envia ela ao nó destino, se o prefixo não pertence ao conjunto de prefixos *skewed*

7: se o prefixo pertence ao conjunto de prefixos *skewed*, particiona ela em p outras bases e envia cada uma delas a um nó de processamento.

8: **end for**

//Computação do subcubo completo

9: **for all** para cada conjunto de base de dados de mesmo prefixo **do**

10: enquanto cada base é lida do disco, crie preencha uma árvore de subcubo completo até que se atinja o *threshold*

11: caso o *threshold* tenha sido atingido, gere todas as agregações sob demanda do subcubo completo, e em seguida persista o subcubo, crie uma nova árvore

12: **end for**

um conjunto que será enviado a um nó de processamento distinto, garantindo assim um equilíbrio das cargas.

Por fim, a geração dos subcubos completos em relação a cada prefixo, pode ser visualizada nas linhas de 9 a 12 do Algoritmo 2. Neste passo é utilizado o benefício da criação de nós sob demanda, como mostrado na Figura 4.4. Cada um dos subcubos é serializado e persistido em disco utilizando o *framework* Kryo [2]. A persistência dos subcubos só ocorre quando se atinge a máxima quantidade de tuplas permitidas para serem materializadas em memória principal, ou quando a árvore de subcubo completo ainda não tiver sido persistida. Este processo é repetido até que todas as relações sejam processadas. Ao fim deste processo, cada nó de processamento possui uma floresta de subcubos completos, sem redundâncias, persistida em disco.

[2]http://code.google.com/p/kryo/

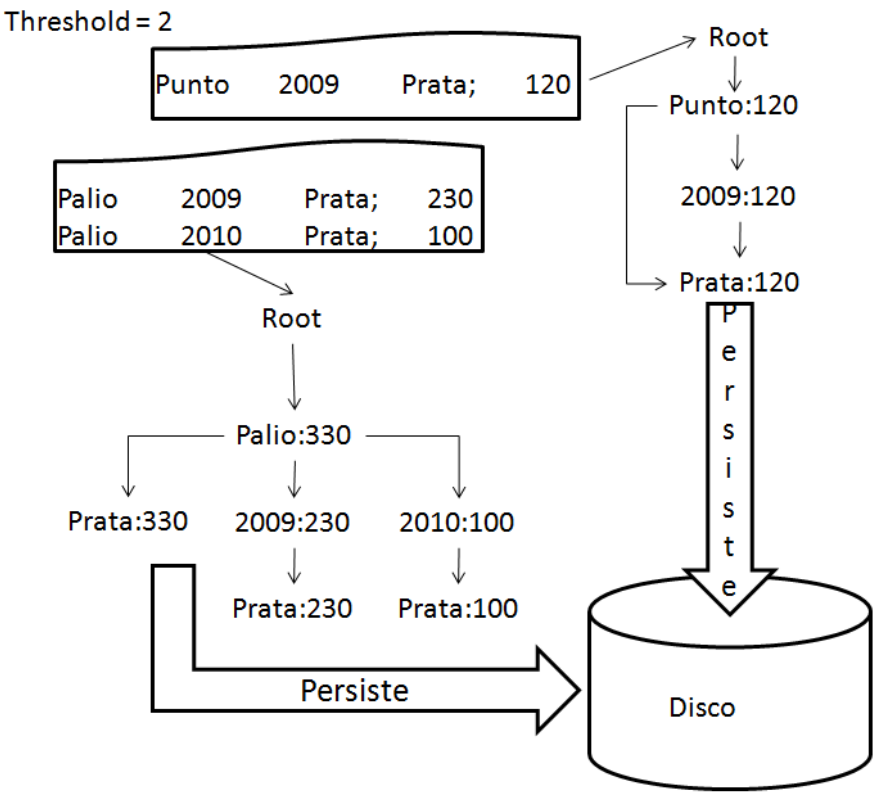

Figura 4.4: Exemplo de Geração de árvore de Subcubo Completo

Independentemente da distribuição dos dados serem *skewed* ou não, a P2CDM possui comunicação minimizada. Isso porque, cada nó de processamento só se comunica uma vez com cada nó de processamento do conjunto de processadores participantes da computação do cubo, ao contrário da PnP, que se comunica d vezes mais que a P2CDM. Esta otimização possui fundamental importância para a P2CDM, uma vez que implica na diminuição do tempo de computação do cubo de dados.

