Tese apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica, como parte dos requisitos para obtenção do título de Doutor em Ciências no Programa de Pós-Graduação em Engenharia Eletrônica e Computação, Área de Informática.

Rodrigo Rocha Silva

ABORDAGENS PARA CUBO DE DADOS MASSIVOS COM ALTA DIMENSIONALIDADE BASEADAS EM MEMÓRIA PRINCIPAL E MEMÓRIA EXTERNA: HIC E BCUBING

Tese aprovada em sua versão final pelos abaixo assinados:

Prof. Dr. Celso Massaki Hirata
Orientador

Prof. Dr. Joubert de Castro Lima
Coorientador

Prof. Dr. Luiz Carlos Sandoval Góes
Pró-Reitor de Pós-Graduação e Pesquisa

Campo Montenegro
São José dos Campos, SP – Brasil
2015

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Sobrenome, Prenome do autor
Abordagens para Cubo de Dados Massivos com Alta Dimensionalidade Baseadas em Memória Principal e Memória Externa: HIC e bCubing / Rodrigo Rocha Silva.
São José dos Campos, 2015.
144f.

Tese de doutorado – Curso de Engenharia Eletrônica e Computação, Área de Ciência da Computação – Instituto Tecnológico de Aeronáutica, 2015. Orientador: Prof. Dr. Celso Massaki Hirata

1. Cubo de Dados. 2. Alta Dimensionalidade. 3. Índice Invertido. I. Instituto Tecnológico de Aeronáutica. II. Título

REFERÊNCIA BIBLIOGRÁFICA

ROCHA SILVA, Rodrigo. **Abordagens para Cubo de Dados Massivos com Alta Dimensionalidade Baseadas em Memória Principal e Memória Externa: HIC e bCubing**. 2015. 144f. Tese de doutorado Ciência da Computação – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Rodrigo Rocha Silva
TÍTULO DO TRABALHO: Abordagens para Cubo de Dados Massivos com Alta Dimensionalidade Baseadas em Memória Principal e Memória Externa: HIC e bCubing
TIPO DO TRABALHO/ANO: Tese / 2015

_____

Rodrigo Rocha Silva
Rua Nossa Senhora da Paz, 25 – Vila Lavínia.
CEP: 08735-500, Mogi das Cruzes - SP

ABORDAGENS PARA CUBO DE DADOS MASSIVOS COM ALTA DIMENSIONALIDADE BASEADAS EM MEMÓRIA PRINCIPAL E MEMÓRIA EXTERNA: HIC E BCUBING

Rodrigo Rocha Silva

Composição da Banca Examinadora:

Prof. Dr.	José Maria Parente de Oliveira	Presidente	-	ITA
Prof. Dr.	Celso Massaki Hirata	Orientador	-	ITA
Prof. Dr.	Joubert de Castro Lima	Coorientador	-	UFOP
Prof. Dr.	Carlos Henrique Quartucci Forster	Membro	-	ITA
Prof. Dra.	Cristina Dutra de Aguiar Ciferri	Membro Externo	-	USP
Prof. Dra.	Valeria Cesário Times	Membro Externo	-	UFPE

ITA

Com amor e por serem o alicerce de minha vida, dedico este trabalho:

Aos meus pais, Mauricio e Jovelina.

A minha esposa Dayane e minhas amadas filhas Alice e Helena.

Agradecimentos

Em especial a minha família: minha esposa Dayane, minhas filhas Alice e Helena, minha mãe Jovelina, meu pai Mauricio, meu irmão Diogo, meus tios e amigos Viviane e Saulo.

Ao meu orientador Professor Hirata e ao meu coorientador Professor Joubert, por toda ajuda neste trabalho.

Ao ITA que me acolheu desde 2006 no projeto Harpia e foi um divisor de águas em minha vida, aqui fica todo meu agradecimento.

Agradeço imensamente aos vários amigos e colegas que durante a jornada deste trabalho me ajudaram em especial: Luciano, Leandro, Naresh, Dinesh, Gustavo e Lucas.

À Faculdade de Tecnologia de Mogi das Cruzes e o Centro Paula Souza pelo apoio dado no desenvolvimento deste trabalho.

" O melhor resultado acontece quando todos em um grupo fazem o melhor por si próprio e pelo grupo".

(John Nash)

Resumo

Abordagens para computação de cubos de dados utilizando a estratégia de índices invertidos, tais como Frag-Cubing, são alternativas eficientes em relação às tradicionais abordagens para computação de cubos de dados com alta dimensionalidade, entretanto tais abordagens são limitadas pela memória principal disponível. Neste trabalho, são apresentadas duas abordagens iniciais: qCube e H-Frag. qCube é uma extensão da abordagem Frag-Cubing que possibilita consultas de intervalo e H-Frag é uma abordagem que utiliza memória principal e memória externa a partir de definições do usuário. Com base nas abordagens iniciais, propomos duas outras que utilizam o sistema de memória composto por memória principal e memória externa, o qual chamamos de sistema híbrido de memória, para computar e manter atualizado cubos com alta dimensionalidade e elevado número de tuplas: HIC e bCubing. Em HIC, partições de cubos são armazenados em memória principal e na memória externa utilizando a mesma representação de Frag-Cubing, contudo valores de atributos frequentes são armazenados em memória principal e valores de atributos pouco frequentes são armazenados em memória externa. HIC utiliza um parâmetro, chamado frequência acumulada crítica, para definir quais os valores de atributo são armazenados em memória principal ou em memória externa. bCubing particiona uma lista de identificadores de tuplas (TIDs) implementando a inversão de tuplas em dois níveis: um nível onde o identificador é o índice de bloco (BID) e o segundo nível onde o identificador é o TID. As listas de TIDs dos valores de atributos são armazenadas em memória externa. As listas de BIDs são mantidas em memória principal e indexadas pelos valores de atributos. bCubing é capaz de calcular e manter atualizadas medidas holísticas de forma exata em cubos com alta dimensionalidade e elevado número de tuplas. Experimentos utilizando uma relação com 480 dimensões e 10^7 tuplas mostram que a abordagem bCubing é apenas 30% mais lenta do que Frag-Cubing para computação de cubos e aproximadamente 3 vezes mais rápida para responder consultas multidimensionais complexas a partir de tais relações. Um cubo massivo com 60 dimensões e 10^9 tuplas foi computado por bCubing usando 84 GB de RAM, enquanto o Frag-Cubing não computou tal cubo em uma máquina com 128 GB de RAM sem realizar operações de swap do sistema operacional. O impacto do cálculo de medidas holísticas em um cubo de dados com alta dimensionalidade também foi avaliado e os resultados demonstram que a abordagem bCubing gasta, em média, 10% mais tempo ao calcular medidas holísticas do que consultas com medidas COUNT. A abordagem bCubing respondeu consultas em um cubo de dados com 1.2 bilhões de tuplas em até 4 minutos, sendo uma destas consultas

Q composta por dois operadores de subcubo e um operador EQUAL. A consulta *Q* calculou três medidas holísticas de forma exata: desvio padrão, mediana e moda.

Abstract

Cubing methods based on inverted indices, such as Frag-Cubing, are efficient alternatives to conventional approaches of computing high-dimensional data cubes. methods approaches for data cubing based on inverted indices, such as Frag-Cubing, are efficient alternatives to conventional approaches for computing high-dimensional data cubes. However, similar to other memory-based cube solutions, the efficiency of such methods is constrained by available dynamic random-access memory (DRAM). In this work, we propose two initial approaches: qCube and H-Frag. qCube is a high dimension sequential range cube approach which implements Equal, Not Equal, Distinct, Sub-cube, Greater or Less than, Some, Between, Similar and Similar query operators over a high dimension data cube. H-Frag is an approach which adopts both main and external memories, but the end-user must configure its memory thresholds manually. Based on initial approaches, we develop two other contributions to improve hybrid memory systems utilization: HIC and bCubing. HIC describes a hybrid memory system to store cube partitions in external memory. The most frequent attribute values are stored in DRAM and the remaining attribute values are retained in hard disk drive storage. HIC introduces a new property, named critical cumulative frequency, to define which attribute values will be stored in RAM or in external memory. bCubing partitions a list of TIDs implementing the inverted index in two levels: one level in which the identifier is the block index (BIDs) and the second level in which the identifier is the tuple index (TIDs). The TIDs of attribute values are retained in hard disk drive storage. The BIDs are kept in main memory indexed by the attribute values. bCubing is able to calculate and maintain holistic measures accurately updated within high dimensional cubes with high number of tuples. Tests using a relation with 480 dimensions and 10^7 tuples show that bCubing is only 30% more time slower than Frag-Cubing when computing a data cube, and approximately 3 times faster than Frag-Cubing when answering complex cube queries. A massive cube with 60 dimensions and 10^9 tuples was computed by bCubing using 84 GB of RAM, while Frag-Cubing could not compute such a cube in a machine with 128 GB of RAM without operating system swaps. We also computed holistic measures in a high dimension data cube with bCubing and results demonstrated that bCubing spends, on average, 12% longer to calculate holistic measures than queries with COUNT measures. bCubing answered queries from a data cube with 1.2 billion tuples in up to 4 minutes, where a query *Q* has two sub-cub operators and one EQUAL operator. Query *Q* accurately calculated three holistic measures: standard deviation, median and mode.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Generalização de dados é um processo de abstração de um grande conjunto de dados relevantes para uma determinada tarefa a partir de um nível conceitual relativamente inferior para níveis conceituais mais elevados ou superiores. Para realizar análises assumimos a hipótese que usuários necessitam organizar os dados em diferentes níveis de granularidade e de diferentes perspectivas.

Do ponto de vista de análise de dados, a generalização de dados é uma forma de mineração de dados descritiva que apresenta os dados de uma forma agregada e concisa, assim como expõe propriedades gerais dos dados (HAN; KAMBER, 2006).

Existem duas alternativas cientificas para prever comportamentos a partir de uma abstração. A primeira alternativa estabelece o uso de teorias que descrevem o comportamento de uma abstração através de fenômenos físicos, usando a matemática. A segunda alternativa utiliza o comportamento passado descrito por uma enorme quantidade de dados para prever o comportamento de uma abstração. Tal alternativa é adotada por abordagens orientadas a dados, utilizadas quando nenhuma teoria existe.

As abordagens orientadas a dados são divididas em abordagens orientada à hipótese e abordagens orientada à descoberta. Em uma abordagem orientada à hipótese um analista de negócios normalmente inicia a exploração de dados manualmente, ou seja, ele tenta encontrar anomalias no conjunto de dados, porém muitas vezes o espaço de busca é muito grande, tornando tal tarefa inviável (SARAWAGI; AGRAWAL; MEGIDDO, 1998).

Em uma abordagem orientada a descoberta, um analista de negócios realiza exploração de dados guiado por indicadores de exceções pré-computados em diversos níveis de detalhe, com isso as chances do usuário perceber padrões anormais pode aumentar (SARAWAGI; AGRAWAL; MEGIDO, 1998).

Em geral, as abordagens baseadas em dados permitem descobrir padrões não triviais implícitos, previamente desconhecidos e potencialmente úteis a partir de dados (HAN; KAMER, 2006).

A tecnologia *Data Warehouse* (DW) e *Online Analytical Processing* (OLAP) executam a generalização de dados agregando enormes quantidade de dados em vários níveis de abstração. Assim a tecnologia DW se torna um dos elementos essenciais de apoio à decisão e atrai a atenção tanto da indústria como das comunidades de pesquisa. Ferramentas de análise, com base em abordagens orientada à hipótese ou orientada à descoberta, são desenvolvidas e

consequentemente reforçam a tendência predominante para o uso de DW. Sistemas OLAP, que são tipicamente dominados por consultas complexas que envolvem operadores *group-by* e operadores de agregações, são as principais características entre essas ferramentas.

A tecnologia OLAP baseia-se no operador relacional chamado cubo de dados. O operador cubo de dados é a generalização do operador relacional *group-by* sobre um conjunto de atributos da base de dados, normalmente um DW.

O operador cubo de dados proposto inicialmente por GRAY et al. (1997) permite materializar todas ou algumas células de um cubo, representadas por medidas e dimensões.

Dimensões e hierarquias – Uma dimensão é uma perspectiva do processo decisório formada por um subconjunto de atributos, onde cada dimensão pode ser organizada em hierarquias. Pode-se citar a dimensão tempo como uma dimensão fundamental ao processo decisório. A dimensão tempo é formada, por exemplo, pelos atributos ano e dia. Uma hierarquia pode ser formada pelos atributos ano e turno, outra por mês, semana e turno e assim sucessivamente. Valores de atributos são instâncias de atributos, tais como janeiro, março ou mesmo dia 1º ou 18 de um mês. Como normalmente consideramos várias perspectivas no processo decisório, daí o nome cubo multidimensional de dados.

Medidas - Medidas são atributos associados a uma ou várias funções estatísticas. Um valor de uma medida salário pode ser R$9.800,00. Uma função estatística pode ser classificada em três tipos (O'NEIL; QUASS, 1997):

- Distributiva: agregada por operação distributiva sobre dados atômicos ou medidas distributivas como: *count*, *sum*, *max*, *min*;
- Algébrica: agregada por operações algébricas sobre dados atômicos ou medidas distributivas ou algébricas, por exemplo: média e desvio padrão;
- Holística*:* agregada por operações sem limite constante sobre o espaço necessário para armazenar sub-agregações, típicos exemplos são: mediana e moda. Tipicamente processadas apenas de forma aproximada pelas clássicas abordagens de computação de cubo de dados.

Dentre os três tipos de medidas descritos anteriormente, pode-se observar que medidas holísticas impõem desafio para seu cálculo à medida que o volume de dados aumenta, seja elevando o número de dimensões ou tuplas ou ambos. Na literatura, muito se explora o cálculo aproximado (NANDI; et al., 2011; LENZ; THALHEIM, 2001; VASSILIADIS, 1998; GIBBONS; MATIAS, 1998; POOSALA; GANTI, 1999; CHIOU; SIEG, 2001; LI; et al., 2004;

LAKSHMANAN; PEI; HAN, 2002; BRAZ; et al., 2007) de medidas holísticas. Neste trabalho, é detalhado o cálculo exato de medidas holísticas complexas como *rank*, variância e moda.

Uma célula em um cubo de dados n-dimensional de dados pode ser definida como $C = \{d_1, d_2, ..., d_n, m_1, m_1, ..., m_z\}$, onde d_n indica o valor da dimensão n e m_j indica a j-ésima medida na célula. As consultas são respondidas mais rapidamente à medida que mais células estejam previamente computadas em um cubo de dados. Para gerar todas ou algumas células de um cubo de dados, diversos trabalhos foram propostos (LIMA, 2009; AGARWAL; et al., 1996; CHEN; et al., 2008; FANG, et al.; 1998; FRIEDMAN; PAWLOWSKI; CIESLEWICZ, 2009; HAN; et al., 2001; HARINARAYAN; RAJARAMAN; ULLMAN, 1996; NG; WAGNER; YIN, 2001; FENG; et al., 2004; LEHNER; et al., 2000).

Um cubo de dados possui células base e células agregadas. Suponha a relação de entrada de *R* com três dimensões (*A*, *B* e C) e uma única tupla, $t_1 = (a_1, b_1, c_1, m)$, onde a_1, b_1 e c_1 são valores de atributo; e *m* representa um valor numérico de um valor de medida para t_1.

Assim dado a relação *R*, um cubo de dados completo possui oito tuplas que representam todas as possíveis agregações de *R*: t_1, $t_2 = (a_1, b_1, *, m)$, $t_3 = (a_1, *, c_1, m)$, $t_4 = (*, b_1, c_1, m)$, $t_5 = (a_1, *, *, m)$, $t_6 = (*, b_1, *, m)$, $t_7 = (*, *, c_1, m)$, e $t_8 = (*, *, *, m)$, onde o asterisco "*" indica um carácter universal que representa o agregado de todos os valores de atributos de uma dimensão do cubo de dados. De um modo geral, partindo de um cubo de dados computado a partir de *R* com cardinalidade $C_a = C_b = C_c = 1$ tem-se 2^3 ou $(C_a + 1) \times (1 + C_b) \times (C_C + 1)$ tuplas. Neste exemplo, t_1 é uma célula base e t_2, t_3, t_4, t_5, t_6, t_7, e t_8 são células agregadas.

Se considerarmos uma relação com as dimensões *ABCD* em vez da relação com as dimensões ABC, e $C_a = C_b = C_c = C_d = 2$, existirão 16 células base e 81 células agregadas em um cubo de dados completo.

Como observado, o problema de computar e manter um cubo de dados é exponencial em tempo de execução e consumo de memória. Desta forma, reduzir o consumo de memória de um cubo de dados, assim como seu tempo de computação, atualização e consulta, incluindo medidas holísticas, é de fundamental importância para o desenvolvimento de serviços OLAP.

Neste trabalho, propomos quatro abordagens eficientes para computar e manter atualizado um cubo de dados com alta dimensionalidade e com elevado número de tuplas, sendo que três destas abordagens (H-Frag, HIC e bCubing) adotam sistema de memória híbrido. As abordagens propostas que fazem uso de um sistema híbrido de memória (principal e externa) possuem baixo tempo de execução e baixo consumo de memória RAM. Neste trabalho também é proposto um método para o cálculo exato de medidas holísticas a partir de bases de dados com elevado número de tuplas.

As abordagens HIC e bCubing necessitam de 6 a 14 vezes menos de memória RAM e respondem a diferentes tipos de consulta com desempenho similar a abordagem Frag-Cubing, contudo HIC e bCubing são mais eficientes para responder consultas em cubos massivos que necessitam utilizar memória externa para serem computados.

As abordagens que apresentamos neste trabalho realizam consultas com três operadores combinatórias em bases massivas, consultas estas não processadas pela abordagem Frag-Cubing, além de processar medidas holísticas de forma exata em consultas pontuais e consultas a subcubos.

1.1 Problemas

Além do problema de como gerar todas ou algumas células organizadas hierarquicamente e garantir consultas diversas a tais células de um cubo, há também o problema de como manter um cubo de dados atualizado segundo algum intervalo de tempo. Uma abordagem para cubo de dados não é completamente prática sem as funcionalidades para os problemas descritos anteriormente.

Possibilitar o cálculo de medidas holísticas e bases de dados massivas tornam os problemas já descritos ainda mais árduos. Diversos trabalhos na área de DW (LIMA, 2009; XIN; et al., 2003; LABIO; et al., 2000; MUMICK, I.; QUASS; MUMICK, B., 1997; ROSS; SRIVASTAVA; SUDARSHAN, 1996; CHEN; et al., 2008; NG; WAGNER; YIN, 2001; DOKA; TSOUMAKOS; KOZIRIS, 2011) apresentam testes apenas para medidas algébricas ou distributivas.

Medidas holísticas não são decompostas e não usufruem da hierarquia na sua computação. Não foi encontrado trabalho na literatura que calcula medidas holísticas de forma exata a partir de cubos de dados com alta dimensionalidade e elevado número de tuplas.

A dimensionalidade é um problema que afeta quase todas as abordagens propostas na literatura. A solução para cubos com alta dimensionalidade é sua computação parcial, conforme explica DEHNE, EAVIS e CHAPLIN (2001). Cubos parciais não computam algumas células, mas assumem que é possível gerá-las quando necessário. Entretanto, as abordagens que endereçam solução para alta dimensionalidade e alta cardinalidade não consideram calcular medidas holísticas. As abordagens que utilizam índices bitmap para resolver o problema da alta dimensionalidade não são projetadas para permitir que um cubo de dado seja atualizado, pois demandam reprocessamento, tal necessidade inviabiliza o uso de destas abordagens em cenários de grande demanda de atualização.

A utilização de memória externa, conforme apresentado pela abordagem Top-k H-Cubing (HAN; et al., 2001), gera um enorme número de swaps e nem uma ordenação prévia pode resolver a estratégia de manter agregações para cada tupla utilizando a abordagem *bottom-up*, portanto a abordagem Top-k H-Cubing é inviável para cubos com alta dimensionalidade.

Partimos da hipótese de que um cubo de dados pode ser representado parcialmente em memória primária e memória externa e que o espaço de memória de trabalho necessário para computação e consultas é menor do que a representação total do cubo. Consideramos também que o tempo de processamento de consultas pode ser reduzido uma vez que somente as partições do cubo de dados necessárias para responder tais consultas são recuperadas da memória externa.

1.2 Objetivos

Diante dos problemas apresentados, propõe-se as abordagens HIC e bCubing para computação, atualização e consulta eficiente de cubos parciais com dados uniformemente distribuídos ou não, com alta cardinalidade, elevado número de tuplas e com alta dimensionalidade. As abordagens propostas devem endereçar solução para o cálculo aproximado ou exato de múltiplas medidas holísticas.

Como objetivo específico, propomos uma alternativa de solução para o índice invertido proposto pela abordagem Frag-Cubing (LI; HAN; GONZALEZ, 2004). Em cenários onde o número de tuplas é massivo e onde há atualizações frequentes a adoção de um índice simples, conforme adotado na abordagem Frag-Cubing, é insuficiente. Neste sentido, passamos a investigar como particionar o índice invertido em blocos de tal forma que o mesmo possa ser armazenado de forma incremental em memória externa. A solução proposta deve atender requisitos de consulta também, portanto espera-se que poucas partições devam ser suficientes para responder consultas multidimensionais complexas (combinatoriais) que tenham como resultado subcubos.

O escopo deste trabalho restringe-se a: (i) uma revisão da área de modelagem multidimensional, descrevendo seus principais conceitos, (ii) uma descrição de trabalhos correlatos incluindo seus benefícios e limitações, (iii) a concepção e desenvolvimento de duas novas abordagens para computar, atualizar, consultar e representar um cubo de dados parcial utilizando memória externa, (iv) um extenso estudo de desempenho, incluindo a utilização de conjuntos de dados sintéticos e reais, e (v) discussões sobre as possíveis extensões e limitações de nossas abordagens propostas. Desta forma, está fora do escopo deste trabalho a computação de cubos de dados completos, incluindo cubos fechados, quociente, iceberg, assim como

executar consultas top-k. Também está fora do escopo deste trabalho abordagens para arquiteturas de computadores de alto desempenho, tais como clusters, máquinas massivamente paralelas utilizando placas gráficas ou mesmo grade computacionais.

1.3 Organização da Tese

No capítulo 2 apresentamos a fundamentação teórica onde escrevemos conceitos como DW, cubo de dados, esquemas multidimensionais, medidas, hierarquias, OLAP, operações OLAP e uso de memória externa para permitir uma melhor compreensão dos capítulos restantes da tese.

No capítulo 3 descrevemos os trabalhos correlatos, incluindo a abordagem Frag-Cubing (LI; HAN; GONZALEZ, 2004) que foi a primeira a endereçar uma solução para computação de cubos de alta dimensionalidade utilizando índices invertidos. Também no capítulo 3, detalhamos abordagens que propõem computar cubos com alta dimensionalidade utilizando índice bitmap, os seus benefícios e limitações. Trabalhos que utilizam memória externa para computar e representar cubos de dados, assim como abordagens que se propõem a calcular medidas holísticas, são descritos e discutidos.

No capítulo 4 é apresentado duas contribuições do nosso trabalho, provenientes de estudos iniciais. Apresentamos as abordagens qCube e H-Frag, incluindo suas estratégias de computação e consulta. Neste capítulo também é apresentado alguns experimentos comparativos, assim como discussão sobre os resultados.

Nos capítulos 5 e 6 apresentamos as duas contribuições finais do nosso trabalho. Apresentamos as abordagens HIC e bCubing, incluindo suas estruturas de dados, seus algoritmos de computação, atualização e consulta. Estes capítulos também apresentam os experimentos comparativos, assim como discussão sobre os resultados.

No capítulo 7 apresentamos uma discussão sobre a aplicabilidade das abordagens HIC, bCubing e Frag-Cubing em diferentes cenários.

O capítulo 8 apresenta as conclusões, realça as contribuições do nosso trabalho e descreve os trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo relatamos os conceitos básicos relacionados à tese, divididos em nove seções. As Seções 2.1, 2.2 e 2.3 descrevem o conceito de armazém de dados (Data Warehouse), OLAP e conjunto de operações OLAP. Na Seção 2.4, expomos o operador cubo de dados, a motivação para sua concepção e os benefícios de seu uso. A Seção 2.5 e 2.6 definem o conceito de células de um cubo de dados e hierarquia. Na Seção 2.7, medidas e seus tipos são descritos e detalhados. Na Seção 2.8, apresentamos as estratégias de computação de cubo de dados. Finalmente, na Seção 2.9, os esquemas multidimensionais existentes são descritos.

2.1 Data Warehouse

Um Armazém de Dados ou Data Warehouse (DW) é um repositório de dados orientado por assunto, 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 é geralmente representado por um modelo dimensional (KIMBALL, 1996) que permite eficiência na organização dos dados e na recuperação de informações gerenciais. Neste modelo são definidos os seguintes elementos: fatos, dimensões e medidas. Um fato corresponde ao assunto de negócio a ser analisado, por sua vez cada dimensão é uma perspectiva de visualização do assunto de negócio e medidas são valores numéricos que quantificam o assunto de negócio. Em um DW, uma das dimensões é sempre temporal para permitir a análise do assunto de negócio ao longo do tempo.

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 (MOREIRA; LIMA, 2012).

2.2 OLAP

O termo *On-line Analytical Processing* (OLAP) foi criado por (CODD et al, 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.

Geralmente, DWs armazenam os dados utilizados pelas ferramentas OLAP. Cada ferramenta OLAP deve manipular um novo tipo abstrato de dados (TAD), chamado de cubo de dados, utilizando estratégias específicas devido ao modo de como os dados são armazenados, sendo classificadas em (MOREIRA; LIMA, 2012):

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

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

- ***Hybrid* OLAP (HOLAP)**: 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 Conjunto de operações OLAP

Ao iniciar uma consulta em um DW é necessário traduzi-la de forma inteligível ao ambiente computacional assim, devem ser oferecidos aos analistas meios para realizar eficientemente uma consulta a fim de obter resultados coerentes. Como solução, os desenvolvedores de ferramentas OLAP fornecem apoio para as operações de derivação de dados complexos, que recebem o nome de *slice* e *dice*.

O suporte às operações *slice* e *dice* é uma das principais características de uma ferramenta OLAP. A operação *slice* provida pelas ferramentas OLAP faz restrição de um valor ao longo de uma dimensão. Já a operação *dice* é mais complexa, pois faz restrições de valores em várias dimensões (WREMBEL; KONCILIA, 2007).

Algumas das operações OLAP típicas para dados multidimensionais são ilustradas na Figura 2.1. Neste exemplo tem-se um cubo com três dimensões que são tempo, disciplina e departamento de uma universidade, sendo a medida a nota e a função de agregação a média.

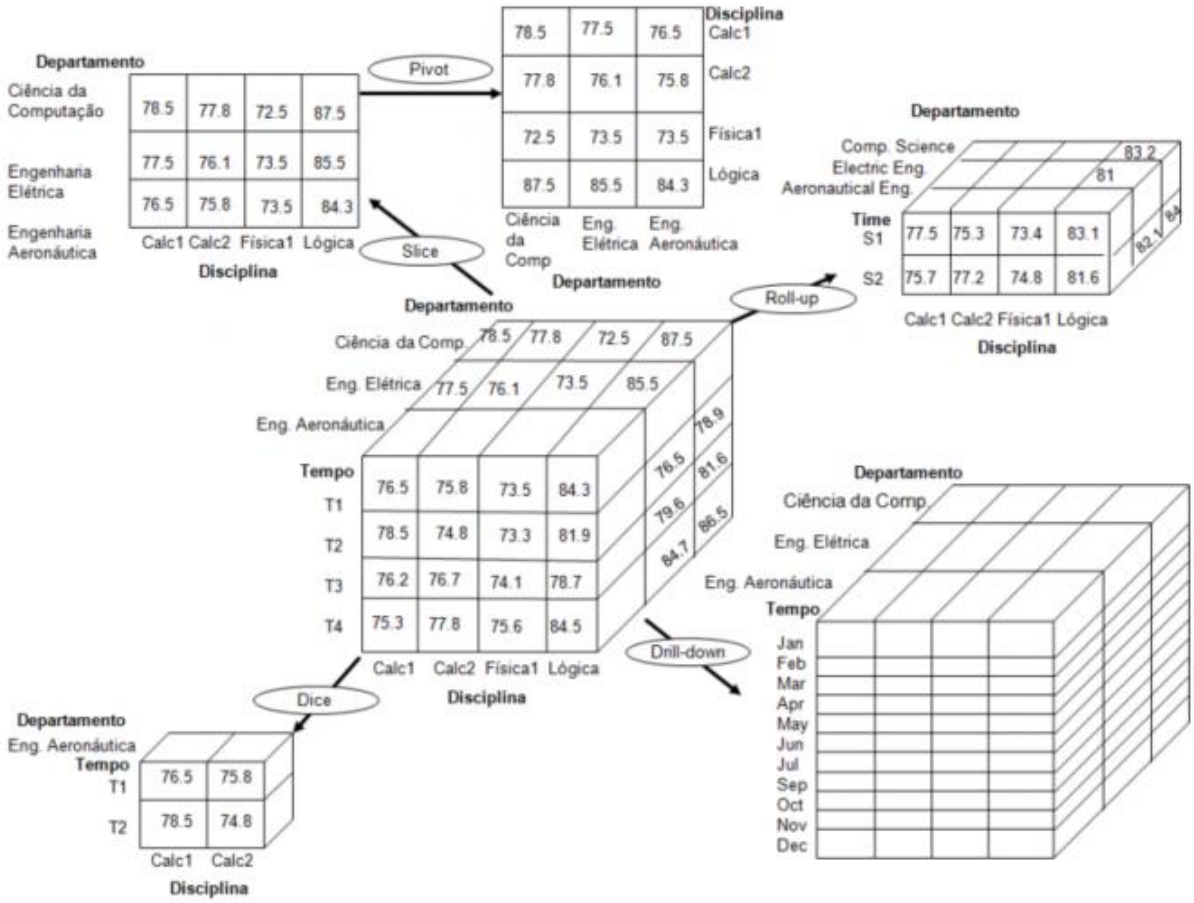

FIGURA 2.1 - Exemplo de Operações OLAP.

Algumas dimensões possuem diferentes hierarquias (ou diferentes níveis de agregação). Se o usuário almeja navegar nos dados a partir dos níveis hierárquicos inferiores para níveis mais altos da hierarquia, dizemos que se trata de uma operação *roll-up*. No caso de uma operação inversa dizemos que se trata de uma operação *drill-down*. Por exemplo, se o usuário desejar saber agregações na seguinte ordem: dia → mês → trimestre, uma operação roll-up irá possibilitar tal visualização. Uma operação drill-down, permite que o usuário navegue no seguinte sentido: trimestre → mês → dia.

Operações de *slice* ("fatiamento") realizam seleções em uma dimensão de um cubo de dados, resultando em um subcubo. A Figura 2.1 apresenta uma operação de slice, onde as notas são selecionadas de um cubo central utilizando como critério o *time="S1"*. A operação *dice* determina um subcubo realizando uma seleção em duas ou mais dimensões.

A Figura 2.1 apresenta uma operação de *dice* no cubo central utilizando o critério de seleção, que inclui três dimensões: (departamento="Eng. Aeronáutica") e (*time="T1"*ou *"T2"*) e (disciplina="Calc1"ou "Calc2").

A operação de pivotação (*pivot*) permite ao usuário mudar as posições das dimensões em foco, ou seja, possui a flexibilidade de alterar o eixo de visualização dos dados, portanto permite alterar linhas por colunas. Na Figura 2.1, é possível visualizar a operação de pivotação, onde os eixos da dimensão disciplina estão rotacionados e "fatiados" (isso porque a pivotação foi aplicada após o resultado da operação de *slice*).

2.4 Cubo de dados

A fim de superar as limitações do operador group-by, que não consegue suprir a necessidade de relacionar dados multidimensionais de maneira eficiente (GRAY et al., 1997), foi concebido o operador cubo de dados por GRAY et al. (1997), definido como um operador relacional que gera todas as combinações possíveis de seus atributos de acordo com uma medida.

Um cubo de dados consiste em um conjunto de medidas para análise e um conjunto de dimensões que provêm o contexto das medidas. Uma medida é um atributo cujos valores definem os fatos de interesse na análise e em cujos valores são aplicadas as funções de agregação (soma, quantidade, média, sentimento, união ou interseção de geo-objetos). Uma dimensão é um elemento que participa de um fato e que determina o contexto de um assunto de negócios. Dimensões podem ser compostas por membros que podem conter hierarquias. Membros são as possíveis divisões ou classificações de uma dimensão. Por exemplo, a dimensão disciplina, pode ser dividida nos seguintes membros: curso, departamento e área, e a dimensão tempo em: ano, mês e dia.

A organização de um cubo de dados possibilita ao usuário a flexibilidade de visualização dos dados a partir de diferentes perspectivas, já que o operador gera combinações através do conceito do valor ALL, onde este conceito representa a agregação de todas as combinações possíveis de um conjunto de valores de atributos. Operações em cubos de dados existem a fim

de materializar estas diferentes visões, permitindo busca e análise interativa dos dados armazenados.

Um cubo de dados é composto por células e cada célula possui valores para cada dimensão, incluindo ALL, e valores 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.

A computação do cubo de dados é um problema exponencial em relação ao tempo de execução e ao consumo de memória, portando dada uma relação de entrada *R* com tuplas de tamanho *n*, a saída é *2n*, onde *n* é o número de dimensões de um cubo. O operador cubo de dados é ilustrado na Figura 2.2, onde é calculada a quantidade de matrículas, representada pela coluna matrículas. Cada tupla possui três valores de atributos e um de medida. Suponha a tupla de entrada (Ciência da Computação, 2014, Banco de Dados), onde $n = 3$. Desta maneira existem 8 tuplas de saída, já que $2^3 = 8$. As tuplas são: [(Ciência da Computação, 2014, Ciência da Computação), (Ciência da Computação, 2014, ALL), (Ciência da Computação, ALL, Banco de Dados), (ALL, 2014, Banco de Dados), (Ciência da Computação, ALL, ALL), (ALL, 2014, ALL), (ALL, ALL, Banco de Dados), (ALL, ALL, ALL)].

Curso	Ano	Disciplina	Matrículas
Ciência da Comp.	2014	Banco de Dados	25
Ciência da Comp.	2013	Banco de Dados	40

Cubo Completo

Curso	Ano	Disciplina	Matrículas
Ciência da Comp.	2014	Banco de Dados	25
Ciência da Comp.	2013	Banco de Dados	40
Ciência da Comp.	2014	ALL	25
Ciência da Comp.	2013	ALL	40
Ciência da Comp.	ALL	Banco de Dados	65
Ciência da Comp.	ALL	ALL	65
ALL	2014	Banco de Dados	25
ALL	2013	Banco de Dados	40
ALL	2014	ALL	25
ALL	2013	ALL	40
ALL	ALL	Banco de Dados	65
ALL	ALL	ALL	65

FIGURA 2.2 - Cubo de Dados

2.5 Células de um cubo de dados

Diversos subcubos compõe um cubo de dados e cada subcubo é composto por diversas células base e células agregadas. Deste modo uma célula em um subcubo base é uma célula base. Da mesma maneira 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 (LIMA, 2009).

Caso exista 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 existir entre células. Em um cubo de dados *n*-dimensional, uma célula $a = (a_1, a_2, a_3, ..., a_n, medidas)$ de nível *i* é um ancestral de uma célula $b = (b_1, b_2, b_3, ..., b_n, medidas)$ de nível *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, 2006).

Com base no mesmo exemplo, uma célula *a = (T1, *, *, 78.9)* com um membro e uma célula *b (T1, *, Calc1, 76.3)* com dois membros são ancestrais da célula *c = (T1, Ciência da Comp., Calc1, 78.8)* que possui três membros, e *c* é descendente de *a* e *b*, onde *b* é pai de *c*.

2.6 Hierarquias

Os membros de uma dimensão são organizados (agregados) em níveis de hierarquias. Por exemplo, a dimensão tempo da Figura 2.1 poderia organizar seus membros em níveis de granularidade, como ano, trimestre, mês, dia; ou seja, em uma hierarquia temporal. Em uma hierarquia, a granularidade de um membro de nível inferior é sempre menor do que a de um nível superior. Por exemplo, a granularidade de dia é menor que a de ano. A disposição dos membros de uma dimensão, nos níveis de uma hierarquia desta dimensão, deve ser feita respeitando-se o grau de agregação dos mesmos, pois um membro de menor granularidade sempre deve estar imediatamente abaixo do seu membro de granularidade maior seguindo-se a hierarquia (PEREIRA, 2013).

Pode existir mais de uma hierarquia em uma dimensão, com base 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.

2.7 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 particionados 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 ser 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 (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.

2.8 Computação de cubos de dados

Computar um cubo de dados é uma tarefa essencial, dado que a pré-computação de parte ou de todo o cubo de dados pode reduzir substancialmente o tempo de execução e melhorar o desempenho de sistemas OLAP. Porém, esta tarefa é um dos problemas mais importantes e pesquisados na área de DW. Já que o problema possui complexidade exponencial em relação ao número de dimensões, a materialização completa de um cubo envolve uma grande quantidade de células e um elevado tempo para sua geração.

Existem três métodos para que seja gerado as agregações de um cubo base: a não materialização, a materialização completa e a materialização parcial.

No caso da não materialização, cubos de dados agregados não são pré-computados, o que leva a uma computação imediata de alto custo, que pode ser extremamente lenta.

A materialização completa pré-computa todas as agregações possíveis de um cubo de dados, gerando um cubo de dados completo. Tal método viabiliza tempos de resposta a uma consulta extremamente rápidas, já que o cubo completo está previamente computado. No entanto, isso pode exigir uma grande quantidade de espaço de memória.

Por fim, um cubo de dados ainda pode ser materializado de forma parcial ou seletiva, desta maneira computa-se um subconjunto específico de inúmeras possibilidades. Também é possível computar um subconjunto de um cubo de dados que contém somente células que satisfazem um dado critério especificado pelo usuário. Este tipo de cubo de dados chama-se iceberg (BEYER; RAMAKRISHNAN, 1999).

Há outra técnica, chamada de *shell fragment*, que computa cubos pequenos (com 3 a 5 dimensões) para formar cubos completos. A medida que é necessário a união dos pequenos cubos, são computadas as agregações necessárias para responder a consulta. Um cubo de dados computado utilizando a técnica shell fragment é chamado de cubo shell (shell cube) (LI; HAN; GONZALEZ, 2004). Finalmente, temos a agregação semântica de cubos de dados, chamada de cubos fechados (*closed cubes*) (XIN et al., 2006) ou *quotient cubes* (LAKSHMANAN; PEI; HAN, 2002), onde um conjunto de células de um cubo de dados com medidas idênticas são encapsuladas em uma única abstração, chamada célula fechada (*closed cell*) ou células de classe.

Materializar parcialmente um cubo de dados apresenta-se um interessante equilíbrio entre 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.

A partir de um cubo base, a computação do operador cubo de dados pode utilizar a estratégia *Top-down* ou *Bottom-up* para a geração dos subcubos remanescentes. Na Figura 2.3, é possível observar a geração de um cubo de dados de quatro dimensões por meio da estratégia *Top-down*. Sendo ABCD um cubo base, os subcubos de três 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 descendentes sucessivos.