O cenário ótimo de geração de cubos completos com ausência de dados redundantes ocorre quando a distribuição de frequência dos dados é homogênea e o universo de tuplas de um mesmo prefixo cabe na memória principal. Dessa maneira, cada nó de processamento possui uma floresta de subcubos completos disjuntos, sendo que a união de todas elas formam um cubo completo sem redundância de dados, como ilustra a Figura 4.4. No exemplo, é considerado como número máximo de tuplas que podem ser materializadas em memória, igual a 2. Isso possibilita a criação de árvores de subcubos completos disjuntas e sem redundância de dados, sendo que em nenhuma das bases de entrada o limiar sejma ultrapassado.

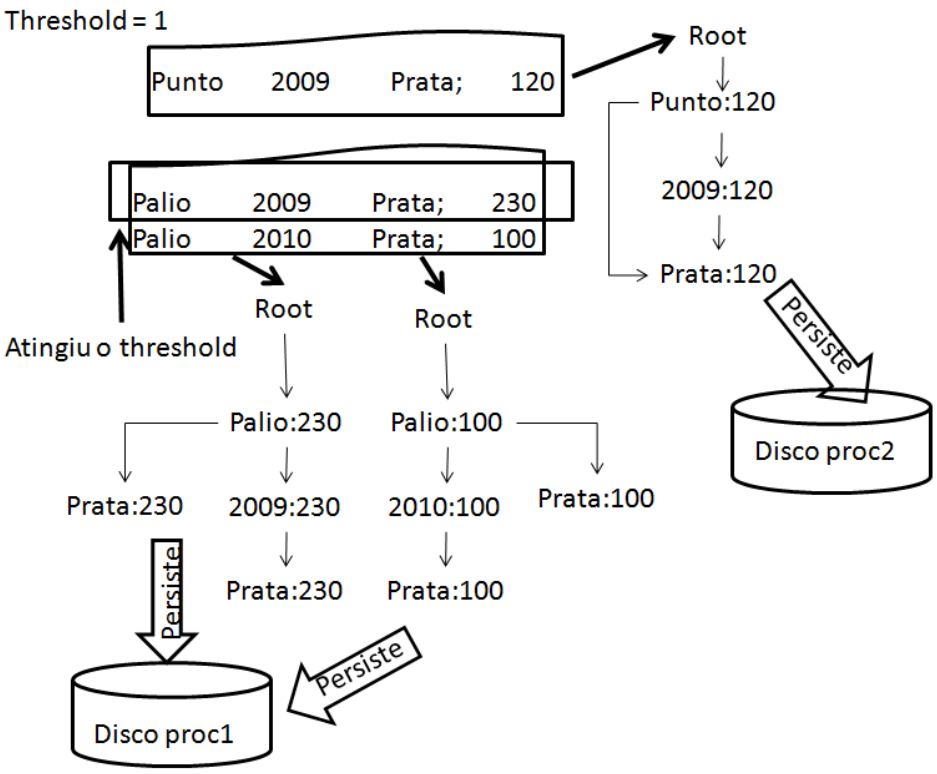

Figura 4.5: Exemplo de Geração de árvore de Subcubo Completo com Dados Redundantes

No entanto, o pior cenário ocorre quando o conjunto de dados de cada prefixo não cabe na memória e a distribuição de frequência é heterogênea. Desta forma, é necessária a geração de agregações redundantes à medida que cada nó de processamento esgota sua capacidade de armazenamento para um determinado prefixo, como ilustrado na Figura 4.5. No entanto, mesmo nos cenários onde ocorre aumento de *swaps* em disco e ao aumento no particionamento das relações, o tempo de execução e o consumo de memória da abordagem P2CDM continuam menores que os da PnP, conforme será mostrado no Capítulo 5. Isto se deve ao método de geração de agregações sob demanda e a menor quantidade de comunicações entre os nós de processamento.

Capítulo 5

Avaliação de Desempenho

Um estudo abrangente do desempenho foi realizado para verificar a eficiência e a escalabilidade do algoritmo proposto. Testamos o algoritmo P2CDM contra nossa implementação para o algoritmo PnP. Seguimos o algoritmo descrito no artigo (Chen, Dehne, Eavis & Rau-Chaplin 2008) para implementar uma versão em Java. Todos os algoritmos foram codificados em Java 32 bits (JRE 6.0 *update* 30).

Os testes foram realizados em um *cluster* homogêneo com 32 nós de processamento, cada uma com o P2CDM hospedado e com as configurações descritas na Tabela 5.1. Devido a restrições do sistema operacional, os programas P2CDM e PnP foram executadas com somente 1.5GB de memória primária para cada máquina do *cluster*.