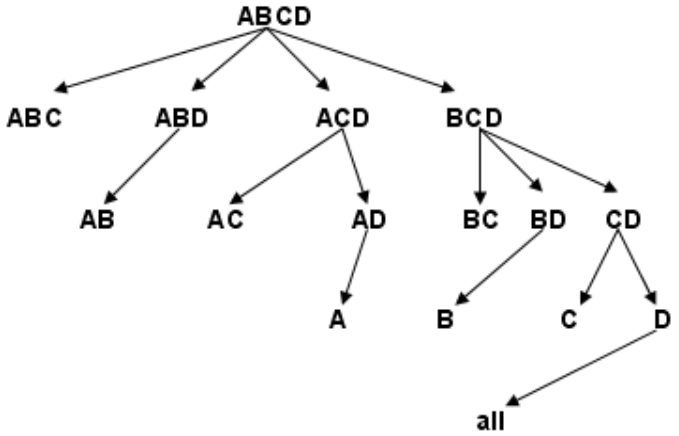

FIGURA 2.3 - Computação de cubo de dados através da estratégia *Top-down*

Na Figura 2.4, é ilustrada 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 do início.

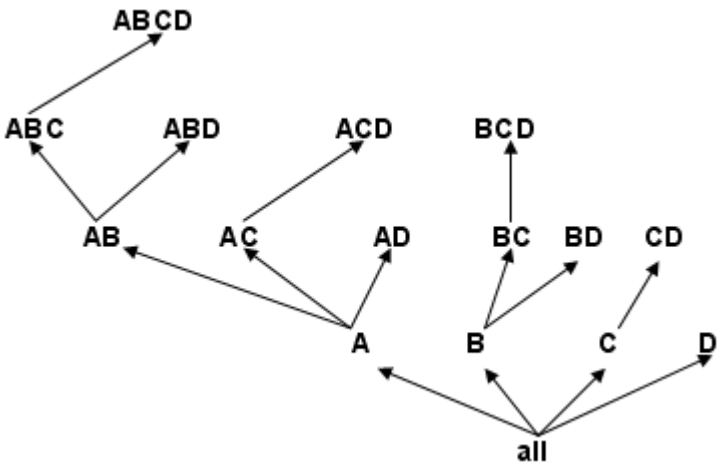

FIGURA 2.4 – Computação de cubo de dados através da estratégia *Bottom-up*

2.9 Esquemas para modelagem dimensional

Existem três esquemas utilizados para modelagem dimensional, são eles: Esquema Estrela (*Star Schema*), Esquema Floco de Neve (*Snowflake Schema*) e Esquema Constelação de Fatos (*Facts Constallation Schema*).

2.9.1 Esquema estrela

Idealizado e criado por Ralph Kimball, o Esquema Estrela é uma forma de dispor as tabelas do modelo relacional para o modelo dimensional (KIMBALL; ROSS, 2002).

Conforme ilustra a Figura 2.5, o Esquema Estrela é uma estrutura com tabelas e ligações bem definidas, baseado no formato de uma estrela. É formado por uma tabela central, denominada tabela de fatos, a qual possui os dados principais da visão da análise, ou seja, o assunto que está sendo analisado, por exemplo, as vendas em dólar, as unidades vendidas, o custo do dólar, etc. Nela ficam ligadas as tabelas de dimensão, que possuem os aspectos pelos quais se deseja observar as medidas relativas ao processo que se está analisando.

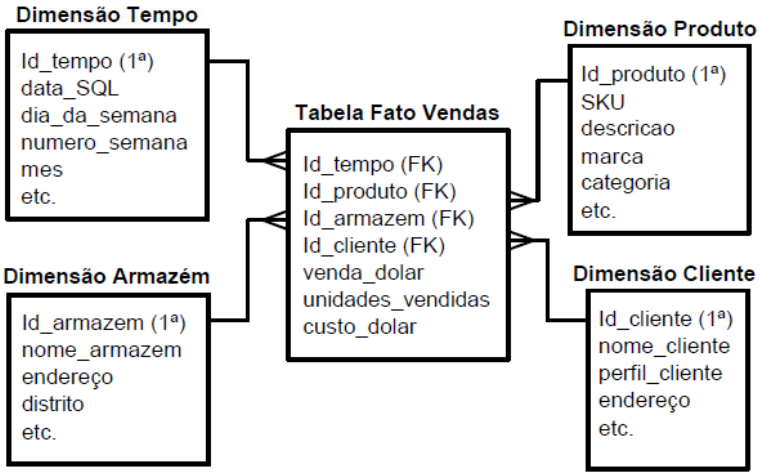

FIGURA 2.5 – Exemplo esquema Estrela

As tabelas dimensionais são desnormalizadas para aumentar o desempenho das consultas. A consulta ocorre inicialmente nas tabelas de dimensão e em seguida na tabela de fatos, assegurando a precisão dos dados através de uma estrutura completa de chaves onde não é preciso percorrer todas as tabelas. Isso garante um acesso mais eficiente e um melhor desempenho (HAN; KAMBER, 2006).

Ao contrário das tabelas de dimensão, a tabela de fatos armazena grandes quantidades de dados históricos, normalmente numéricos, obtidos a partir da interseção de todas as dimensões do Esquema Estrela. Ela também armazena os indicadores de desempenho (medidas) do negócio. Para cada dimensão há uma chave primária que corresponde a um dos campos, chave estrangeira, da chave da tabela de fatos.

2.9.2 Esquema floco de neve

O Esquema Floco de Neve é uma extensão do Esquema Estrela e consiste na decomposição de uma ou mais dimensões, formando hierarquias nas dimensões, isto é, normalizando-as. Esse tipo de esquema é utilizado quando se tem dimensões grandes que são estáticas ou semi-estáticas. A Figura 2.6 ilustra um exemplo geral deste tipo de esquema, nele as dimensões Período, Produto e Mercado foram normalizadas;

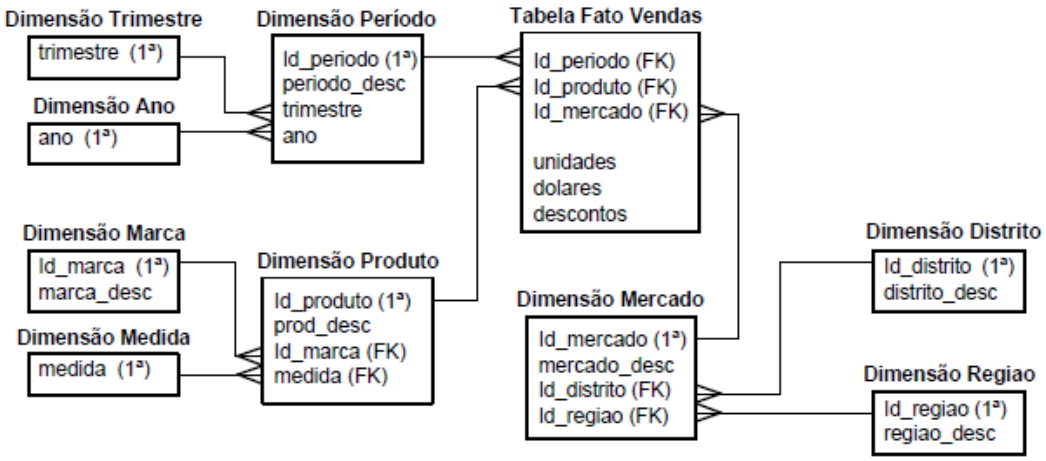

FIGURA 2.6 – Exemplo Floco de Neve

O esquema floco de neve gera uma economia de espaço de armazenamento através da redução do tamanho das tabelas dimensão. Porém, reduz a eficiência da recuperação de dados; normalmente não valendo a pena já que a economia de espaço de armazenamento, segundo Kimball et al. (1996), em geral, não chega 0,1 %.

2.9.3 Esquema constelação de fatos

O Esquema Constelação de Fatos é constituído de duas ou mais tabelas de fatos que compartilham uma ou mais dimensões. Esse tipo de esquema pode ser visto como uma coleção de esquemas estrelas, conforme ilustra a Figura 2.7, na qual as tabelas Dimensão Período e Dimensão Produto e Dimensão Mercado são compartilhadas pela Tabela de Fatos Vendas e Vendas Previstas.

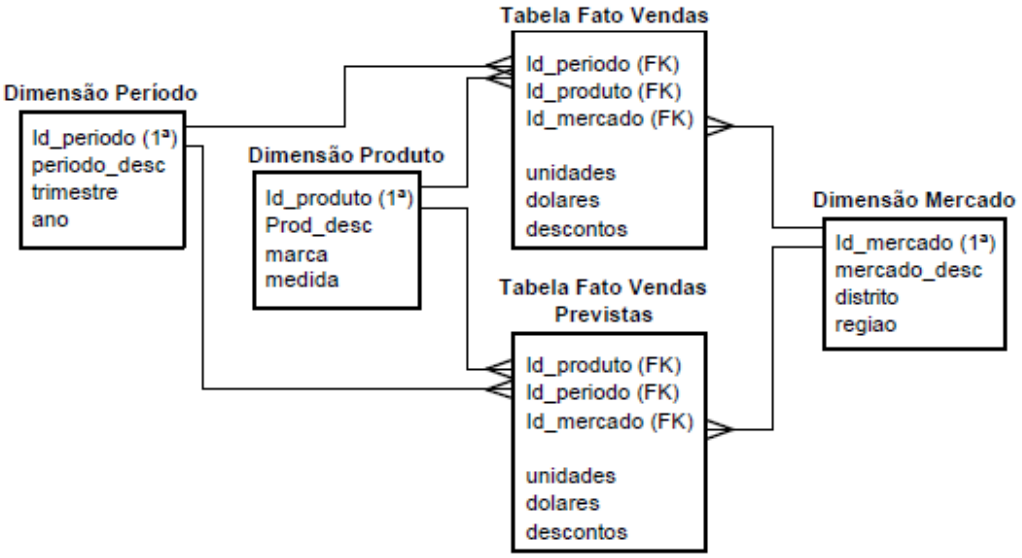

FIGURA 2.7 – Exemplo Constelação de Fatos

Para DWs, o esquema de Constelação de Fatos é mais comumente utilizado, visto que ele pode modelar assuntos múltiplos e inter-relacionados. Geralmente, múltiplas tabelas fatos existem porque elas contêm fatos que não estão relacionados ou por causa da diferente periodicidade de tempo de carga.

2.10 Resumo

No Capítulo 2, apresentamos os principais conceitos da fundamentação teórica da nossa pesquisa. Descrevemos as principais técnicas que realizam agregações em diferentes níveis de abstração. Além disso, foi feita uma descrição sobre o operador cubo de dados, e como ferramentas executam operações OLAP em cubos de dados. Cada cubo de dados é composto por vários subcubos e estes subcubos são compostos por células base e células agregadas, portanto, uma definição formal de ambos os tipos de células do cubo foi descrita.

Um DW representa um modelo de dados multidimensional e tal modelo pode existir sob a forma de um esquema em estrela, esquema floco de neve, ou um esquema de constelação de fatos. As características de cada esquema estão descritas e ilustradas. Os tipos de medida que um cubo de dados pode manipular, incluindo medidas holísticas, e o conceito de hierarquias que podem ser obtidas manualmente ou automaticamente, também são descritos.

No próximo capítulo, são descritos alguns trabalhos correlatos a esta pesquisa. Descrevemos a abordagem Frag-Cubing (LI; HAN; GONZALEZ, 2004) que introduz o uso da técnica de índices invertidos na computação de cubo de dados, também abordamos algumas

limitações das abordagens que utilizam a técnica de índice invertido e bitmap para a computação de cubo de dados de alta dimensionalidade.

Fazemos também no próximo capítulo uma análise de abordagens para computação de cubo de dados que utilizam memória externa e que fazem processamento de medidas holísticas. Abordamos a ideia de cálculo de medidas holísticas através de resultados aproximados, destacando os principais trabalhos que utilizam tal ideia. Por fim, citamos um trabalho que utiliza uma estrutura não sequencial para o cálculo de medidas holísticas.

3 Trabalhos Correlatos

Apesar de diversas abordagens para computação de cubos de dados existirem, como por exemplo a MultiWay (ZHAO; DESPANDE; NAUGHTON, 1997), Star (XIN; et al., 2003), Bottom-Up Computation (BUC) (BEYER; RAMAKRISHNAN, 1999), Dwarf (SISMANIS; et al., 2002), Multidimensional Direct Acyclic Graph Cubing(MDAG) (LIMA; HIRATA, 2007), Multidimensional Cyclic Graph (MCG) (LIMA; HIRATA 2011), poucas implementam soluções para computar cubo de dados com alta-dimensionalidade sequencialmente.

Alguns trabalhos empregam as técnicas de índice invertido e índice bitmap para computar cubos com alta-dimensionalidade (LI; HAN; GONZALEZ, 2004; LENG; et al., 2006; WU; STOCKINGER; SHOSHANI, 2008; FERRO; et al., 2009; LO; et al., 2008; SILVA; LIMA; HIRATA, 2013; SILVA; LIMA; HIRATA, 2015a). Normalmente, o cubo original é decomposto em fragmentos que podem ser reunidos eficientemente para responder uma consulta multidimensional. LI; HAN e GONZALEZ (2004) ilustram o impacto exponencial no consumo de memória nas diferentes abordagens de computação de cubos de dados usando apenas 12 dimensões. Nota-se que há uma clara saturação quando cubos com 20, 50 ou 100 dimensões são computados utilizando abordagens de cubos completos, cubos DWARF, MCG, cubos fechados ou quocientes (BRAHMI, 2012; RUGGIERI; PEDRESCHI; TURINI, 2010; LIMA; HIRATA, 2011; XIN; et al., 2006; SISMANIS, 2002).

Para relações onde a cardinalidade das dimensões e número de tuplas são baixos, o uso de índices bitmap (AMER-YAHIA; JOHNSON, 2000; CHAN; IOANNIDIS, 1998; O'NEIL; QUASS, 1997; WU; BUCHMANN, 1998) se apresenta como uma alternativa extremamente viável para reduzir o tempo e consumo de memória na computação de um cubo de dados. Ainda, a operação de interseção pode ser executada muito rapidamente com a operação bit-AND que padroniza a união de interseções, possibilitando um desempenho muito satisfatório para diferentes tipos de consulta. Existem trabalhos recentes (WU; OTOO; SHOSHANI, 2001, 2002, 2004; FERRO; et al., 2009) que implementam métodos para computação de cubos com alta dimensionalidade e cardinalidade utilizando técnicas de compreensão de dados e indexação bitmap, porém o problema para relações com elevado número de tuplas e alta cardinalidade persiste. Mais à frente neste capítulo são detalhadas as razões de abordagens bitmap não serem eficientes para relações com alta cardinalidade e elevado número de tuplas.

Índices invertidos são mais adequados quando a relação possui um elevado número de tuplas e dimensões com alta cardinalidade (SILVA; LIMA; HIRATA, 2013), contudo técnicas

como as adotadas por abordagens como Frag-Cubing degradam o consumo de memória à medida que o número de tuplas cresce, pois consequentemente nestes cenários, o tamanho do índice invertido cresce substancialmente. As propostas desta tese endereçam soluções também para tal problema. Mais à frente neste capítulo são detalhadas as razões de índices invertidos não serem adequados para bases com elevado número de tuplas.

3.1 Computação de cubos com alta-dimensionalidade com índice invertido

Frag-Cubing (LI; HAN; GONZALEZ, 2004) implementa o conceito de inversão de tupla. Cada tupla *iT* tem um valor de atributo, uma lista de identificadores da tupla (TIDs) e um conjunto de valores de medida. Por exemplo, consideremos quatro tuplas: $t_1 = (tid_1, a_1, b_2, c_2, m_1)$, $t_2 = (tid_2, a_1, b_3, c_3, m_2)$, $t_3 = (tid_3, a_1, b_4, c_4, m_3)$, e $t_4 = (tid_4, a_1, b_4, c_1, m_4)$. Estas quatro tuplas geram oito tuplas invertidas: iTa_1, iTb_2, iTb_3, iTb_4, iTc_1, iTc_2, iTc_3, e iTc_4. Para cada valor de atributo é construído uma lista de ocorrências, assim para a_1 temos $iTa_1 = (a_1, tid_1, tid_2, tid_3, tid_4, m_1, m_2, m_3, m_4)$ onde o valor de atributo a_1 está associado aos TIDs: tid_1, tid_2, tid_3, e tid_4. O identificador de tupla tid_1 tem o valor de medida m_1, tid_2 tem o valor de medida m_2, tid_3 tem o valor de medida m_3, e tid_4 possui o valor de medida m_4. A consulta $q = (a_1, b_4, COUNT)$ pode ser respondida por $iTa_1 \cap iTb_4 = (a_1b_4, tid_3, tid_4, COUNT\ (m_3, m_4))$. Em q, $iTa_1 \cap iTb_4$ indica os TIDs comuns em iTa_1 e iTb_4.

A complexidade da interseção é proporcional ao número de ocorrências de um valor de atributo, mais precisamente é igual ao tamanho da menor lista. Em nosso exemplo, iTb_4 com dois TIDs é a menor lista. O número de TIDs associado a cada valor de atributo pode ser enorme, assim relações com dimensões de baixa cardinalidade e elevado número de tuplas necessitam de alta capacidade de processamento. Listas de TIDs pequenas permitem que consultas sejam respondidas rapidamente, portanto relações com baixo *skew* e alta cardinalidade são mais adequadas de serem computadas pela abordagem Frag-Cubing. *Skew* é o grau de uniformidade dos valores de atributos numa relação. *Skew* zero indica relação com valores de atributos uniformemente distribuídos e quanto maior o skew menos uniformemente distribuída a relação se encontra.

Neste sentido, abordagens como Frag-Cubing, que fazem uso apenas de memória principal (RAM), normalmente não conseguem computar cubos com alta dimensionalidade, cardinalidade e elevado número de tuplas eficientemente, pois extrapolam a memória existente e necessitam operações de swap do sistema operacional.

3.2 Computação de cubos com alta-dimensionalidade com índice bitmap

LENG et al. (2006) propõe um método chamado *Compressed Bitmap Index Based*, que implementa a ideia proposta na abordagem Frag-Cubing, substituindo o índice invertido por um índice bitmap, assim cada valor de atributo *at* tem um conjunto de bits *B* que indica se o valor de atributo está presente em cada uma das tuplas da relação de entrada. É observada uma clara limitação na proposta de LENG et al. (2006), já que os conjuntos de bits tornam-se enormes. Propondo uma solução para o problema dos grandes conjuntos de bits, os autores apresentam uma compactação de índices, que elimina as sequências de zeros e uns de *B*. Mesmo com tal compressão de bits a abordagem é útil para relações com poucas tuplas, pois cada nova tupla gera novas entradas em vários conjuntos de bits *B*. A cardinalidade se apresenta como outro grande problema, já que para cada novo valor de atributo *at'* um novo conjunto de bits *B'* deve ser criado com tamanho igual ao número total de tuplas. No trabalho de LI, HAN e GONZALEZ (2004) as limitações de índices bitmap são reforçadas.

O trabalho de WU, OTOO e SHOSHANI (2004) é uma abordagem para computação de cubo de dados que introduz o esquema de compressão bitmap *word-aligned hybrid* (WAH). WAH é considerado uma das estratégias mais eficientes de compressão de índices de bitmap. É uma solução híbrida baseada em *run-length encoding* e bitmaps literais (WU; OTOO; SHOSHANI, 2001) onde uma sequência de bits do mesmo tipo é representada pelo valor do bit e a sua quantidade (STOCKINGER; WU, 2006).