Característica	*Descrição*
Processador	AMD Athlon(tm) Turion X2 5400B
Clock	2,81 GHz
Memória Primária	2GB
Memória Secundária	60GB
Sistema Operacional	Windows XP Professional versão 2002 com Service Pack 3

Tabela 5.1: Ambiente de Teste

Todos os tempos registrados incluem computação e I/O, é importante frisar que cada teste foi executado dez vezes e os tempos apresentados neste capítulo foi fruto do cálculo do desvio padrão dos valores coletados durante os experimentos. As relações utilizadas

utilizadas nos testes eram sintéticas, criadas através de um gerador de base de dados fornecida pelo projeto IlliMine [1]. Nesta seção é utilizada a seguinte nomenclatura: T é o número de tuplas da base de dados R, S é o skew da relação, D é o número de dimensões, C é a cardinalidade de cada dimensão e N é o número de nós de processamento utilizado. Cada um destes parâmetros é utilizado para ver como é o desempenho do algoritmo em cenários com uma grande quantidade de dados, sendo estes esparsos.

Foi implementada a versão da PnP que utiliza memória secundária, para que os experimentos pudessem ter base de dados maiores. No entanto, a medida que o consumo de memória principal aumenta, os tempos de execução do algoritmo PnP começam a aumentar de forma drástica. Não foram efetuados testes com mais de 12 dimensões, uma vez que os cubos gerados pelo PnP excedem a capacidade do *cluster* quando algumas máquinas são utilizadas. Não foram utilizadas mais do que 9 dimensões quando uma única máquina é utilizada, uma vez que o consumo de memória excedia os 1.5GB de RAM e os 60GB de disco.

Ambas as abordagens foram testadas utilizando a medida média (avg), e na Figura 5.1 é possível visualizar os tempos de pesquisa nos cubos gerados por ambas as abordagens. Neste teste foi pesquisada a média do total de todas as dimensões do cubo. Excluindo somente o valor ALL, foram submetidas 100 consultas do tipo (A, *, *, *, *, *, *, *, *, *, *), (*, B, *, *, *,*, *, *, *, *, *), (*, *, C, *, *, *, *, *, *, *, *), ..., (*, *, *,*, *, *, *, *, *, J), totalizando 1000 consultas. No exemplo A=[a1...a100], B=[b1...b100], etc. Foram calculados o tempo de execução, a aceleração e o consumo de memória. Todos os parâmetros utilizados nos experimentos (dimensão, número de tuplas, cardinalidade, etc.) foram avaliadas de maneira individual. Figura 5.2 ilustra a variação da cardinalidade. Figura 5.3 ilustra a variação do *skew*. Figura 5.4 ilustra a variação do número de tuplas e, finalmente, a Figura 5.5 ilustra a variação do número de dimensões.

As abordagens PnP e P2CDM possuem comportamento similar quando o número de tuplas aumenta. Ambas as abordagens possuem um bom desempenho quando a cardinalidade é aumentada. O problema da dimensionalidade que existe em cubo de dados fica claro na Figura 5.5, uma vez que, conforme o número de dimensões aumenta linearmente, o tempo de execução aumenta exponencialmente. O tempo de execução de ambas as abordagens não muda de maneira significativa quando se aumenta o *skew*.

[1] O um projeto IlliMine é um projeto de código livre, o qual fornece uma grande quantidade de abordagens para mineração de base de dados e aprendizado de máquina, que pode ser encontrado no site *http* : *//illimine.cs.uiuc.edu/*.

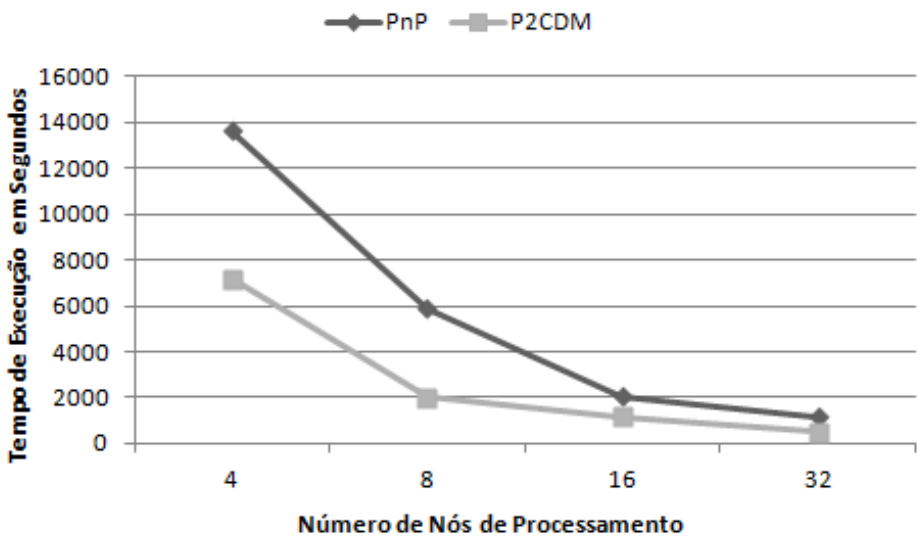

Figura 5.1: T = 10M, D = 10, C = 100, S = 0

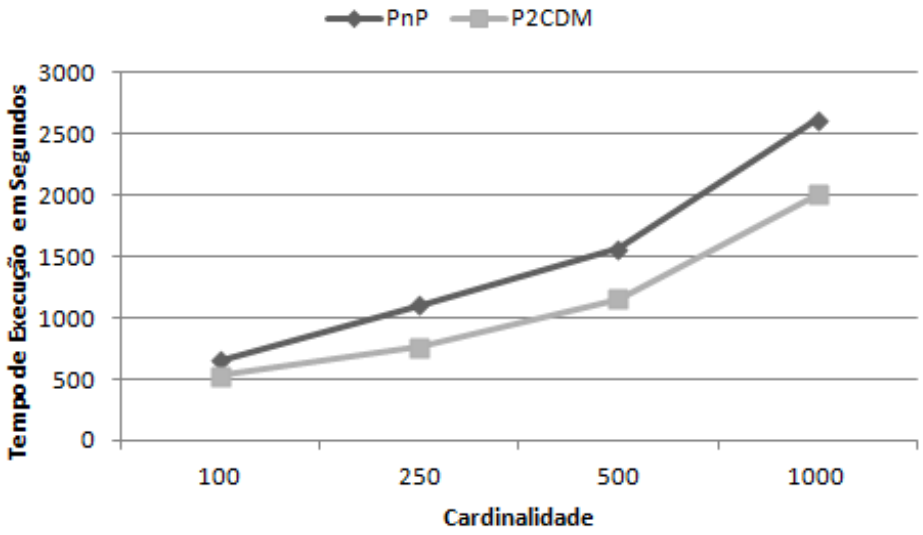

Figura 5.2: T = 10M, D = 10, S = 0, N = 32

A abordagem P2CDM possui desempenho similar a PnP, fornecendo uma aceleração próxima do linear para um grande número de máquinas com uma grande entrada de dados, como pode ser visualizado na Figura 5.7. No entanto, diferentemente da PnP, a P2CDM possui um baixo consumo de memória, mesmo nos cenários onde o volume de dados de entrada cresce, como pode ser visualizado na Figura 5.6. A P2CDM é cerca de 20-25% mais rápida mesmo em cenários onde cada processador possui uma pequena quantidade de dado para se computar.