O trabalho de FERRO; et al. (2009) apresenta uma abordagem para a computação de cubo de dados utilizando um índice de bitmap, chamada BitCube. A abordagem é semelhante à Frag-Cubing, realizando partições horizontais na relação de acordo com os valores dos atributos de uma dimensão, podendo executar *group-bys* utilizando a estratégia bottom-up para cada partição. Além da computação e da consulta em cubo de dados, os autores apresentam cálculos de medidas distributivas como soma, máximo e mínimo. Infelizmente, há o elevado consumo de memória para construção e atualização dos índices bitmaps.

Para resolver a limitação de alta cardinalidade, os autores da abordagem BitCube (FERRO; et al. 2009) propõem uma extensão utilizando a técnica de compressão WAH, proposta em WU, OTOO e SHOSHANI (2004). Mesmo com tal otimização, a atualização do cubo de dados ainda se mantém como severa restrição a sua adoção em cenários reais, assim como inviabiliza o cálculo exato de medidas algébricas e holísticas devido à perda de precisão exigida na compressão.

A abordagem S-OLAP (LO; et al., 2008) assume o conceito de padrões como dimensões, conseguindo agrupar sequências de dados baseando-se em padrões de ocorrências. Funções agregadas comuns, tais como a frequência, podem ser aplicadas a grupos de agregação. Este trabalho sofre dos mesmos problemas das abordagens descritas anteriormente, não permitido atualização no cubo de dados, cálculo de medidas holísticas e não possibilita a computação de cubos com elevado número de tuplas e alta cardinalidade por dimensão.

É proposto por HU, et al. (2005) uma abordagem chamada *Shell Fragment Mini-Cube* que realiza uma fragmentação hierárquica, particionando um cubo de dados de alta dimensionalidade em mini-cubos. Usando um esquema de codificação para hierarquias e resultados de pré-agregações, as consultas OLAP são executadas dinamicamente a partir dos subcubos. *Shell Fragment Mini-Cube* possui as mesmas deficiências de outras abordagens que utilizam índice de bitmap para computação de cubo de dados, assim *também* não é adequada para relações como elevado número de tuplas, alta cardinalidades por dimensão e não permite cálculo de medidas holísticas.

A abordagem *Sampling Cube* (LI; et al., 2008) propõe a computação eficiente de amostras de confiança para qualquer consulta OLAP fazendo agrupamentos que aumentam o tamanho de amostragem quando necessário. Tal abordagem assume que amostras são suficientes e nesta tese destacamos métodos, técnicas e abordagens para conjuntos completos e não apenas amostras. Há clara limitações para se manter tais cubos atualizados, assim como realizar cálculos exatos de medidas holísticas.

3.3 Uso de memória externa na computação de cubo de dados

Top-k H-Cubing (HAN; et al., 2001) foi a abordagem que inicialmente utiliza a memória externa, criando-se projeções para armazenar porções de um cubo de dados em memória externa. Dado um cubo com as dimensões A, B e C, temos as seguintes projeções em memória externa: A $\{a_1, ..., a_n\}$, B $\{b_1, ..., b_m\}$, C $\{c_1, ..., c_z\}$, AB $\{a_1b_1, ..., a_1b_m, ..., a_nb_1, ..., a_nb_m\}$, AC $\{a_1c_1, ...\}$, ABC $\{a_1b_1c_1, ...\}$. Para cada tupla a abordagem Top-k H-Cubing gera a projeção a_1, b_1, c_1, depois as projeções a_1b_1 e b_1, depois as projeções $a_1b_1c_1$, a_1c_1 e c_1 e tais projeções se repetem pelo número tuplas da base de dados. Essa estratégia gera um enorme número de swaps e nem uma ordenação prévia pode resolver. As agregações para cada tupla são geradas usando a abordagem bottom-up apresentada em (BEYER, K; RAMAKRISHNAN, 1999). Infelizmente, a abordagem H-Cubing não foi projetada para cubos de alta dimensionalidade.

3.4 Processamento de medidas holísticas

Alguns trabalhos propõem o processamento de medidas holísticas, incluindo novos algoritmos, estruturas de dados e modelos de cubo de dados (NANDI; et al., 2011; LENZ; THALHEIM, 2001; VASSILIADIS, 1998; GIBBONS; MATIAS, 1998; POOSALA; GANTI, 1999; CHIOU; SIEG, 2001; LI; et al., 2004; LAKSHMANAN; PEI; HAN, 2002; BRAZ; et al., 2007). Estes trabalhos concentram-se em fornecer apenas o cálculo aproximado de medidas holísticas, assim como, apresentam apenas testes com cubos de baixa dimensionalidade e baixo número de tuplas. Não foram encontrados trabalhos que realizem o cálculo exato de medidas holísticas.

Para computar resultados aproximados para consultas de agregação em DWs GIBBONS e MATIAS (1998) introduziram duas novas técnicas baseadas em amostragem, chamadas *Concise Samples* e *Counting Samples*. Os autores apresentam também como manter tais amostras quando existe a inserção de dados no DW. A principal contribuição desse trabalho é a melhoria na precisão dos resultados calculados com base nas novas técnicas de amostragem.

No trabalho de POOSALA e GANTI (1999), é proposto o processamento de consultas holísticas a partir de histogramas utilizados para materializar o cubo de dados, obtendo resultados aproximados com extrema rapidez.

CHIOU e SIEG (2001) realizaram uma melhoria do trabalho de POOSALA e GANTI (1999), e propuseram uma abordagem para otimização no processamento de consultas de agregações multidimensionais com medidas holísticas (*Optimization for Queries with Holistic Functions*), contudo neste trabalho não há experimentos com cubos massivos que comprovem a precisão das medidas calculadas. Esse trabalho sugere uma precisão no cálculo de medidas holísticas melhor que o trabalho POOSALA e GANTI (1999), porém não explora cubos com alta dimensionalidade, elevado número de tuplas e alta cardinalidade.

Em LI, CONG e WANG (2004), são propostos dois algoritmos chamados *Addset Data Structure* e *Sliding Window* que permitem calcular a mediana para agregações em um cubo quociente. Infelizmente, esse trabalho foca apenas em manter um cubo quociente, nenhuma atenção é dada para a melhoria do desempenho das consultas.

BRAZ, et al. (2007) discutem o cálculo de medidas holísticas em cubo de dados, dando destaque a função *holistic aggregate* que retorna o número de trajetórias distintas em uma área

espaço-temporal especifica. A solução proposta é substituir a função holística por duas funções agregadas distributivas que aproximam a função real.

NANDI, et al. (2011) utiliza uma estrutura MapReduce para calcular medidas holísticas em um cubo de dados. A abordagem baseia-se na identificação de subconjuntos de medidas holísticas que são parcialmente algébricos, usando tal propriedade para facilitar o processamento paralelo.

3.5 Resumo

Neste capítulo, apresentamos trabalhos correlatos a nossa pesquisa. Dada a vasta quantidade de trabalhos e pesquisas já realizadas para computação de cubo de dados, focamos em apresentar trabalhos que computam cubos de alta dimensionalidade utilizando as técnicas de índice invertido e bitmap. Apresentamos trabalhos que fazem uso de memória externa e por fim apresentamos pesquisas que propõem o cálculo de medidas holísticas.

A abordagem Frag-Cubing se apresenta como a abordagem mais eficiente para computação de cubos de alta dimensionalidade, além de ser o primeiro a fazer experimentos com cubos de 60 dimensões. Porém, tal abordagem é limitada pela quantidade de memória principal disponível, além de não fazer cálculo de medidas holísticas. O trabalho que propôs o algoritmo de compactação para bitmaps para computação de cubo de dados Word-aligned Hybrid se apresenta como o mais promissor para computar cubos massivos em memória RAM, porém ainda é limitado pela memória RAM, além de não permitir atualizações e não fazer cálculo de medidas holísticas.

Abordagens como *Concise* S*amples*, *Counting Samples*, *Optimization for Queries with Holistic Functions*, *Addset Data Structure* e *Sliding Window* calculam aproximadamente medidas holísticas e não foram projetadas para cubos massivos.

Na Tabela 3.1, resumimos as principais características das abordagens relacionadas. São enfatizados a quantidade de dimensões, número de tuplas e cardinalidade dos cubos de dados que cada abordagem permite computar. Também apresentamos se cada abordagem permite calcular medidas holísticas, fazer atualizações e se utilizam memória externa.

Nos próximos capítulos, apresentamos as abordagens HIC e bCubing que fazem uso otimizado de memória externa para computar cubos de alta dimensionalidade e elevado número de tuplas. Para a abordagem bCubing assumimos a hipótese de calcular e manter atualizado medidas holísticas de forma exata em cubos com alta dimensionalidade e elevado número de tuplas. Experimentos mostram que bCubing calcula medidas holísticas apenas 10% mais lento

quando comparado a cálculos com medidas algébricas e distributivas igualmente calculadas de forma exata. Cenários extremos, como por exemplo, *stream holistic cubes* podem se valer da abordagem bCubing, contudo redesenhada para clusters.

TABELA 3.1 – Principais Características dos Trabalhos Correlatos

	Dimensões	Tuplas	Cardinalidade	Calcula Medidas Holísticas	Realiza Alterações	Memória Externa
Frag-Cubing	>15	10^6	10^4	Não	Sim	Não
Compressed Bitmap Index Based	>15	10^5	10^2	Não	Não	Não
Word-aligned Hybrid	>15	10^6	10^3	Não	Não	Não
BitCube	>15	10^6	10^2	Não	Não	Não
S-OLAP	>15	10^5	10^2	Não	Não	Não
Shell Fragment Mini-Cube	>15	10^6	10^2	Não	Não	Não
Sampling Cube	>15	10^6	10^2	Não	Não	Não
H-Cubing	15	10^5	10^2	Não	Não	Sim
Concise Samples e Counting samples	15	10^5	10^2	Aproximada	Sim	Não
Optimization for Queries with Holistic Functions	>15	10^6	10^2	Aproximada	Não	Não
Addset Data Structure e Sliding Window	15	10^5	10^2	Aproximada	Sim	Não

Para o nosso conhecimento, este é o primeiro trabalho que aborda o cálculo de medidas holísticas exatas em um cubo de dados com algoritmos sequenciais.

4 Abordagens qCube e H-Frag

Neste capítulo, apresentamos as abordagens qCube (SILVA; LIMA; HIRATA, 2013) e H-Frag (SILVA; LIMA; HIRATA, 2015a), contribuições iniciais deste trabalho para a concepção, desenvolvimento e testes das abordagens HIC e bCubing, descritas nos capítulos subsequentes.

4.1 A abordagem qCube

Muitas tarefas de apoio à decisão envolvem consultas de intervalo, como *maior que*, *menor que*, *entre*, *similar*, *distinto*, *alguns*. As abordagens tradicionais para computação de cubo de dados possibilitam apenas consultas utilizando operadores de igualdade e operadores de subcubos combinatoriais.

A abordagem MCG (LIMA; HIRATA 2011) implementa operadores que possibilitam consultas de intervalo, porém tal abordagem não é projetada para cubos de alta-dimensionalidade. As abordagens projetadas para cubos de alta-dimensionalidade, tais como Frag-Cubing (LI; HAN; GONZALEZ, 2004), Compressed Bitmap Index Based (LENG; et al. 2006), WAH (WU, OTOO e SHOSHANI, 2004), BitCube (FERRO; et al. (2009), S-OLAP (LO; et al., 2008), Shell Fragment Mini-Cube (HU, et al. 2005) e Sampling Cube (LI; et al., 2008), são as soluções sequenciais mais promissoras para cubos de dados de alta-dimensionalidade, porém tais abordagens permitem apenas o operador de igualdade em seus diversos tipos de consultas.

A abordagem qCube (SILVA; LIMA; HIRATA, 2013) estende a abordagem Frag-Cubing e permite responder consultas de intervalo de valores. Tais consultas são extensões de consultas clássicas em cubo de dados onde se utiliza apenas o operador de igualdade, seja em consultas pontuais ou consultas denominadas *inquire*.

4.1.1 Estrutura da abordagem qCube

A entrada de dados para computação de cubos na abordagem HIC é uma relação *d*-dimensional *R* com *n* tuplas, onde $n \subset [1, \infty]$. Formalmente, *R* é um conjunto de tuplas, onde cada tupla *t* é definida como $t = (tid, d_1, d_2, ..., d_z, m_1, m_2, ..., m_k)$. Em *t*, o atributo *tid* é um identificador único, assim, em uma relação, não existem tuplas iguais, conforme proposto por

CODD (1972), *z* é o número de dimensões, *d* é uma dimensão específica, definida como $d = (att_1 + att_2 + ... + att_n)$ e *att* é o atributo da dimensão *d*. O número de medidas é representado por *k*, *m* por $m = (mea_1 + mea_2 + ... + mea_n)$, e *mea* é o valor de medida de *m*. O símbolo '+' representa o operador lógico OR.

A arquitetura da abordagem qCube tem dois componentes principais: *qCubeComputation* e *qCubeQuery*. A Figura 4.1 ilustra como os componentes são organizados e como eles se comunicam para realizar a computação, consulta e atualização de cubo de dados.

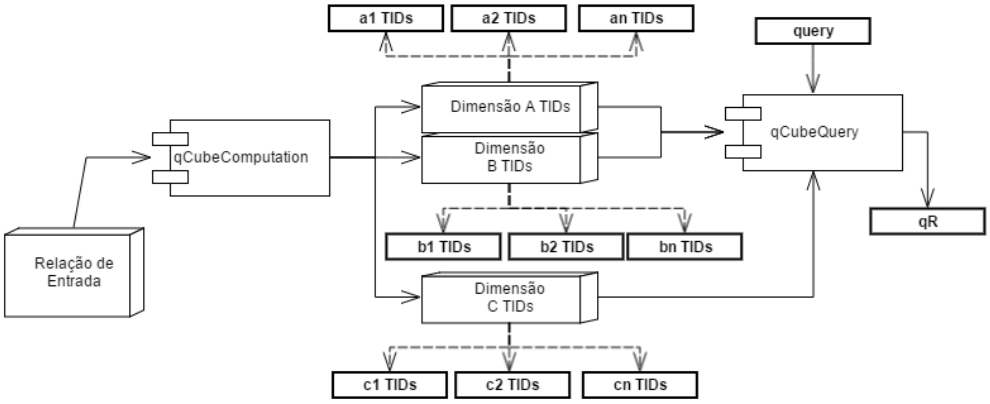

FIGURA 4.1 – Arquitetura da abordagem qCube

O componente *qCubeComputation* é responsável por ler toda a relação de entrada *R* com dimensões D e criar um mapa de identificadores de tuplas (TIDS) por valor de atributo de *R*. Cada mapa tem um valor de atributo como uma chave e uma lista de TIDs como valor. Desta forma, *qCubeComputation* cria todas as tuplas invertidas da abordagem qCube. No exemplo acima, há dimensões A, B e C, cada dimensão tem um conjunto de valores de atributo. A dimensão A tem, por exemplo os valores de atributos: $a_1, a_2, ..., a_n$. O componente *qCubeQuery* recebe uma consulta do usuário, executa operações de interseções e uniões de TIDs gerando um resultado qR. Todos os TIDs gerados a partir de *R* deve ser possível de ser armazenado em memória primária. Consultas pontuais e consultas de subcubos são processadas pelo componente *qCubeQuery*.

4.1.2 Experimentos com qCube

Testes com qCube e a abordagem Frag-Cubing (LI; HAN; GONZALEZ, 2004) variando a quantidade de dimensões do cubo de dados foram realizados, para tais testes foram adotadas relações com D = 20, 30 e 60, $T = 10^7$, S=0 e C = 5000, onde D é o número de dimensões, T é

o número de tuplas em uma relação de entrada, C é a cardinalidade de cada dimensão e S é o skew. Também foi testado as consultas disponibilizadas por ambas abordagens.

O algoritmo da abordagem qCube foi codificado em Java 64 bits (versão 7.0). Utilizamos a implementação da abordagem em C++ do algoritmo da abordagem Frag-Cubing disponibilizada pelos autores e compilada para 64 bits (ILLIMINE, 2015).

Os algoritmos foram implementados apenas em versões sequenciais. Executamos os algoritmos em dois processadores six-core Intel Xeon com 2,4 GHz cada núcleo, cache de 12 MB e 128 GB de RAM DDR3 1333MHz. O disco em que os experimentos foram executados é SAS 15k rpm com 64MB de cache. O sistema operacional é Windows HPC (*High Performance Computing*) Server 2008 versão de 64 bits. Todos os experimentos foram executados cinco vezes, removemos o maior e menor tempo de execução e uma média foi calculada para os três tempos de execução restantes.

Em geral, qCube tem comportamento linear semelhante para computar um cubo de dados quando comparado com Frag-Cubing apensar de ser cerca de 3 vezes mais lento, porém é mais rápido para responder consultas pontuais. A abordagem qCube se mostrou ser mais eficiente, do que a abordagem Frag-Cubing, para responder consultas de subcubo com o operador *inquire*.

Em geral, consultas complexas com 30 operadores, combinando operadores de igualdade, operadores de intervalo e operadores *inquire*, demoraram menos de 10 segundos para serem respondidas por qCube.

Uma consulta em um cubo de dados com D = 60, $T=10^8$ e C = 5000 foi respondida em 2 minutos, para tal consulta foi utilizado: cinco operadores de intervalo, dez operadores de igualdade e um operador *inquire*.

4.2 A abordagem H-Frag

Abordagens que utilizam índices invertidos, como Frag-Cubing e qCube, são consideradas eficientes em termos de tempo de processamento e uso de memória principal para computação e consulta de cubo de dados. Como qualquer outra solução para cubo de dados que utiliza somente memória externa, as abordagens Frag-Cubing e qCube são limitadas pela memória principal disponível, assim, se o tamanho do cubo de exceder a capacidade de memória principal, é necessário o uso de memória externa.

O desafio de usar memória externa é definir critérios para selecionar quais partes do cubo devem estar na memória principal. Neste contexto a abordagem H-Frag (SILVA; LIMA;

HIRATA, 2015a) implementa um sistema de memória híbrida para armazenar partições de cubos de dados na memória externa, onde valores de atributos frequentes são armazenados na memória principal e atributos de baixa frequência são armazenados na memória externa. H-Frag introduz um segundo nível de partição onde um valor de atributo frequente pode ser associado a várias sulistas de TIDs.

Tal como na abordagem qCube a abordagem H-Frag tem como entrada para computação de um cubo de dados uma relação *R*. Dada uma relação *R* como entrada, há o particionamento do cubo de dados em diferentes dimensões e a indexação dos valores de atributos conforme sua frequência, os valores de atributos que possuem maior frequência, em cada dimensão, são armazenados em memória principal e os elementos de menor frequência, de cada dimensão, são armazenados em memória externa. As medidas são sempre armazenadas em memória externa.

4.2.1 Estrutura da abordagem H-Frag

A entrada de dados para a computação do cubo de dados na abordagem H-Frag é uma relação *R* d-dimensional *n* tuplas, onde $n \subset [1, \infty]$, tal como na abordagem qCube.

O processo de computação é iniciado percorrendo *R* por completo para obter a frequência de cada valor de atributo de cada dimensão. Para cada dimensão, é verificado a média da frequência dos atributos e os atributos com a frequência menor do que a frequência média, são marcados à serem armazenados em memória externa.

R é percorrido pela segunda vez para gerar para cada partição de *R* um mapa por dimensão, com somente os valores de atributo marcados para serem armazenados em memória externa. Uma partição de *R* representa o maior percentual possível de tuplas de *R* que pode ser processado na memória principal disponível.

Ao percorrer *R* pela segunda vez, os valores de atributo frequentes não são processados, portanto para cada valor de atributo marcado à ser armazenado em memória externa, cria-se uma estrutura indexada pelo valor de atributo com com a lista de identificadores de tuplas (TIDs) como valor, esta estrutura é a implementação de um mapa.

A lista de TIDs dos valores de atributos, que são selecionados para serem armazenados em memória externa, podem ocupar toda a memória principal rapidamente. Para que isto não ocorra, é possível particionar *R* em partições complementares. Cabe ao usuário final definir o número de partições de *R*. Caso o usuário não defina, assume-se duas porções como limite, sendo que cada partição conterá 50% de tuplas de *R*.