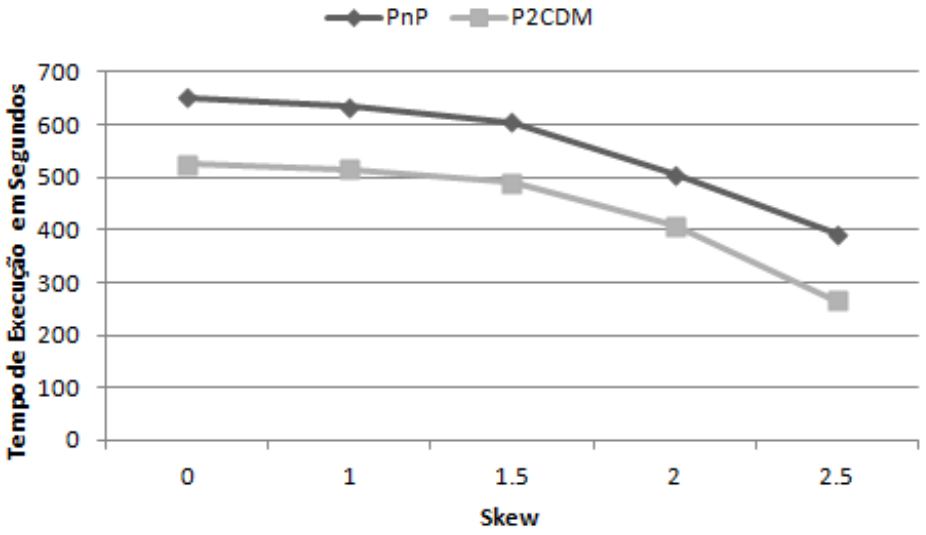

Figura 5.3: T = 10M, C = 100, D = 10, N = 32

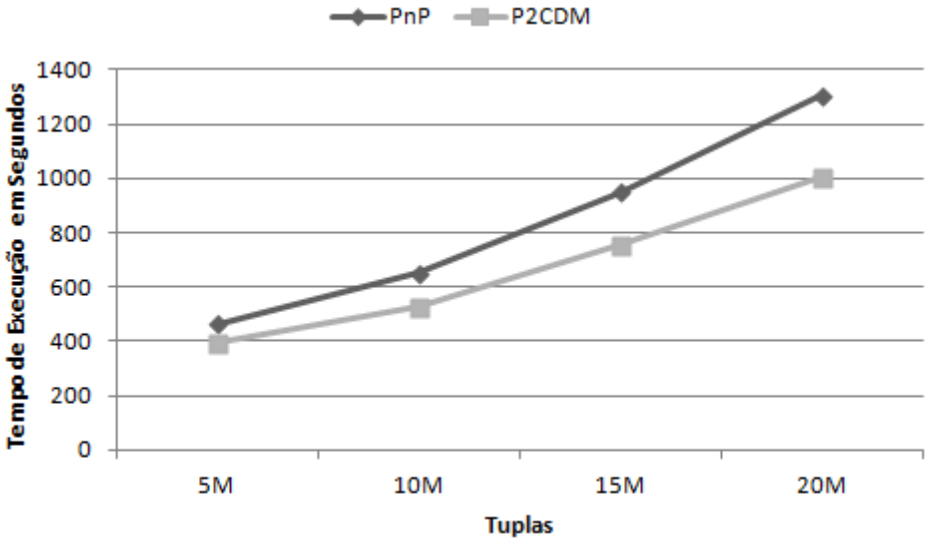

Figura 5.4: D = 10, C = 100, S = 0, N = 32

De uma forma geral, os resultados obtidos com a abordagem P2CDM podem ser explicados pela consideração do *skew* na partição dos dados, que influenciou a minimização da quantidade de comunicação entre os nós de processamento. Outra característica positiva da abordagem P2CDM é a integração com a abordagem MCG, que auxilia na computação de cubos maiores em ambientes de baixo custo, que no geral possuem capacidade de armazenamento em disco reduzida. A abordagem MCG reduz consideravelmente o tamanho do cubo de dados, o que implicou em uma redução de 30-

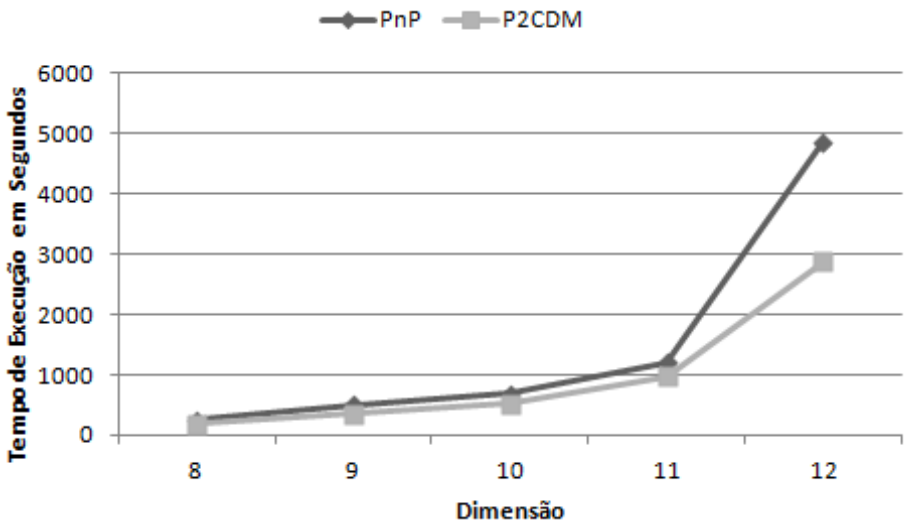

Figura 5.5: T = 10M, C = 100, S = 0, N = 32

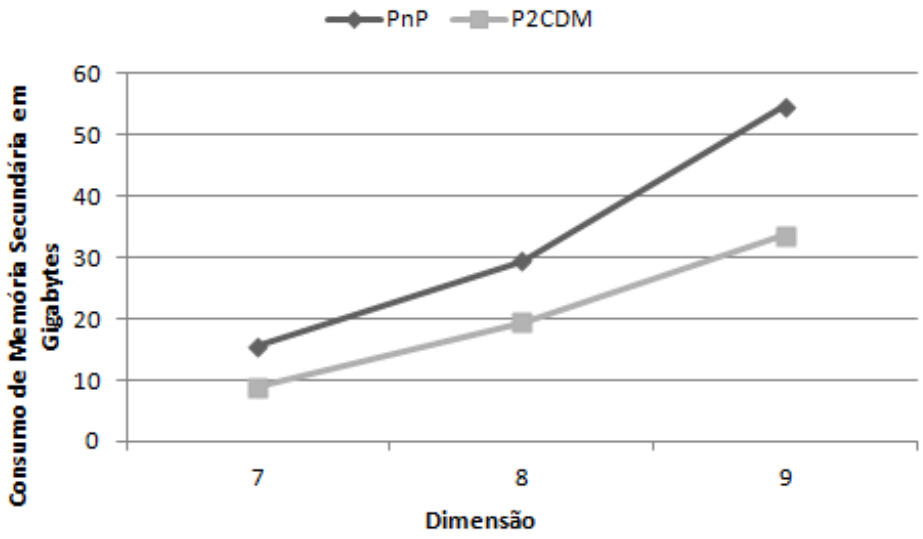

Figura 5.6: T = 10M, C = 100, S = 0, N = 1

40% do tamanho do cubo se comparado com os resultados gerados pela PnP, permitindo a P2CDM computar cubos maiores que a PnP quando a capacidade de armazenamento de cada nó é reduzida. Neste sentido, foi possível a computação do cubo de dados com 9 dimensões, cardinalidade 100, 200M de tuplas e *skew* igual a zero, em 2h 45min 54s, com uma saída de 27,6GB em cada um dos nós de processamento, usando a medida média. A relação de entrada possuía em média 1.7GB em cada nó de processamento.