Cada partição deve ser armazenada por completo em memória principal. Contudo, para evitar excessivos acessos a memória externa, para armazenar partições com valores de atributo pouco frequentes, o usuário define um percentual de ocorrência *P* para os valores de atributo das partições. Com isso as ocorrências de um valor de atributo só são armazenadas caso a frequência seja igual a *P*.

Caso a partição seja toda processada e um valor de atributo não igual a *P*, este valor de atributo continua em memória principal e uma nova partição é carregada para ser processada, até a frequência deste valor de atributo ser igual a *P*.

As medidas de todos valores de atributo, também são armazenadas em memória externa, porém a cada TID é armazenado todos valores de medidas associados.

De uma forma geral, mantemos o uso da memória principal sempre alto e adiamos ao máximo o acesso a memória externa para armazenar os valores de atributo. Graças a inversão de índices, um atributo pode ter *n* listas de TIDs complementares e armazenados em memória externa.

Para finalizar a computação do cubo de dados *R* é percorrida pela terceira vez, gerando como saída um novo mapa por dimensão com os valores de atributo mais frequentes de *R*. Tal mapa é mantido em memória principal. Para cenários extremos há possibilidade de armazenar partições complementares dos TIDs de tais valores de atributo em memória externa. A Tabela 4.1 ilustra um exemplo onde há dimensões A, B e C, sendo a dimensão A com cardinalidade 3 e o conjunto de valores $\{a_1, a_2, a_3\}$, a dimensão B tem cardinalidade 3 e o conjunto de valores $\{b_1, b_2, b_3\}$ e a dimensão C com cardinalidade 2 e conjunto de valores $\{c_1, c_2\}$.

TABELA 4.1 – Relação de entrada R

TID	A	B	C	M1	M2
1	a_1	b_1	c_1	1.5	1
2	a_2	b_2	c_2	2.5	1
3	a_2	b_2	c_2	2	3
4	a_3	b_3	c_2	78.5	2
5	a_1	b_1	c_1	100	5
6	a_2	b_1	c_2	102.5	4
7	a_3	b_1	c_1	100	2
8	a_1	b_3	c_2	22.5	3
9	a_1	b_3	c_2	13.89	8

A Tabela 4.1 também apresenta duas medidas (M1, M2). Os identificadores únicos de cada tupla são representados por TIDs.

Para o exemplo apresentado pela Tabela 4.1, primeiramente a frequência de cada valor de atributo de cada dimensão é calculado, sendo o resultado: $fa_1=4$, f-$a_2=3$, $fa_3=2$, $fb_1=4$, $fb_2=2$,

fb_3=3, fc_1=3 e fc_2=6. Com isso os valores de atributo a serem armazenados em memória externa são os com frequência menor do que a frequência média de cada dimensão ou seja 3, já que 3 é a frequência média das dimensões A e B, uma vez que ambas dimensões possuem cardinalidade igual a 3 e o total de tuplas é 9. Na dimensão C a frequência média é 4.5 (dado truncamento com arredondamento para baixo). Com isso os valores de atributos a_3, b_2, b_3 *e* c_1 são marcados para serem armazenados em memória externa.

Percorre-se *R* pela segunda vez, com o objetivo de armazenar em memória externa os valores de atributo identificados anteriormente. Supondo que o usuário defina 3 partições, com 3 tuplas cada uma, à cada dimensão é criada uma estrutura, com as listas de TIDs, indexada por cada valor de atributo identificados para serem armazenados em memória externa. Quando cada lista de TIDs atingir o tamanho em quantidade igual a, para este exemplo vamos considerar 50%, do tamanho de cada partição, esta lista é armazenada em memória externa. Ao final do processamento de todas as partições, todas listas de TIDs ainda em memória principal são armazenadas em memória externa.

Por fim, *R* é percorrido pela última vez e a estrutura de listas de TIDs dos valores de atributo frequentes: a_1, b_1 e c_2, indexadas pelos respectivos valores de atributos é mantido em memória principal.

A Tabela 4.2 apresenta a estrutura de listas de TIDs, indexadas pelos respectivos valores de atributo, armazenadas em memória externa.

TABELA 4.2 – Valores de atributo em memória principal

Valor de Atributo	TIDs
a_1	1,5,8,9
b_1	1,5,6,7
c_2	2,3,4,6,8,9

A Tabela 4.3 ilustra a estrutura de listas, indexadas pelos respectivos valores de atributo, armazenadas em memória externa (cada linha representa um arquivo armazenado).

TABELA 4.3 – Valores de atributo em memória externa

Valor de Atributo	TIDs
a_2	2, 3
a_2	6
a_3	4, 7
b_2	2, 3
b_3	4, 8
b_3	9
c_1	1, 5, 7

A Tabela 4.4 apresenta as medidas do cubo com as tuplas invertidas armazenadas em memória externa, para cada grupo é criado um arquivo com o valor de todas medidas associadas. Cada grupo é identificado pelo primeiro e último TID de cada partição computada.

TABELA 4.4 – Valores de atributo em memória externa

TIDs	M1	M2	Grupo
1	1.5	1	1_3
2	2.5	1	
3	2	3	
4	78.5	2	4_6
5	100	5	
6	102.5	4	
7	100	2	7_9
8	22.5	3	
9	13.89	8	

Dado uma consulta do usuário, o algoritmo de consulta da abordagem H-Frag executa intersecções e uniões entre as listas de TIDs, dos valores de atributo em memória principal e memória externa. A lista de TIDs resultante da consulta é utilizado para obter os valores numéricos das medidas, permitindo que funções estatísticas possam ser calculadas.

4.2.2 Experimentos com H-Frag

Sendo D é o número de dimensões, T é o número de tuplas em uma relação de entrada, C é a cardinalidade de cada dimensão e S é o skew, realizamos testes para computação e consulta de cubos de dados com H-Frag e a abordagem Frag-Cubing (LI; HAN; GONZALEZ, 2004) variando a quantidade de dimensões onde foi utilizado relações com D = 30, 60, 90, 120, 150, 180 e 240, T = 10 M e C = 10^4. Também foi realizado testes com relações variando a quantidade de tuplas com T=1M, 25M, 50M, 75M e 100M, D = 15, C=10^4 e S=0.

O algoritmo da abordagem H-Frag foi codificado em Java 64 bits (versão 8.0). Utilizamos a implementação da abordagem em C++ do algoritmo da abordagem Frag-Cubing disponibilizada pelos autores e compilada para 64 bits (ILLIMINE, 2015).

Os testes variando o número de dimensões para computar os cubos de dados apresentaram consumo de memória linear para ambas as abordagens, porém a abordagem Frag-Cubing utilizou entre 35% a 47% mais de memória do que a abordagem H-Frag, uma vez que na abordagem H-Frag somente os valores de atributo frequentes são mantidos em memória externa.

Os tempos de processamento para os testes de computação dos cubos de dados onde variou-se quantidade de dimensões também foram lineares. De um modo geral, H-Frag foi entre 3,5 e 5 vezes mais lento do que o Frag-Cubing, este desempenho se apresentou como promissor uma vez que H-Frag faz acesso a memória externa.

Os testes variando a quantidade de tuplas teve comportamento linear estável em ambas as abordagens. Em geral, a abordagem do H-Frag tem consumo de memória 45 a 65% menor do que a abordagem Frag-Cubing.

A abordagem Frag-Cubing é mais lenta que a abordagem H-Frag entre 2 e 5 vezes para executar consultas de subcubos combinatoriais, mesmo em cenários onde existem muitos valores de atributos armazenados na memória externa utilizando uma relação de entrada com $T = 10^7$, $C = 10^4$, $D = 30$ e $S = 0$.

Uma relação com $T = 10^9$ tuplas foi computada pela abordagem H-Frag. Este experimento levou 64 horas e consumiu 126 GB de RAM. Os resultados mostram que é possível computar cubos massivos utilizando a abordagem H-Frag permitindo que consultas possam ser realizadas. Consultas com cinco operadores de intervalo, dez operadores de igualdade e um operador *inquire* foram respondidas em 35 segundos. Não é de nosso conhecimento a existência de outra abordagem para computação de cubo de dados sequencial que responda eficientemente consultas em relações de alta-dimensionalidade com $T = 10^9$ tuplas.

4.3 Resumo

Neste capítulo apresentamos as abordagens qCube e H-Frag, ambas abordagens são projetadas para computar cubos de alta-dimensionalidade e H-Frag é uma abordagem que faz uso de memória híbrida (primária e externa), possibilitando a computação de cubos massivos.

A abordagem qCube utiliza a estratégia de índices invertidos da mesma maneira que a abordagem Frag-Cubing. A principal contribuição da abordagem qCube é possibilitar a execução de consultas de intervalo como: *maior que, menor que, entre, similar, distinto, alguns.*

Nossos testes demostraram que qCube é uma solução promissora para consultas complexas com muitos operadores diferentes, incluindo operadores de igualdade, intervalo e operadores de subcubos.

A abordagem qCube tem desempenho para computação de um cubo de dados e desempenho resposta de consultas eficientes, contudo sofre das mesmas limitações impostas por Frag-Cubing já que utiliza índices invertidos e também é projetada apenas para trabalhar somente com memória principal, consequentemente o consumo de memória degrada à medida

que o número de tuplas cresce e as listas de TIDs consequentemente crescem substancialmente, portanto, é necessário propor uma solução eficiente para particionar listas de TIDs que possibilite uma rápida recuperação.

A abordagem H-Frag também é uma extensão da abordagem Frag-Cubing que permite o uso de memória principal e memória externa, assim cubos de dados com 10^9 tuplas podem ser computados. H-Frag usa a seguinte estratégia: valores de atributos com altas frequências são armazenados na memória principal e valores de atributos com baixas frequências são armazenados na memória externa.

Os resultados mostraram que o tempo de execução das abordagens qCube e H-Frag são lineares como o número de dimensões aumenta. Quando comparado com Frag-Cubing, qCube e H-Frag é mais rápido para responder a consultas com operadores de igualdade e operadores de subcubos.

Os experimentos mostram que H-Frag é uma solução eficaz para cubos de dados com elevado número de tuplas. Os resultados demonstram que a H-Frag tem tempo de execução e o consumo de memória linear quando o número de tuplas aumenta. O consumo de memória é sempre menor do que a abordagem Frag-Cubing. Quando comparado com Frag-Cubing, H-Frag tem desempenho semelhante em consultas pontuais, porém possui desempenho superior, sendo até 9 vezes mais rápido, para responder consultas de subcubos combinatoriais. A abordagem de memória híbrido H-Frag é, em média, 3 vezes mais lento do que Frag-Cubing na computação de um cubo, que pode também ser considerado um resultado promissor, uma vez que H-Frag utiliza memória externa.

Além de H-Frag demandar conhecimento prévio do usuário para definição da quantidade de tuplas das partições, ainda é limitado para computar relações com elevado número de tuplas e alto skew, pois pode não ser capaz de armazenar todos TIDs de um valor altamente frequente somente em memória principal. Uma vez que a abordagem H-Frag não possui um mecanismo eficiente de indexação das partições em memória externa, consultas de subcubos que são combinatórias podem não ser executadas, uma vez que é necessário carregar todas sulistas em memória principal para realizar interseções.

Por fim também é almejado que cálculos de medidas holísticas possam ser realizados, seja de forma exata ou aproximada.

5 A abordagem HIC

Neste capítulo, apresentamos a abordagem *Hybrid Inverted Cubing* (HIC) (SILVA; LIMA; HIRATA, 2015b), responsável por um sistema híbrido de memória para armazenar partições de cubos de dados tanto na memória externa quanto na principal. HIC tem como estratégia armazenar em memória principal os valores de atributos frequentes e armazenar valores de atributos pouco frequentes em memória externa.

A entrada de dados para computação de cubos na abordagem HIC é uma relação *d*-dimensional *R* com *n* tuplas, onde *n*. Formalmente, *R* é como definido no capítulo 4 na Subseção 4.1.1.

HIC utiliza um parâmetro, chamado frequência acumulada crítica para definir quais os valores de atributo são armazenados em qual tipo de memória.

A computação na abordagem HIC é composta por três fases. Na primeira fase, a frequência acumulada crítica de cada dimensão e as frequências de cada valor de atributo são determinadas. Na segunda fase, para cada dimensão onde os valores de atributos têm frequência menor ou igual a frequência crítica, o conjunto de identificadores de tuplas (TIDs) é armazenado em memória externa. Na terceira fase, para cada dimensão onde os valores de atributos têm frequência maior do que a frequência crítica, os mesmos são armazenados em memória principal. Os valores de medidas são sempre armazenados em memória externa. Os valores de medida são agrupados em conjuntos de TIDs: cada grupo de valores de medida é identificado por um intervalo de TIDs.

Os resultados apresentados na Seção de experimentos mostram que é possível computar relações massivas que ultrapassam 10^9 tuplas e mais de 60 dimensões numa única máquina e de forma sequencial.

A Tabela 5.1 ilustra uma relação de entrada *R* utilizada como exemplo para descrever a computação de cubo de dados na abordagem HIC. *R* é composta pelos atributos A B, e C e a medida M1.

A seguir são descritas as três fases para computação do cubo de dados pela abordagem HIC:

1ª Fase – Esta fase consiste em determinar as frequências de cada valor de atributo e a frequência acumulada crítica de cada dimensão de *R*. Desta forma, percorre-se *R* por completo. A partir das frequências é possível definir quais valores de atributos terão suas ocorrências armazenadas em memória principal e quais os valores de atributos que terão suas ocorrências

mantidas em memória externa. A ideia é armazenar em memória principal as ocorrências dos valores de atributos mais frequentes cujas frequências sejam maiores do que a frequência acumulada crítica. A definição da frequência acumulada critica depende do tamanho da memória principal e do tamanho do cubo de dados. Na Tabela 5.2 podemos observar as frequências dos valores de atributos de cada dimensão.

TABELA 5.1 – Relação de Entrada R para Computação com HIC

TID	A	B	C	M1
1	a_1	b_1	c_1	2.56
2	a_2	b_2	c_2	3.14
3	a_1	b_3	c_2	2.45
4	a_3	b_1	c_2	6.7
5	a_1	b_1	c_1	88.9
6	a_1	b_1	c_2	1.5
7	a_1	b_2	c_2	3.65
8	a_2	b_2	c_2	14.9
9	a_3	b_3	c_2	75.9
10	a_3	b_1	c_2	76.9
11	a_1	b_1	c_1	65.3
12	a_1	b_3	c_1	44.5

TABELA 5.2 – Frequências dos Atributos na Relação

Dimension	A			B			C	
Valor de Atributo	a_1	a_3	a_2	b_1	b_2	b_3	c_2	c_1
Frequência	7	3	2	6	3	3	8	4
Frequência Acumulada	7	10	12	6	9	12	8	12

As seguintes diretrizes são feitas para determinar quais valores de atributos terão suas ocorrências armazenadas em memória principal:

- Todas as dimensões terão quantidade similar de informações na memória principal. Caso o tamanho da memória principal disponível para armazenamento do cubo seja MMS, então cada dimensão deverá utilizar no máximo MMS/Z desse espaço, sendo Z a quantidade de dimensões. Desta forma, dimensões com baixa cardinalidade têm menos valores de atributos que são armazenados em memória principal.
- Deve-se considerar a estimativa de tamanho da parte do cubo que pode ficar armazenada na memória principal. Seja CS o tamanho do cubo (com suas informações de frequência). Se, por exemplo, metade das ocorrências dos valores de atributos de um cubo puder ficar armazenado na memória principal. Neste sentido, a frequência acumulada crítica deve ser tal que possibilite a seleção dos valores de atributos mais frequentes cujas informações de

ocorrências utilize um espaço menor que o disponível para aquela dimensão, ou seja, CS/2Z.

No nosso exemplo, vamos assumir que CS seja um terço maior que MMS, portanto a memória externa é requerida. O espaço de memória principal disponível para armazenar as informações de ocorrências é o mesmo para cada dimensão, ou seja, MMS/3. Como CS é maior do que MMS, parte das informações de ocorrências dos valores de atributos das dimensões A, B e C são armazenados em memória principal e outra parte em memória externa.

O cálculo das frequências acumuladas ocorre conforme Tabela 5.2. Para a dimensão A, como o valor de atributo a_1 tem a maior frequência de ocorrência (7), o valor de atributo a_1 é a primeira entrada na tabela. A seguir a_3, que tem frequência de ocorrência 3, aparece como segunda entrada. Por fim, a ocorrência de a_2 com frequência 2 é registrada. A terceira linha da tabela ilustra as frequências acumuladas. O acúmulo é feito à medida que a linha é percorrida da esquerda para direita. Por exemplo, para computar a frequência acumulada na coluna de a_3 deve-se acumular as frequências de a_1 e a_3, o que resulta em 10. Para os últimos atributos em cada dimensão a frequência acumulada é igual ao número de tuplas.

Como o tamanho do cubo (CS) é um terço maior do que o tamanho da memória principal (MMS) e o número de tuplas de *R* é 12, então a frequência acumulada crítica é 9 já que MMS é 8. Como resultado desta fase, têm-se identificados dois conjuntos de atributos MMAS (*Main Memory Attribute Set*) e SMAS (*Secondary Memory Attribute Set*). O MMAS é composto por valores de atributos de maior frequência cuja acumulação de frequência resulta em valor menor que a frequência acumulada crítica. No nosso exemplo, $MMAS = \{a_1, b_1, c_2\}$ e $SMAS = \{a_3, a_2, b_2, b_3, c_1\}$.

2ª Fase – Nesta fase, em cada dimensão, os valores de atributos que têm frequência menor do que a frequência critica têm seu respectivo conjunto de identificadores de tuplas (TIDs) armazenado em memória externa.

Contudo, para evitar gravações em memória externa de valores de atributos muito pouco frequentes, gerando um elevado número de acessos a memória externa e também evitar a falta de memória de trabalho, o algoritmo HIC implementa uma estratégia de buffer. A estratégia de buffer estabelece o armazenamento em memória externa da maior lista de TIDs, dentre os valores de atributo já processados, quando o tamanho total do conjunto de listas de TIDs já processadas atingir 80% (valor configurável) do tamanho de MMS, assim é possível obter um equilíbrio no tamanho das listas de TIDs complementares de cada valor de atributo armazenadas em memória externa.

Ao final, todas as listas de TIDs dos valores de atributos presentes em SMAS, independentemente do tamanho em relação a CS, são armazenadas em memoria externa. Os valores de medidas de cada tupla de *R* são armazenadas em memoria externa sempre que os valores de uma determinada medida ocupar o equivalente a 80% do tamanho de MMS.

Graças a inversão de índices, um valor de atributo pode ter *n* sulistas de TIDs e armazenadas em memória secundária. Como resultado desta fase, têm-se todas as informações de ocorrências (lista de TIDs) dos valores de atributos presentes em SMAS armazenadas em memória externa. O resultado da computação parcial do cubo em memória externa é apresentado nas Tabelas 5.3 e 5.4. A Tabela 5.3 ilustra todos os valores de atributos armazenados em memória externa onde cada linha representa um arquivo armazenado em um diretório especifico que representa uma determinada dimensão.