No geral, a estratégia de geração de dados redundantes da P2CDM não influencia

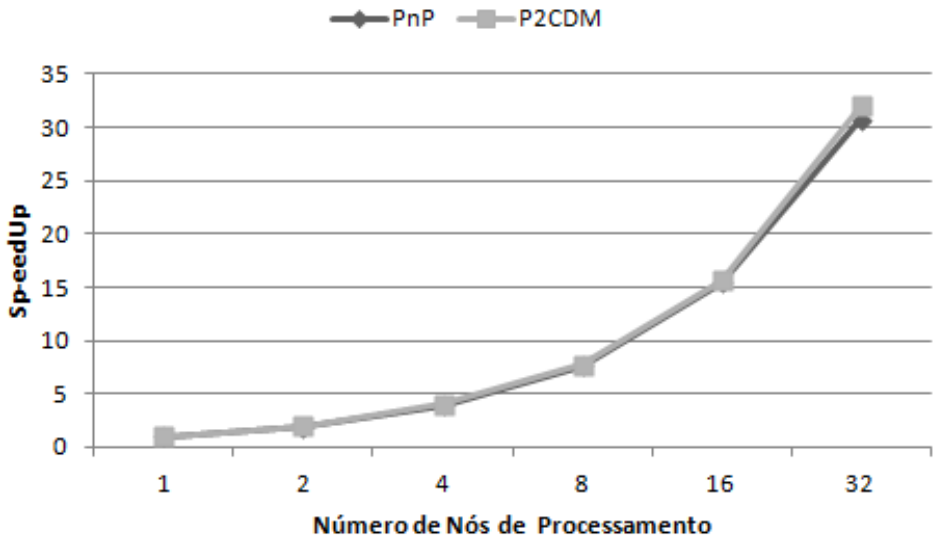

Figura 5.7: T = 10M, D = 8, C = 100, S = 0

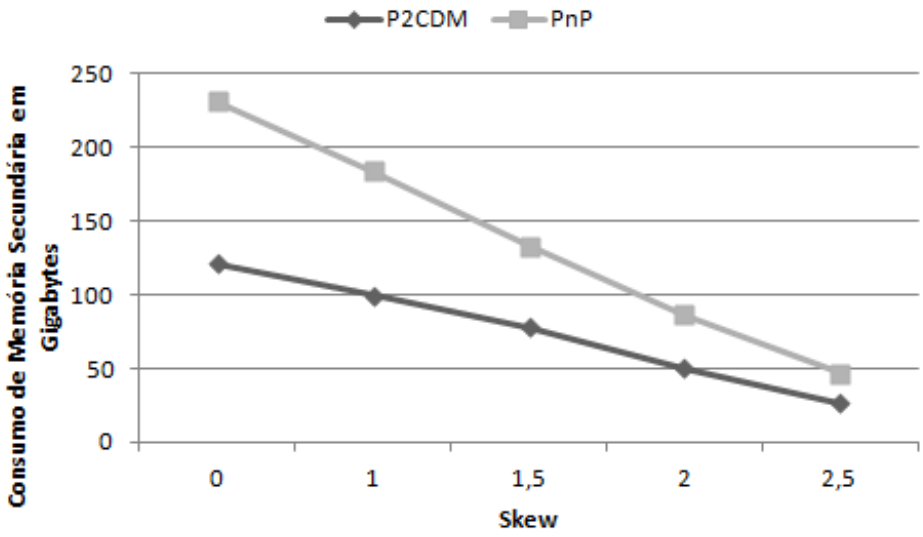

Figura 5.8: T = 20M, C = 100, D = 10, N = 32

negativamente no tempo de execução, além de permitir a computação de cubo de dados completos. A P2CDM produz dados redundantes de maneira similar a PnP somente em cenários nos quais todos os atributos de uma relação são *skewed*, conforme pode ser visualizado na Figura 5.8. Conforme o *skew* aumenta o gasto de memória de ambas as abordagens tendem a ficar similares. Geralmente, tais cenários não são encontrados em DWs reais.

Capítulo 6

Conclusão e Trabalhos Futuros

Neste trabalho, foi apresentada a abordagem P2CDM para a representação e computação de cubos de dados em sistemas com arquitetura de memória distribuída, possuindo consumo de memória reduzido e comunicação minimizada entre os nós de processamento.

O método de geração de agregações sob demanda da abordagem MCG, incorporado pela P2CDM, consome menos memória e é mais rápido que o método Pipe 'n Prune utilizado pela abordagem PnP. A estratégia de geração de redundância de dados sob demanda adotada pela P2CDM demonstrou ainda que é possível gerar subcubos completos sem redundância de dados entre os nós de processamento do *cluster*.

Somente em alguns cenários *skewed* a abordagem P2CDM gera dados redundantes. Já a abordagem PnP gera máxima redundância de dados até em cenários com distribuição uniforme de dados. Foi assumido que a redundância de dados é um problema no tempo de consulta no cubo de dados, uma vez que cada resultado da consulta deve ser fundido depois que forem obtidos todos os resultados parciais do *cluster*.

Como trabalho futuro, os autores propõem o uso combinado do paralelismo com memória compartilhada, incluindo GPUs, com memória distribuída, como forma de melhorar a escalabilidade do algoritmo. É interessante a comparação com a abordagem Brown Dwarf, assim como as abordagens baseadas no paradigma MapReduceMerge ((You, Xi, Zhang & Chen 2008), (Sergey & Yury 2009), (Wang, Song & Luo 2010) e (Nandi, Yu, Bohannon & Ramakrishnan 2011)). Consultas pontuais, atualizações e o desenvolvimento de uma metodologia otimizada para o cálculo de medidas holísticas também são necessárias, uma vez que isso irá contribuir para reforçar as ideias propostas pela abordagem P2CDM. Uma vez disponível o mapa de localização é possível, por

exemplo, calcular a moda local, pois com o mapa de atributos é possível garantir que somente um nó da rede conterá determinado atributo, sendo assim se a moda local for calculada, o nó mestre pode receber os resultados e gerar a moda final, ou a moda aproximada, no entanto tais propostas devem ser testadas.

O casamento exato de sub-grafos da MCG é uma ideia que deve ser implementada na P2CDM, uma vez que ela prova que o consumo de memória pode ser ainda mais reduzido, assim como descrito por (Lima & Hirata 2011).Experimentos com atualização são particularmente importantes com a P2CDM sem os sub-grafos redundantes, uma vez que a fusão dos sub-grafos na MCG e a de-fusão tem um grande impacto, possivelmente gerando um gargalo com o aumento das atualizações.

O problema da dimensionalidade presente na P2CDM pode ser resolvida pelo uso de cubos parciais baseadas em índices invertidos, como *Frag-Cubing* (Li, Han & Gonzalez 2004). A limitação do número de tuplas imposta pela *Frag-Cubing* pode ser eliminada com o uso do método de particionamento de dados implementado pela P2CDM.

Referências Bibliográficas

Beyer, K. & Ramakrishnan, R. (1999a). Bottom-up computation of sparse and iceberg cube, *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ACM, New York, NY, USA, pp. 359–370.

Beyer, K. & Ramakrishnan, R. (1999b). Bottom-up computation of sparse and iceberg cube, *SIGMOD Records* **28**: 359–370.

Chen, Y., Dehne, F., Eavis, T. & Rau-Chaplin, A. (2008). Pnp: Sequential, external memory, and parallel iceberg cube computation, *Distributed Parallel Databases* **23**(2): 99–126.

Codd, E. F., Codd, S. B. & Salley, C. T. (1993). Providing OLAP to User-Analysts: An IT Mandate.

DeWitt, D. & Gray, J. (1992). Parallel database systems: The future of high performance database systems, *Communications of the ACM* **35**(6): 85–98.

Doka, K., Tsoumakos, D. & Koziris, N. (2011). Brown dwarf: A fully-distributed, fault-tolerant data warehousing system, *Journal of Parallel and Distributed Computing* **71**: 1434–1446.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. & Pirahesh, H. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals, *Data Mining and Knowledge Discovery* **1**(1): 29–53.

Han, J., Kamber, M. & Pei, J. (2006). *Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*, 2 edn, Morgan Kaufmann.

Inmon, W. H. & Hackathorn, R. D. (1994). *Using the Data Warehouse*, Wiley-QED Publishing, Somerset, NJ, USA.

Kaczmarski, K. (2011). Comparing gpu and cpu in olap cubes creation, *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM'11, Springer-Verlag, Berlin, Heidelberg, pp. 308–319.

Lakshmanan, L. V. S., Pei, J., U, S. F. & Han, J. (2002). Quotient cube: How to summarize the semantics of a data cube, pp. 778–789.

Lauer, T., Datta, A., Khadikov, Z. & Anselm, C. (2010). Exploring graphics processing units as parallel coprocessors for online aggregation, *Proceedings of the ACM 13th International Workshop on Data Warehousing and OLAP*, DOLAP '10, ACM, New York, NY, USA, pp. 77–84.

Li, X., Han, J. & Gonzalez, H. (2004). High-dimensional olap: A minimal cubing approach, *Proceedings of 2004 International Conference on Very Large Data Bases (VLDB'04*, pp. 528–539.

Lima, J. d. C. & Hirata, C. M. (2007). Mdag-cubing: A reduced star-cubing approach, *Proceedings of the 22rd Brazilian Symposium on Databases*, SBBD '07, Sociedade Brasileira de Computacao, Joao Pessoa, Paraiba, Brazil, pp. 362–376.

Lima, J. d. C. & Hirata, C. M. (2011). Multidimensional cyclic graph approach: Representing a data cube without common sub-graphs, *Inf. Sci.* **181**: 2626–2655.

Nandi, A., Yu, C., Bohannon, P. & Ramakrishnan, R. (2011). Distributed cube materialization on holistic measures, *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, IEEE Computer Society, Washington, DC, USA, pp. 183–194.

Ng, R. T., Wagner, A. S. & Yin, Y. (2001). Iceberg-cube computation with pc clusters, *SIGMOD Conference*, pp. 25–36.

Sarawagi, S., Agrawal, R. & Gupta, A. (1996). On computing the data cube, *Technical report*, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

Sergey, K. & Yury, K. (2009). Applying map-reduce paradigm for parallel closed cube computation, *Proceedings of the 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*, IEEE Computer Society, Washington, DC, USA, pp. 62–67.

Sismanis, Y., Deligiannakis, A., Roussopoulos, N. & Kotidis, Y. (2002). Dwarf: Shrinking the petacube, *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, ACM, New York, NY, USA, pp. 464–475.

Wang, Y., Song, A. & Luo, J. (2010). A mapreducemerge-based data cube construction method, *Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing*, GCC '10, IEEE Computer Society, Washington, DC, USA, pp. 1–6.

Xin, D., Han, J., Li, X., Shao, Z. & Wah, B. W. (2007). Computing iceberg cubes by top-down and bottom-up integration: The starcubing approach, *IEEE Transactions on Knowledge and Data Engineering* **19**(1): 111–126.

Xin, D., Han, J., Li, X. & Wah, B. W. (2003). Star-cubing: Computing iceberg cubes by top-down and bottom-up integration, *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '2003, VLDB Endowment, pp. 476–487.

Xin, D., Shao, Z., Han, J. & Liu, H. (2006). C-cubing: Efficient computation of closed cubes by aggregation-based checking, *In ICDE'06*, IEEE Computer Society, p. 4.

You, J., Xi, J., Zhang, P. & Chen, H. (2008). A parallel algorithm for closed cube computation, *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, IEEE Computer Society, Washington, DC, USA, pp. 95–99.

Zhao, Y., Deshpande, P. M. & Naughton, J. F. (1997). An array-based algorithm for simultaneous multidimensional aggregates, *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, ACM, New York, NY, USA, pp. 159–170.