TABELA 5.3 – Valores de Atributo pouco Frequentes em Memória Externa

Dimensão	Valor de Atributo	TIDs
A	a_2	2, 8
	a_3	4, 9, 10
B	b_2	2, 7, 8
	b_3	3, 9, 12
C	c_1	1, 5, 11, 12

TABELA 5.4 – Valores de Medidas em Memória Externa

TID	M1	Grupo
1	2.56	1_6
2	3.14	
3	2.45	
4	6.7	
5	88.9	
6	1.5	
7	3.65	7_12
8	14.9	
9	75.9	
10	76.9	
11	65.3	
12	44.5	

A Tabela 5.4 apresenta as medidas do cubo de dados com as tuplas invertidas armazenadas em memória externa. Para cada grupo de valores de medida, um arquivo é criado com o valor de todas as medidas associadas com todas as tuplas processadas a cada armazenamento realizado. Neste exemplo, como o CS é um terço maior que MMS e CS tem 12 tuplas, MMS tem tamanho 8, a porcentagem 80% de MMS é 6,4, portanto cada grupo de valores de medida contem 6 tuplas (dado truncamento com arredondamento para baixo).

3ª Fase - Na terceira fase, em cada dimensão os valores de atributos que têm as maiores frequências e, de forma acumulada, seja menor do que a frequência acumulada crítica

(pertencentes ao MMAS), têm suas listas de TIDs armazenadas em memória principal. Nesta fase percorre-se *R* novamente, contudo, segundo os experimentos realizados com relações com diferentes quantidades de tuplas e dimensões, a fase 3 é rápida, pois não há instruções para gravação em memória externa.

Como resultado desta fase, têm-se todas as informações de ocorrências dos valores de atributos (lista de TIDs) presentes em MMAS armazenadas em memória principal e prontos para a realização de consultas. No nosso exemplo, $MMAS = \{a_1, b_1, c_2\}$ e o resultado da computação parcial do cubo em memória principal é apresentado na Tabela 5.5.

TABELA 5.5 – Valores de Atributo Frequentes em Memória Principal

Dimensão	Valor de Atributo	TIDs
A	a_1	1,3,5,6,7,11,12
B	b_1	1,4,5,6,10,11
C	c_2	2,3,4,6,7,8,9,10

Dado uma consulta do usuário, é executado interseções e uniões com TIDs dos conjuntos em memória principal. Após obter os TIDs dos atributos frequentes, processam-se os atributos da consulta que estão armazenados em memória externa. A lista de TIDs resultante da consulta é utilizado para obter os valores numéricos das medidas.

5.1 O Algoritmo de computação HIC

A partir de um algoritmo para computação de cubos HIC denominado *CO* e uma relação de entrada *R*, obtém-se um cubo de dados $HIC=(\{iTati_1, iTati_2 \ldots iTati_n\}, \{[iEati_1, iEati_2 \ldots iEati_n], [im_1, im_2 \ldots im_x]\})$, onde cada elemento interno *iTati* e *iEati* representa um conjunto de tuplas invertidas de uma dimensão específica, representada por att_i, sendo *iTati* em memória principal e *iEati* em memória externa. Cada *iTat* e *iEat*, são definidos como $(tid_1, \ldots, tid_p)$, representando os identificadores de tuplas (TIDs) associados a um atributo att_i de *R*, cujo tamanho máximo é *p*. A abordagem HIC também possui em memória externa $(im_1, im_2, \ldots, im_x)$, onde cada *im* é definida como $im = (tid, mv)$, sendo também o índice invertido para uma medida, composta por um TID e um valor de medida *mv* numérico. O algoritmo CO é apresentado no Algoritmo 5.1.

Algoritmo 5.1: (Computação (CO)) computação de cubo de dados com conjuntos de tuplas indexados por atributos.

Entrada: *R* com um conjunto de tuplas *t*, é definido como $t = (tid, d_1, d_2, \ldots, d_n)$.

Saída: Um cubo de dados HIC.

Método:

```
1. MMS <- get memory available
2. op <- MMS*0.8

3. while R has tuples{
4.       check the frequency of attribute values in each dimension
5.       for each dimension in tuple{
6.           atts{di, {att, count}} <- tuple.attributeValue and count the attribute
             value in dimension
         }
   }
7. CS <- calculate cube size

8. if CS > MMS {
9.     CF <- MMS+1 //cumulative frequency critical
   }
10. for each dimension in tuple {
       i <- 0;
       for each att value in atts{
11.           if att.count >= CF{ //SMAS  <- with frequency cumulative critical
                     SMAS[i] <- attribute value
              }else { //MAVS <- with cumulative frequency less than critical frequency
12.                  MAVS[i] <- attribute.value
              }
13.           i <- i+1;
       }
    }
14. while R has tuple{
15.    if SMAS contains attribute value{
16.           build inverted index in an entry in hicDiT: {att, tids}
17.           if the size hicDiT is OP compared to CS {
18.                  storage hicDiT.att
19.                  hicDiT.att
              }
       }
20.    for each measure value in tuple {
21.           build inverted index in an entry in hicDiM: {mi, {tid, mv}}
22.           if the size the measure values is OP compared to CS{
23.                  storage hicDiM
24.                  hicDiM <- null

              }
       }
25.    storage hicDiT and hicDiM
   }
26. while R has tuple {
27.    if MAVS contains attribute values{
28.           build inverted index in an entry in hicMiT: {att, tids}} //in main memory
       }
    }
```

O algoritmo CO recupera o tamanho da memória RAM disponível na linha 1 e atribui para a variável `MMS`. A variável auxiliar `OP` na linha 2, representa 80% da memória RAM disponível, para controle do sistema de buffer.

São verificadas as frequências de todos valores de atributos de cada dimensão de *R* e armazenadas na variável `atts` (linhas 3-6). O tamanho do cubo de dados é calculado após todas tuplas serem percorridas (linha 7). Em seguida, a frequência acumulada crítica é calculada e atribuída à variável `CF` (linha 9), o tamanho de `MMS` estabelece o limite de tuplas em memória principal, então `CF` será `MMS+1`. Os valores de atributos a serem armazenados na memória externa e na memória principal são definidos; tais valores de atributos são atribuídos para as variáveis SMAS e MMAS respectivamente (linhas 10 e 13).

Para cada valor de atributo de *R* (linha 15), é construído a inversão de índices para cada atributo na variável `hicDiT` (linha 16), quando o tamanho das listas de TIDs dos valores de atributo é igual a variável auxiliar `OP`, a maior lista de TIDs é armazenada na memória externa (linhas 17 e 18) e `hicDiT` é reinicializada na linha 19. Quando a quantidade de valores de medida de *R* processadas (armazenados na variável `hicDiM`) for igual a `OP` (linhas 20 e 23) `hicDiM` é armazenada na memória externa e depois reinicializada na linha 24. Na linha 25, os valores de atributos e os valores de medidas são armazenados em memória externa (variáveis `hicDiT` e `hicDiM`). Finalmente, HIC armazena em memória externa a lista TID de valores de atributos presentes na variável MMAS que permanece na memória RAM (linhas 27 e 28).

5.2 O Algoritmo de atualização HIC

Quatro tipos de atualizações podem ocorrer em um cubo de dados, são estas: (I) uma nova tupla ser adicionada a *R*; (II) atributos de *R* podem ser fundidos; (III) novas dimensões e novas medidas podem ser adicionadas a *R*; (IV) hierarquias de uma dimensão podem ser reorganizadas. Inversão de índices se apresenta como uma excelente estratégia para tais tipos de atualizações. O algoritmo de computação é utilizado para atualizações do tipo (I), desta maneira a ocorrência dos novos valores de atributos são armazenadas em memória externa e memória principal conforme a computação do cubo de dados. Caso as ocorrências dos valores de atributos em memória principal já ocupe toda memória principal disponível, o valor de atributo menos frequente em memória principal é armazenado em memória externa para liberar espaço. Se em memória principal estiver armazenado somente as ocorrências do valor de atributo atualizado, sublistas dos novos TIDs das atualizações deste valor de atributo são armazenadas em memória externa.

Exemplo atualização I: adicionamos três novas tuplas onde a primeira tupla (tid_{13}) possui novos valores de atributo, a segunda tupla (tid_{14}) tem valores de atributos já existentes e que estão armazenados na memória externa e a terceira tupla (tid_{15}) é formada por valores de atributo existentes no cubo de dados que estão armazenados na memória principal, como ilustra a Tabela 5.6.

A relação de atualização com três novas tuplas é percorrida e para cada valor de atributo de cada tupla, HIC verifica se o mesmo já existe no cubo de dados. Caso um valor de atributo já exista, é verificado se tal valor de atributo está armazenado em memória principal ou memória externa.

TABELA 5.6 – Relação com novas tuplas para atualização.

TID	A	B	C	M1
13	a_4	b_4	c_4	3
14	a_3	b_3	c_1	4.7
15	a_1	b_1	c_2	5.5

Para cada valor de atributo armazenado na memória principal é verificado se ainda há espaço suficiente para o armazenamento da lista de TIDs complementar do respectivo valor de atributo. Caso não exista espaço suficiente disponível na memória principal, a lista de TIDs complementar é armazenada em memória externa. A atualização de dados após a inserção das três novas tuplas é ilustrada nas Tabelas 5.7, 5.8 e 5.9.

TABELA 5.7 – Valores de Atributo em Memória Externa Após a Atualização I.

Dimensão	Valor de Atributo	TIDs
A	a_2	2, 8
	a_3	4, 9, 10
	a_4	13
	a_1	14
B	b_2	2, 7, 8
	b_3	3, 9, 10
	b_4	13
	b_3	14
C	c_1	1, 5, 11, 12
	c_4	13
	c_1	14

TABELA 5.8 – Valores de Atributo em Memória Principal Após a Atualização I

Dimensão	Valor de Atributo	TIDs
A	a_1	1,3,5,6,7,11,12,15
B	b_1	1,4,5,6,10,11,15
C	c_2	2,3,4,6,7,8,9,10,15

TABELA 5.9 – Valores de Medidas em Memória Externa Após a Atualização I

TID	M1	Grupo
1	2.56	1_4
2	3.14	
3	2.45	
4	6.7	
5	88.9	5_8
6	1.5	
7	3.65	
8	14.9	
9	75.9	9_12
10	76.9	
11	65.3	
12	44.5	
13	3	13_15
14	4.7	
15	5.5	

Para atualizações do tipo (II), quando os valores de atributo a serem fundidos estiverem em memória externa, somente é alterada a indexação da lista de TIDs dos valores de atributos fundidos para o novo valor de atributo.

De uma forma geral, atualizações do tipo (II) são triviais.

Exemplo atualização II: suponha que os valores de atributo a_2 e a_3 sejam fundidos em um novo valor de atributo a_9. A Tabela 5.10 ilustra listas de TIDs de a_2 e a_3 reunidas em a_9.

Caso o valor do atributo a_9 passe a ser mais frequente do cubo de dados, então há substituição do valor do atributo a_1 na memória principal. Desta maneira, a lista de TIDs do valor de atributo a_1 passa a ser armazenado em memória externa e a_9 em memória principal.

TABELA 5.10 – Valores de Atributo em Memória Externa Após a Atualização II

Dimensão	Valor de Atributo	TIDs
A	a_9	2, 8
	a_9	4, 9, 10
B	b_2	2, 7, 8
	b_3	3, 9, 12
C	c_1	1, 5, 11, 12

TABELA 5.11 – Relação R atualizada: nova dimensão D e nova medida M2

TID	A	B	C	D	M1	M2
1	a_1	b_1	c_1	d_1	2.56	1
2	a_2	b_2	c_2	d_1	3.14	1
3	a_1	b_3	c_2	d_1	2.45	3
4	a_3	b_1	c_2	d_1	6.7	2
5	a_1	b_1	c_1	d_3	88.9	5
6	a_1	b_1	c_2	d_2	1.5	4
7	a_1	b_2	c_2	d_1	3.65	2
8	a_2	b_2	c_2	d_3	14.9	3
9	a_3	b_3	c_2	d_2	75.9	8
10	a_3	b_1	c_2	d_1	76.9	2
11	a_1	b_1	c_1	d_1	65.3	11
12	a_1	b_3	c_1	d_1	44.5	6

Atualizações do tipo (III) requerem que a nova dimensão ou medida seja percorrida para que seus atributos sejam associados aos TIDs já contidos no cubo computado. Para isto, também é executado o algoritmo de computação, porém nesse caso *R* é entrada com apenas a nova dimensão.

Exemplo atualização III: adicionamos à relação *R* uma nova dimensão D e uma nova medida M2, conforme Tabela 5.11.

O algoritmo de computação é executado apenas para a dimensão D e a medida M2, desta maneira, apenas os novos valores de atributo e medida são inseridos com as respectivas listas de TIDs, de acordo com a Tabelas 5.12, 5.13 e 5.14.

TABELA 5.12 – Valores de Atributo em Memória Externa após a Atualização III

Dimensão	Valor de Atributo	TIDs
A	a_2	2, 8
	a_3	4, 9, 10
B	b_2	2, 7, 8
	b_3	3, 9, 12
C	c_1	1, 5, 11, 12
D	d_3	5, 8
	d_2	6, 9

TABELA 5.13 – Valores de Atributo em Memória Principal após Atualização III

Dimensão	Valor de Atributo	TIDs
A	a_1	1,3,5,6,7,11,12
B	b_1	1,4,5,6,10,11
C	c_2	2,3,4,6,7,8,9,10
D	d_1	1,2,3,4,7,10, 11, 12

TABELA 5.14 – Valores de Medidas em Memória Externa após a Atualização III

TID	M1	M2	Grupo
1	2.56	1	1_6
2	3.14	1	
3	2.45	3	
4	6.7	2	
5	88.9	5	
6	1.5	4	
7	3.65	2	7_12
8	14.9	3	
9	75.9	8	
10	76.9	2	
11	65.3	11	
12	44.5	6	

Atualização do tipo (IV) não gera impacto no cubo de dados, já que uma consulta pode proceder em qualquer ordem. Para a abordagem HIC não há diferença se uma consulta utiliza os atributos ABC ou CBA de R.

5.3 O algoritmo de consulta HIC

A partir de um cubo HIC pode-se realizar consultas *Q*, que geram como saída vários sub-conjuntos de TIDs, derivados de dois possíveis tipos de consultas: (1) consulta pontual e (2) consultas com múltiplas agregações. Uma consulta pontual é realizada quando se utiliza um filtro com operador de igualdade e consultas que têm como resultado múltiplas agregações são aquelas onde filtros de intervalo ou filtros inquire são utilizados.

Filtros com diferentes operadores podem ser utilizados em *Q*. Cada dimensão de *R* pode conter um ou vários filtros. Assim, três possíveis sub-consultas são geradas a partir de *Q: pQ* (consultas com filtros de igualdade), *rQ* (consultas com filtros de intervalo) e *iQ* (consultas

com filtros subcubo). Um único resultado *Q* consolida os resultados das três possíveis sub-consultas através de algoritmos de interseção de conjuntos com complexidade *O(n)*, onde *n* é o número de elementos do menor conjunto. Para cada tipo de consulta a abordagem HIC possibilita a obtenção das medidas devidamente agregadas. Assumimos $pQ, rQ, iQ \in Q$.

Consultas *pQ* geram resultados pontuais com uma única agregação de um conjunto de valores de atributos de *R*. Uma consulta *rQ* pode ter como resultado um conjunto de agregações a partir dos valores de atributos presentes em *R*. Uma sub-consulta inquire *iQ* tem como resultado a combinação de diferentes dimensões. *iQ* representa consultas *inquire*, onde dois operadores *iOp* (subcube + distinct) são definidos para diferentes dimensões. O operador de intervalo *rOp* é definido como $rOp = (maior\ que + menor\ que + entre + alguns + diferente + similar\ x\ (v_1, v_2, ..., v_n))$. O símbolo *'+'* representa o operador lógico OR e *'x'* representa o operador lógico AND. Os valores definidos pelo usuário para um determinado operador de intervalo são representados por $(v_1, v_2, ..., v_n)$.

A partir de um cubo HIC, um filtro *F* é executado em uma consulta *pQ*. Sendo *F* definido como $F: \{op_1 \cap op_2 \cap ... \cap op_n\}$, onde op_i é o operador i^{th} *EQUAL* de *F* aplicada a dimensão *i* de HIC. Já em uma consulta *rQ* a partir de um cubo de dados, HIC executa um filtro *F´*. Desta maneira TIDs de *pQ* são interseccionados com TIDs de *rQ*. A definição de *F´* é dada por $F': \{œ_1 \cap œ_2 \cap ... \cap œ_n\}$, onde $œ_i$ é o operador i^{th} RANGE de *F´* aplicada a dimensão *i* de HIC. *F* e *F´* são filtros aplicados à diferentes dimensões, pois os filtros de igualdade requerem isto. Cada $œ_i$ retorna um conjunto de TIDs para os valores de atributo que atendem o critério definido por um operador *rOp*. Com isso, é executado um grupo de intersecção dos conjuntos de TIDs para cada possível associação entre valores de atributos instanciados em cada sub-consulta, sendo estas interseções sempre iniciadas a partir dos valores de atributos com menor conjunto de TIDs.

Consultas *iQ* são também combinatórias, portanto uma consulta *iQ* inquire recebe um cubo de dados, HIC, e executa um terceiro filtro *F´´*. O TIDs de *iQ* são interseccionados com TIDs resultantes de $(pQ \cap rQ)$. O filtro *F´´* é definido como $F'': \{ı_1 \cap ı_2 \cap ... \cap ı_n\}$, onde $ı_i$ é o operador i^{th} INQUIRE de *F´´* aplicado a dimensão *i* de HIC. *F*, *F´* e *F´´* são filtros aplicados a diferentes dimensões, pois filtros de igualdade requerem isto.

As primeiras sub-consultas processadas são sempre as consultas pontuais. Em seguida, passa-se a processar as consultas de intervalo e inquire de *Q*. A cada sub-consulta, é recuperado o conjunto de TIDs.

Quando os valores de atributos encontram-se em memória principal, ou seja, quando tratam-se de valores de atributos frequentes na dimensão, este conjunto é recuperado em um

único acesso. Já quando os valores de atributos têm menor frequência na dimensão, seu conjunto de TIDs é recuperado da memória externa. Neste caso, como tal conjunto pode estar fragmentado em memória externa, reduzimos o custo das interseções ao começar a leitura dos fragmentos do último fragmento armazenado para o primeiro. Desta forma, assumimos que o último fragmento pode possuir menos tuplas do que os demais e, consequentemente, um conjunto de TIDs menor.

Exemplo de uma consulta com inquire: Suponha que uma consulta do usuário final $Q=\{?,?, c_2\}$. Inicialmente, HIC recupera a lista de TIDs da dimensão instanciada com o parâmetro c_2 e que tem como retorno $c_2:\{2,3,4,6,7,8,9,10\}$. Em seguida, HIC recupera as listas de TIDs de todos valores de atributos das dimensões A e B. Com isso obtêm-se A = [($a_1:\{1,3,5,6,7,11,12\}$), ($a_2:\{2,8\}$) , ($a_3:\{4,9,10\}$)] e B = [($b_1:\{1,4,5,6,10,11\}$), ($b_2:\{2,7,8\}$), ($b_3:\{3,9,12\}$)]. Em seguida são executadas interseções entre a lista de TIDs de c_2 e as listas de TIDs dos valores de atributo da dimensão A. Os resultados são os seguintes: $a_1 \cap c_2:\{3,6,7\}$, $a_2 \cap c_2:\{2,8\}$, $a_3 \cap c_2:\{4,9,10\}$.

HIC também executa as interseções entre a lista de TID de c_2 e todas listas de TIDs dos valores de atributo da dimensão B, com isso é obtido o seguinte resultado: $b_1 \cap c_2:\{4,6,10\}$, $b_2 \cap c_2:\{2,7,8\}$ e $b_3 \cap c_2:\{3,9\}$. Um conjunto final de interseções é feito entre as listas de TIDs de todos os valores de atributo da dimensão A e B, com a lista de TIDs de c_2, assim temos $a_1 \cap b_1 \cap c_2:\{6\}$, $a_1 \cap b_2 \cap c_2:\{7\}$, $a_1 \cap b_3 \cap c_2:\{3\}$, $a_2 \cap b_1 \cap c_2:\{\}$, $a_2 \cap b_2 \cap c_2:\{2,8\}$, $a_2 \cap b_3 \cap c_2:\{\}$, $a_3 \cap b_1 \cap c_2:\{4,10\}$, $a_3 \cap b_2 \cap c_2:\{\}$ e $a_3 \cap b_3 \cap c_2:\{9\}$. Assim o resultado final da consulta $Q=\{?,?, c_2\}$ são as agregações apresentadas na Tabela 5.15 com as respectivas frequências.

No Algoritmo 5.2 apresentamos como é realizado consultas pontuais, consultas de intervalo e consultas *inquire*.

TABELA 5.15 – Resultado da consulta $Q=\{?,?, c_2\}$ processada com a abordagem HIC

Agregação	**Frequência**
$*, *, c_2$	8
$a_1, *, c_2$	3
$a_2, *, c_2$	2
$a_3, *, c_2$	3
$*, b_1, c_2$	4
$*, b_2, c_2$	3
$*, b_3, c_2$	2
a_1, b_1, c_2	1
a_1, b_2, c_2	1
a_1, b_3, c_2	3
a_2, b_2, c_2	2
a_3, b_1, c_2	2
a_3, b_3, c_2	1

Algoritmo 5.2 (**Consulta**) processamento de consultas pontuais, consultas de intervalo e consultas inquire.

Entrada: (1) Um cubo de dados HIC; e (2) uma consulta de usuário *Q*.

Saída: HIC_R que inclui as sumarizações processadas pelo algoritmo de consulta e finalizadas pelo algoritmo de consulta.

Método:

```
1. for each sub-query in Q { //pQ, rQ or iQ
2.        for each attribute in Di{
3.            if attsInExternalMemory contains attribute{
4.                   attribute.tids recover externalMemory //first is recovery the
                     last file created
5.                   tids ← tids U attribute.tids
6.            }else{
7.                  tids ← attribute.tids
              }
8.            for each tid in tids {
9.                  if tidi ∩ [att1, …, attn] //point query and range query
10.                       RQi ← tidi  ∩ [att1, …, attn]
                    }
11.           if tidi ∩ [att1, att2, …, attn]{ //inquire query and
12.                       IQi ← tidi  ∩ [att1, …, attn]
                    }
              }
          }
13.       hicQr ← RQi ∩ IQi;
      }
14. hicQr ← calculateMeasures(hicQr, Q, hicDiM);
```

Inicialmente, para cada sub-consulta recupera-se os TIDs (linha 4) associados aos atributos instanciados em cada dimensão. No caso de um valor de atributo estar em memória externa, é recuperado um fragmento por vez, iniciando pelo último conforme já explicado. Após a interseção, os conjuntos são unidos (linha 5) até que ocorra intersecções com todos TIDs em memória externa. Em seguida, é realizado a interseção entre as listas de TIDs dos atributos de cada possível agregação. Sempre a interseção é realizada a partir das listas com menor número de TIDs (linhas 8-12), desta forma obtemos os TIDs correspondentes as agregações do conjunto de tuplas que forma o resultado da consulta realizada. Por fim, há o cálculo de todas medidas definidas em *Q* (linha 14).

5.4 Experimentos

Almejando verificar a eficiência e a escalabilidade da abordagem proposta, um estudo detalhado foi feito. Foram realizados experimentos com as abordagens HIC, H-Frag (SILVA; LIMA; HIRATA, 2015a) e Frag-Cubing (LI; HAN; GONZALEZ, 2004), testando os algoritmos de computação e as consultas disponibilizadas por ambas abordagens. Os algoritmos das abordagens HIC e H-Frag foram codificados em Java 64 bits (versão 8.0). No caso do Frag-

Cubing, foi utilizado a implementação em C++ disponibilizada pelos autores e compilada para 64 bits (ILLIMINE, 2015).

Executamos experimentos com relações que cabem em memória principal e também relações que necessitam de memória externa. Os testes de computação do cubo de dados incluem tempo para operações de acesso a memória e tempo de CPU. Tempos de acesso a memória externa são considerados para carregar as relações de entrada da memória externa para a memória principal. Para as abordagens HIC e H-Frag, nos testes de consulta, foi considerado o tempo de acesso a porções do cubo em memória externa. Todos os experimentos não ultrapassaram o limite físico de memória principal da máquina, portanto as abordagens não requereram swap do sistema operacional.

Os algoritmos foram implementados apenas em versões sequenciais. Executamos os algoritmos em dois processadores six-core Intel Xeon com 2,4 GHz cada núcleo, cache de 12 MB e 128 GB de RAM DDR3 1333MHz. O disco em que os experimentos foram executados é SAS 15k rpm com 64MB de cache. O sistema operacional é Windows HPC (*High Performance Computing*) Server 2008 versão de 64 bits. Todos os experimentos foram executados cinco vezes, removemos o maior e menor tempo de execução e uma média foi calculada para os três tempos de execução restantes.

Para esta Seção, definimos que D é o número de dimensões, C é a cardinalidade de cada dimensão, T é o número de tuplas em uma relação de entrada e S é o skew de dados. Quando S é igual a 0, os dados estão distribuídos uniformemente; à medida que S aumenta, os dados tornam-se mais *skewed*. Bases de dados reais são normalmente *skewed*. As relações de base sintética foram criadas usando gerador de relações fornecido pelo projeto IlliMine. O projeto IlliMine é um projeto *open-source* para fornecer várias abordagens para mineração de dados e aprendizado de máquina. A abordagem Frag-Cubing faz parte do projeto IlliMine.

5.4.1 Computação de relações com diferentes números de tuplas

Os testes variando quantidade de tuplas tiveram comportamento linear e estável nas três abordagens. Foram utilizadas relações com T = 1M, 25M, 50M, 75M e 100M, D = 60, C = 10^4, e S = 0. Em geral HIC utilizou de duas a três vezes menos memória RAM do que Frag-Cubing e precisou de 20% a 80% menos de memória do que H-Frag, conforme pode ser observado na Figura 5.1.

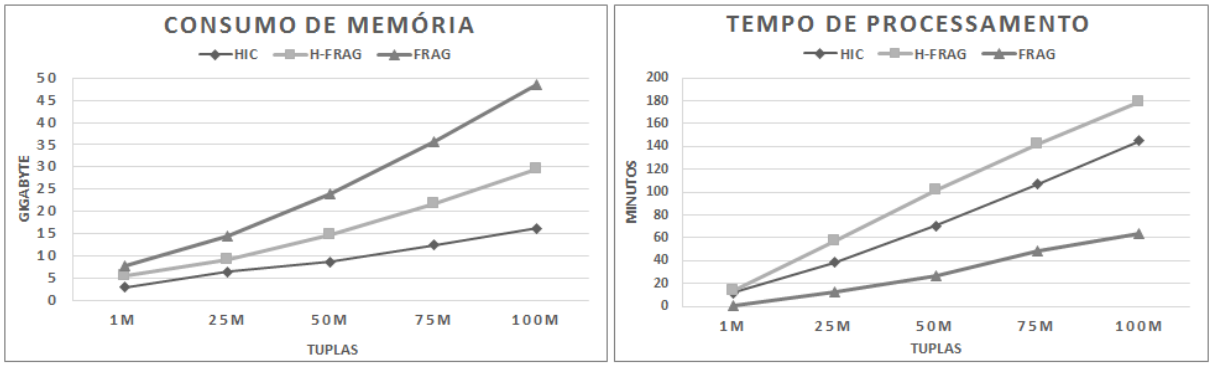

FIGURA 5.1 – Tempo de processamento e consumo de memória para relações com diferentes números de tuplas com as abordagens HIC, H-Frag e Frag-Cubing: D = 60, S = 0, C = 10^4

Os tempos de execução HIC para computar cubos de dados são, em média, três a cinco vezes mais lentos do que Frag-Cubing; no entanto, este é um resultado promissor considerando o armazenamento externo feito por HIC. HIC é, em média, 1.5 vezes mais rápido do que o H-Frag. HIC é mais rápido do que H-Frag já que adia o acesso à memória externa até que a frequência acumulada critica seja atingida, estratégia esta que se mostra mais eficiente do que utilizar o percentual fixo de 50% de uma partição da relação, conforme proposto por H-Frag.

5.4.2 Computação de relações com diferentes números de dimensões

Na Figura 5.2, é apresentado o resultado dos experimentos variando a quantidade de dimensões do cubo de dados. Foram adotadas nos experimentos relações com D = 240, 480, 720, e 960, T = 10^7, e C = 10^4.

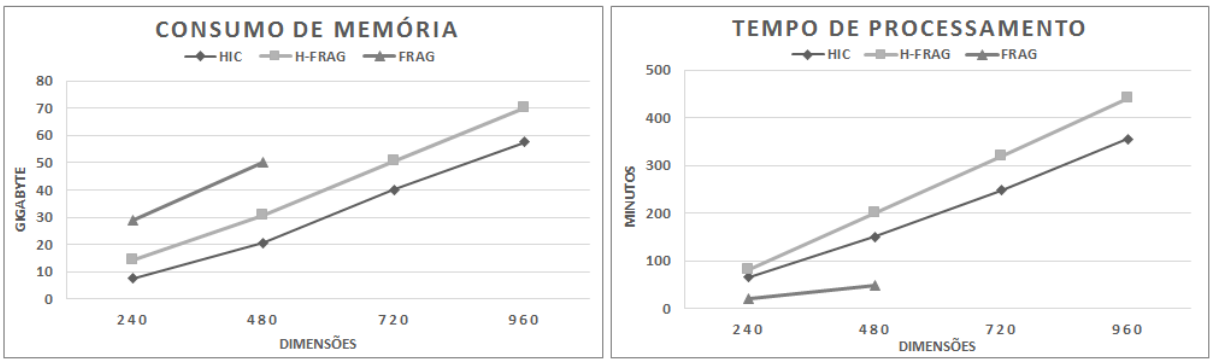

FIGURA 5.2 – Tempo de processamento e consumo de memória para relações com diferentes números de dimensões com as abordagens HIC, H-Frag e Frag-Cubing: D = 15, S = 0, C = 10^4

O consumo de memória foi linear para todas as abordagens; no entanto, uma vez que a abordagem Frag-Cubing faz alocação de memória contígua, esta necessita de cerca de 50% mais memória, portanto as relações com D = 720 e 960 não foram computadas por Frag-Cubing.

Durante os experimentos constatamos que HIC, quando comparado com abordagem Frag-Cubing, necessitou em média de duas a três vezes menos memória para computar cubos de dados de alta dimensionalidade, HIC consome 20% a 50% menos memória do que o H-Frag para computar os mesmos cubos de dados. Como pode ser observado na Figura 5.2, a abordagem Frag-Cubing não computa relações com mais do que 480 dimensões e 10^7 tuplas. Estes resultados mostram que de HIC e H-Frag superam a limitação de computação de cubos de dados com elevada dimensionalidade da abordagem Frag-Cubing. Por fim, HIC é de 23% a 41% mais rápido do que o H-Frag.

5.4.3 Computação de relações de diferentes skew

Nesta Seção a uniformidade dos dados foi testada. Foram avaliadas relações com diferentes skew: S = 0; 0,5; 1; 1,5; 2 e 2,5, D = 15, T = 10^7, e C = 10^4.

A Figura 5.3 ilustra o consumo de memória e tempo de execução para cada relação computada.

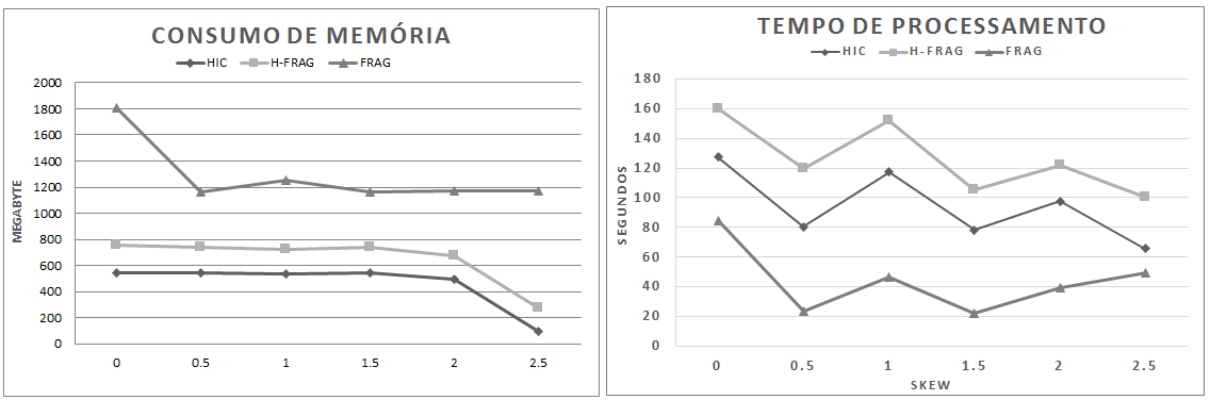

FIGURA 5.3 – Tempo de processamento e consumo de memória para relações com diferentes skew com as abordagens HIC, H-Frag e Frag-Cubing: D = 15, T = 10^7, C = 10^4

Observa-se na Figura 5.3 que todas abordagens mostram o mesmo comportamento, ou seja, quando há o aumento do skew a computação do cubo é mais rápida. Isto faz sentido, uma vez que é necessário a alocação de menos listas de TIDs, com isso as três abordagens são beneficiadas. As abordagens HIC e H-Frag com menos listas de TIDs fazem menos operações de acesso a memória externa, já que em uma relação com alto skew, em geral, a cardinalidade é reduzida, assim existem menos valores de atributos pouco frequentes. A abordagem HIC leva de 1,6 a 3 vezes mais tempo do que a abordagem Frag-Cubing, porém é entre 25% e 52% mais rápida do que H-Frag.

Relações *skewed* onde determinados valores de atributos estão presentes em quase todas as tuplas são comuns em aplicações reais. Como HIC armazena valores de atributos frequentes

na memória principal, este consome mais memória principal e menos operações de acesso a memória externa, tem um desempenho melhor em relações com skew alto do que relações com skew mais baixo.

O melhor desempenho de HIC em relação a H-Frag ocorre devido a estratégia da frequência acumulada critica ser mais eficiente do que o sistema de buffer utilizado por H-Frag, pois H-Frag sempre adota 50% de uma partição do cubo de dados (definida pelo usuário) para realizar uma operação de entrada e saída. Assim, como relações *skewed* podem ter valores de atributo com frequência próximos a 100% em uma partição do cubo de dados a abordagem H-Frag realiza operações de entrada e saída diversas vezes.

Em todos os cenários HIC e H-Frag reduziram significativamente o uso de memória ao representar um cubo parcial. Observamos a partir dos resultados que a abordagem Frag-Cubing consumiu 63% mais memória RAM do que a abordagem HIC quando os dados da relação são uniformes (S = 0). Em relações skewed (por exemplo: S = 2,5) o consumo de memória da abordagem Frag-Cubing é 84% maior do que da abordagem HIC. Os resultados são semelhantes quando comparamos as abordagens H-Frag e Frag-Cubing.

Quando há relações *skewed* com aproximadamente metade dos valores de atributos armazenados em RAM e metade armazenados em memória externa nas abordagens HIC e H-Frag, a diminuição significativa no consumo de memória é justificada pela frequência irregular de valores de atributos; desta maneira, a frequência acumulada crítica é atribuída para todos os valores de atributos não *skewed*, com isso, todas as listas de TIDs destes valores de atributos são armazenadas em memória externa.

5.4.4 Tempo para responder consultas

Os tempos de resposta a consultas da abordagem Frag-Cubing foram significativamente mais lentos do que os da abordagem HIC (aproximadamente 13 vezes). Essa diferença se dá em cenários nos quais muitos valores de atributos com frequência maior que a frequência acumulada crítica são armazenados em memória externa.

A abordagem Frag-Cubing não respondeu consultas com mais de dois operadores subcubo por não haver memória contígua suficiente em 128 GB de RAM. Tal abordagem necessita de alocação de grandes vetores com diversas células vazias, uma vez que Frag-Cubing sempre duplica o tamanho de suas estruturas de dados quando o limite estabelecido é alcançado. Em contraste, a abordagem HIC necessita de listas menores para representar as agregações de um cubo de dados, já que não considera células vazias.

O tempo de consulta também foi drasticamente reduzido devido a HIC de realizar a interseção das listas de TIDs a partir de valores de atributos menos frequentes. As Figuras 5.4, 4.5 e 4.6 ilustram os testes realizados com a mesma relação R utilizada nas seções anteriores.

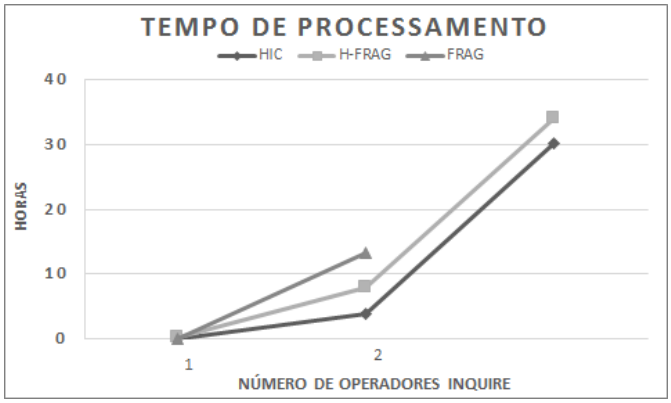

FIGURA 5.4 – Tempo de resposta para consulta com operadores inquire: T = 10^7, C = 10^4, D = 30 e S = 0.

Em geral, o tempo de resposta de consultas usando valores de atributos armazenados em memória principal e memória externa foram 2,5 vezes mais lentas do que as consultas que exigem valores de atributos armazenados somente em memória principal.

A Figura 5.5 mostra os resultados de testes de consultas usando valores de atributo com frequência menor que a frequência acumulada crítica e a Figura 5.6 ilustra resultados de testes com consultas usando os valores de atributos com frequência maior do que a frequência acumulada crítica.

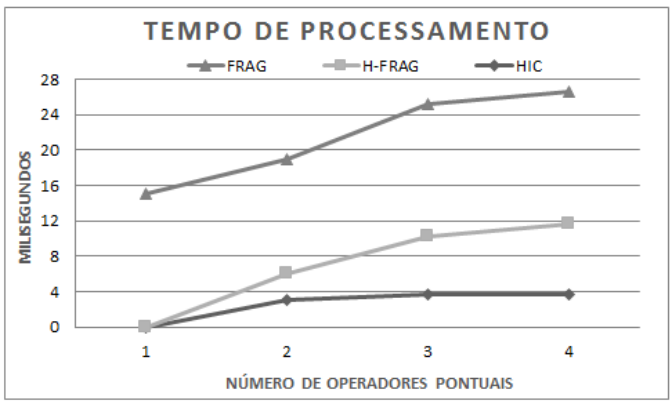

FIGURA 5.5 – Tempo de resposta de consultas pontuais para valores de atributos com frequência menor que a frequência critica: T=10^7, C=10^4, D=30 e S=0

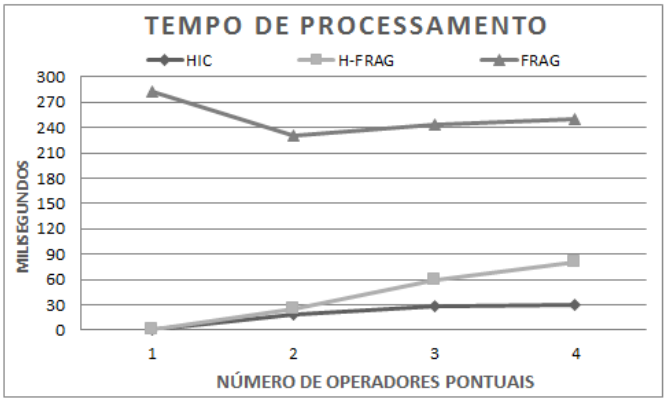

FIGURA 5.6 – Tempo de resposta de consultas pontuais para valores de atributos com frequência maior que a frequência critica: $T=10^7$, $C=10^4$, $D=30$ e $S=0$

5.4.5 Relações massivas

A relação com $T = 10^9$, $C = 10^4$, $D = 60$ e $S = 0$ foi computada pela abordagem HIC. Aproximadamente 80% dos valores de atributo foram armazenados em memória externa. Este teste levou 28 horas e consumiu 110 GB de RAM. Os resultados mostram que é possível computar cubos com elevado número de tuplas e alta-dimensionalidade utilizando a abordagem HIC sem realizar operações de swap do sistema operacional, permitindo assim que atualizações e consultas sejam realizadas.

Apesar de realizar operações de entrada e saída, HIC foi mais rápido do que Frag-Cubing para computar um cubo de dados massivo, portanto fica evidente que a estratégia de criar um vetor novo duas vezes maior do que a anterior, a cada vez que o tamanho da estrutura de dados atinge seu limite é ineficiente. Isso fica mais explicito quando se tem células vazias, pois nestes cenários realocações são feitas desnecessariamente.

HIC aloca listas pequenas, contíguas e complementares, que produzem mais realocações, porém reduz o número de células vazias. Na codificação do algoritmo da abordagem HIC utilizou-se a API FASTUTIL (2015) framework com algoritmos de interseção e união, além de prover estruturas de dados.

Consultas com cinco operadores *range*, dez operadores pontuais e um operador *inquire* foram respondidas em menos de 20 segundos. Não temos conhecimento de abordagens anteriores que consigam responder consultas com operadores de subcubos a partir de relações massivas como a apresentada nesta Seção.

A abordagem Frag-Cubing não conseguiu computar uma relação com 200 milhões de tuplas e 60 dimensões sem *swap* do sistema operacional, porém a abordagem de HIC computou um cubo a partir de uma relação com 1 bilhão de tuplas e 60 dimensões.

5.4.6 Experimentos com relações reais

HIC foi também experimentado com duas relações reais. A primeira relação foi uma relação com tipos de coberturas florestais obtidas no site do repositório de aprendizagem de máquina UC IRVINE (2015). Essa relação é a mesma utilizada em Frag-Cubing (LI; HAN; GONZALEZ, 2004). A relação com tipos de coberturas florestais contém T = 581,012 D = 54, as dimensões incluem 10 variáveis quantitativas, 4 zonas naturais representadas de forma binária e 40 variáveis de tipos de solo também representadas de forma binária. As cardinalidades são (1978, 361, 67, 551, 700, 5785, 207, 185, 255, 5827, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 7).

A computação utilizando HIC levou 18 segundos e consumiu 1 GB de RAM. Uma vez que a relação tem uma alta cardinalidade, consultas com operadores de subcubos são extremamente sensíveis dependendo das dimensões utilizadas. Uma consulta *inquire* em dimensões com cardinalidades 1978, 5785 e 5827 levou 764 milissegundos e utilizou 1,2 GB. Quando a mesma consulta foi realizada em dimensões onde as cardinalidades eram 2, 2 e 7 o tempo de resposta e o consumo de memória caíram significativamente para 62 milissegundos e 1,05 GB de espaço em RAM, respectivamente.

A segunda relação possui registros para comparações de padrões de links de páginas web. Tal relação também foi obtida a partir do repositório de aprendizagem de máquina UC IRVINE (2015). Esta relação contém T = 4.599.306 D = 12. As cardinalidades são (96,633, 96,636, 55, 31, 100, 35, 2, 3, 3, 3, 3, 2). A computação do cubo de dados para esta relação utilizando HIC levou 1,3 minutos e utilizou 550 KB.

Uma consulta inquire com duas dimensões e considerando as dimensões com cardinalidades 96633 e 96636 levou, em média, 6 horas para ser executada e consumiu 2 GB. Executamos esta mesma consulta na abordagem Frag-Cubing, porém mesmo utilizando 128 GB de RAM não foi possível executar tal consulta. Diante disto, refizemos a consulta e deixamos o sistema operacional realizar *swap*. Desta forma, além de utilizar os 128 GB de RAM disponíveis foi necessário o uso de mais 245 GB de disco para realizar operações de swap, com isso a abordagem Frag-Cubing necessitou de 8 horas para responder tal consulta.

Consulta *inquire* nas três dimensões com as cardinalidades mais elevadas (96.633, 96.636, 55) levou mais de onze horas e consumiu 10 GB na abordagem HIC, contudo Frag-Cubing não respondeu tal consulta. Consulta *inquire* em três dimensões com cardinalidades baixas (3, 3, 2) foi executada pela abordagem HIC em um minuto e não necessitou de memória

além da utilizada na computação do cubo. Neste experimento, a abordagem Frag-Cubing teve um desempenho melhor, executando em 8 segundos apenas e utilizando 1.2 GB de RAM.

5.4.7 Resumo

Neste capítulo detalhamos a primeira contribuição desta tese. A abordagem HIC utiliza memória principal e memória externa de forma híbrida; portanto cubos de dados com 60 dimensões e 10^9 tuplas podem ser computados. O parâmetro, denominado frequência acumulada crítica, foi definido para determinar a distribuição do cubo de dados em memórias principal e externa. Este parâmetro permite acessar os valores de atributos com alta frequência, armazenados na memória RAM de maneira mais eficiente dos que os valores de atributo menos frequentes, armazenados em memória externa.

Nossos experimentos demonstraram que HIC é uma solução eficiente, uma vez que o tempo de processamento e o consumo de memória são lineares quando há aumento do número de dimensões e quantidade de tuplas. Em geral, o consumo de memória da abordagem HIC foi muito menor do que o consumo de memória da abordagem Frag-Cubing. Além disso, HIC foi significativamente mais rápido do que Frag-Cubing para responder consultas pontuais e consultas com operadores subcubo.

HIC é também mais eficiente do que H-Frag, uma vez que reduz o seu consumo de memória e número de acessos a memória externa. Nos experimentos, estabelecemos cenários em que não é possível computar um cubo de dados com a abordagem Frag-Cubing, porém mesmo a partir de tais relações a abordagem HIC se mostrou factível ao conseguir computar e consultar um cubo. Os testes extremos com a abordagem HIC chegaram a uma relação com 60 dimensões e 10^9 tuplas, algo que não temos o conhecimento que tenha sido investigado pela literatura OLAP nas últimas décadas.

Apesar da abordagem HIC resolver o problema da computação de bases massivas, o problema de termos listas enormes de TIDs ainda tem que ser resolvido, pois o custo de interseção pode se tornar impraticável. Desta forma, passamos a investir uma solução mais eficiente para particionar TIDs, possibilitando uma rápida recuperação de alguns fragmentos em memória externa e não todos como ocorre atualmente na abordagem HIC.

No próximo capítulo apresentamos a abordagem bCubing, projetada para relações massivas e com alta dimensionalidade, assim como HIC. A abordagem bCubing resolve o problema do tamanho das listas de TIDs e o impacto em abordagens como Frag-Cubing, qCube

(SILVA; LIMA; HIRATA, 2013), H-Frag (SILVA; LIMA; HIRATA, 2015a) e HIC ao introduzir um índice invertido em dois níveis e o conceito e blocos de índices.

6 A abordagem bCubing

Neste capítulo, apresentamos a abordagem bCubing que possui um sistema híbrido de memória para armazenar partições de cubos de dados na memória externa.

Para endereçarmos uma solução utilizando memória híbrida para computação, atualização e consulta de cubo de dados massivos e com alta dimensionalidade, a abordagem bCubing implementa o conceito de blocos de índices invertidos em duas fases, utilizando como base o conceito de tuplas invertidas, denominadas *iT* e explicadas na Seção sobre Frag-Cubing (Seção 3.1).

Na abordagem bCubing, assim como na abordagem Frag-Cubing e na abordagem HIC, a ocorrência de um valor de atributo na relação de entrada *R* é associada à quantidade de TIDs. Os TIDs são únicos e são atribuídos sequencialmente a partir do início de *R*.

O conceito de bloco de índice invertido é utilizado para particionar a lista de TIDs, implementando a inversão de tuplas em dois níveis: um nível onde o identificador é o índice de bloco (BIDs) e o segundo nível onde o identificador é o índice de tupla (TIDs).

Um bloco está associado a uma sequência de TIDs. Formalmente, um bloco é definido como $B_i = \{iW_1, iW_2, ..., iW_d, iX_1, iX_2, ..., iX_m\}$, onde *d* é o número de valores de atributos e *m* é o número de valores de medida na relação *R*, $iW_j = \{[attr_m, tid_1, ..., tid_z], ... [attr_n, tid_1, ..., tid_p]\}$ e $iX_j = \{ [tid_1, mea_1, ..., mea_n] , ... [tid_1, mea_1, ..., mea_n]\}$. Cada valor de atributo *attr* de *R* possui uma lista de TIDs sempre com mesmo limite *[1, k]*, onde o tamanho de cada bloco, calculado a partir do número de tuplas computadas até o momento, deve ser constante. Desta forma, $1 \leq [tid_1, ..., tid_z] \leq k$ *e* $1 \leq [tid_1, ..., tid_p] \leq k$. Cada valor de *tid* de *R* possui uma lista de valores de medida *mea* que tem tamanho *n*.

Ao manter o tamanho do bloco constante, a abordagem bCubing mantém valores de atributos associados a diferentes números de TIDs, porém todos com mesmo limite de TIDs por bloco. Quando se atinge *k* tuplas lidas um novo bloco é criado e o processo se repete.

Note que o consumo de memória com a utilização de blocos é baixo, pois no pior caso tem-se $k.\sum_{i=1}^{D} Ci$ TIDs, onde *D* é o número de dimensões, *k* é o tamanho do bloco e *C* a cardinalidade de uma determinada dimensão. Numa relação com cardinalidade 100 em cada uma das 10 dimensões, temos 10^4 combinações de TIDs, mas existem *k* TIDs associados. Assumindo *k* igual a 10^4, temos 1 milhão de TIDs distintos no pior caso do exemplo escolhido. Note que em memória principal se armazena apenas 10^3 valores, enquanto em memória externa

10^6 valores, portanto é extremamente vantajosa a adoção de blocos e dois níveis de índices invertidos ao invés de apenas um único nível, conforme proposto nas abordagens atuais.

Para ilustrar melhor o conceito blocos de índices invertidos, consideramos uma relação *R* com sete tuplas, como apresentado na Tabela 6.1. A partir de *R*, abordagem bCubing faz a computação de um cubo de dados conforme ilustrado na Tabela 6.2. Formalmente, *R* é um conjunto de tuplas conforme definido no capítulo 4 na Subseção 4.1.1.

TABELA 6.1 – Relação de Entrada R com Sete Tuplas para Computação com bCubing

TIDs	A	B	C	M1
1	a_1	b_1	c_1	1.50
2	a_2	b_1	c_1	2.30
3	a_1	b_1	c_1	33.55
4	a_1	b_2	c_1	15.12
5	a_1	b_1	c_2	5.98
6	a_1	b_2	c_1	10.10
7	a_2	b_2	c_2	10.55

TABELA 6.2 – Conceito de blocos aplicado a R

BIDs	TIDs	A	B	C	Valor de Medida M1
1	1	a_1	b_1	c_1	1.50
	2	a_2	b_1	c_1	2.30
2	3	a_1	b_1	c_1	33.55
	4	a_1	b_2	c_1	15.12
3	5	a_1	b_1	c_2	5.98
	6	a_1	b_2	c_1	10.10
4	7	a_2	b_2	c_2	10.55

Em bCubing é definida uma quantidade fixa de tuplas para os blocos (*tBloc*) em termos do número de tuplas associadas a um valor de atributo, na Seção 6.6 é descrito como a quantidade de tuplas por blocos pode ser definida como base na memória principal disponível. Para o exemplo da Tabela 6.1, *tBloc=2*. Neste trabalho, o bloco é identificado como BID.

No exemplo descrito, para o processamento do bloco 1, as informações de ocorrência dos valores de atributos são armazenadas da seguinte forma:

1. todas as tuplas são computadas sequencialmente do início ao fim do bloco 1, ou seja, as duas primeiras tuplas são processadas;
2. para cada tupla, processam-se os valores de atributos e os valores de medidas do início ao fim da tupla. Note que cada valor de atributo e cada valor de medida está associado a uma dimensão do cubo de dados;
3. para cada valor de atributo, é verificado se o mesmo já existe no bloco. Caso seja um valor de atributo novo no bloco, é criado uma lista de TIDs. Cada ocorrência

do valor de atributo em uma tupla é representada como um TID. Ao final de um bloco, há [1, *tBloc*] TIDs por valor de atributo.

4. Cada ocorrência de um valor de medida em uma tupla é associada a um TID no bloco. Ao final de um bloco, há 1 TID por valor de medida.

No exemplo, o bloco 1 é associado às tuplas 1 a 2, o bloco 2 está associado às tuplas 3 a 4, e assim por diante. Cada bloco tem os valores de cada medida associado aos respectivos TIDs.

Depois de processar o bloco 1, as listas de TIDs de todos os valores de atributos e valores de medidas presentes no bloco são armazenadas em memória externa. Após armazenamento do bloco 1, processa-se o bloco 2, 3 e 4. O último bloco pode ter listas de TIDs com tamanhos menores do que aos dos demais blocos, pois o número de tuplas do último bloco pode ser menor que *tBloc*. No exemplo, o bloco 4 (bid_4) está associado a apenas o tid_7, enquanto os demais blocos estão associados a dois TIDs.

A Tabela 6.3 ilustra as listas de TIDs criadas no bloco 1 (bid_1). É criada uma lista de TIDs para cada valor de atributo presente nas tuplas processadas: a_1, a_2, b_1 e c_1.

TABELA 6.3 – Ocorrências de valores de atributos em termos de TIDs no bloco 1

BID	Valor de Atributo	TIDs
1	a_1	1
	a_2	2
	b_1	1,2
	c_1	1,2

A Tabela 6.4 contém as listas de TIDs criadas para todos os blocos da relação apresentada na Tabela 6.1. Tal representação é armazenada em memória externa.

Para identificar as ocorrências de um determinado valor de atributo é criada uma tabela auxiliar com identificadores dos blocos BIDs, conforme apresentado pela Tabela 6.5.

A Tabela 6.5 ilustra as ocorrências de todos os valores de atributos nos blocos da Tabela 6.4, com o respectivo tamanho das listas de TIDs associados aos valores de atributos em cada bloco. Por exemplo, o valor de atributo a_1 ocorre nos blocos 1, 2 e 3. O número de TIDs associados a cada bloco é 1, 2 e 2 respectivamente. No bloco 1, a_1 está relacionado ao tid_1. No bloco 2, a_1 relaciona-se com tid_3 e tid_4. No bloco 3, o relacionamento de a_1 é com tid_5 e tid_6.

TABELA 6.4 – Ocorrências de valores de atributos em termos de TIDs nos blocos para R

BID	Valor de Atributo	TIDs
1	a_1	1
	a_2	2
	b_1	1,2
	c_1	1,2
2	a_1	3,4
	b_1	3
	b_2	4
	c_1	3,4
3	a_1	5,6
	b_1	5
	b_2	6
	c_2	5
	c_1	6
4	a_2	7
	b_2	7
	c_2	7

TABELA 6.5 – Associação de Valores de Atributos com BIDs

Valor de Atributo	BIDs	Quantidade de TIDs
a_1	1	1
	2	2
	3	2
a_2	1	1
	4	1
b_1	1	2
	2	1
	3	1
b_2	2	1
	3	1
	4	1
c_1	1	2
	2	2
	3	1
c_2	3	1
	4	1

O algoritmo de computação bCubing, detalhado na Seção 6.3, armazena uma estrutura auxiliar *bCubingBlocRAM*, ilustrada pela Tabela 6.5, na memória principal e os blocos ilustrados pela Tabela 6.4 em memória externa. Mesmo na computação de relações massivas, ou seja, com bilhões de tuplas e centenas de dimensões, o consumo de memória principal da Tabela 6.5 é baixo, conforme descrevemos a seguir.

Sejam:

- nTuplas – número de tuplas de R;
- tBlocos – número de tuplas no bloco;
- nBlocos – número de blocos;

- D – número de dimensões;
- M – número de medidas;
- C_i – cardinalidade da dimensão i;

No pior caso, onde todos os valores de atributos ocorrem em todos os blocos, a Tabela 6.5 possui o seguinte número de linhas: $tAuxSize = (\sum_{i=1}^{D} Ci)\ \ nTuplas / tBloc$.

Suponha uma base com: 10^2 dimensões, onde cada dimensão com cardinalidade 10^4, número de tuplas igual 10^9 e o tamanho do bloco igual a 500K tuplas, portanto *tAuxSize* possui, no pior caso, tamanho igual a: $tAuxSize=(10^2 \times 10^4 \times 10^9 \ / \ 5 \times 10^5)=2 \times 10^6$ de linhas. Considerando que cada linha ocupa 8 bytes, 4 bytes referentes ao valor de atributo e 4 bytes referentes ao BID associado, teremos ≅16.000.000.000 bytes ≅ 16GB.

Note que a relação completa (Tabela 6.4) ocupa cerca de $4+(4 \times 10^2) \times 10^9$ bytes ≅ 370 GB, 4 bytes referentes a cada TID e 4×10^2 bytes referentes a cada valor de atributo de cada dimensão de uma tupla, portanto a ideia de blocos oferece um equilíbrio na utilização de memórias com diferentes velocidades.

Sendo *tid* e *attr* um inteiro de 4 bytes e *mea* um valor real de 8 bytes temos cada tupla *t* com tamanho igual a $tSize = 4 + (D \times 4) + (M \times 8)$, assim o tamanho da relação *R* é $rSize = (nTuplas \times tSize)$.

O tamanho em relação ao espaço de armazenamento de um bloco *bSize* é composto por um tamanho fixo *tFixoTuplas*, onde $tFixoTuplas = \sum_{i=1}^{D} 4\ x\ Ci\ x\ 4$, assuminindo que todos os valores de atributos ocorram em todos os blocos, dado que o identificador do bloco e o valor de atributo ocupam 4 bytes, e tem também um tamanho variável que é composto pelo espaço necessário para a associação de valores de atributos *tVariavelAtt* e suas respetivas listas de TIDs e também o espaço ocupado pelos valores de medida associados com cada TID *tVariavelM*.

Mantido em memória externa *tVariavelAtt* tem tamanho $tVariavelAtt = tBloc\ x\ D\ x\ 4$, assumindo que todas tuplas possuem valores para todas medidas de *R*, onde cada valor de medida ocupa 8 bytes sendo necessário mais 4 bytes para o TID associado, $tVariavelM = tBloc\ x\ M\ x\ (8+4)$.

Com o objetivo de facilitar a obtenção dos valores de medidas para cada bloco, optamos por armazenar cada medida em um arquivo especifico para cada BID, assim dado um BID conseguimos recuperar todos os valores de medidas associados a tal bloco (sendo que o identificador do arquivo é o BID).

Como é possível perceber, determinar o *tBloc* é de fundamental importância para viabilizar a computação e a execução de consultas na abordagem bCubing. O *tBloc* deve ser definido no momento da computação do cubo de dados e deve possibilitar a execução de consultas com a memória de trabalho disponível. Para melhor entendimento, discutimos como definir *tBloc* após descrevermos o particionamento dos blocos, fazermos a definição de um cubo de dados bCubing e detalharmos os algoritmos de computação, atualização e consulta na Seção 6.6.

6.1 Particionamento dos blocos de índices invertidos

Na abordagem bCubing os blocos são particionados inicialmente por dimensões com objetivo de tornar mais eficiente o processamento e a gestão dos blocos. A Figura 6.1 ilustra o particionamento realizado para o exemplo apresentado na Tabela 6.1.

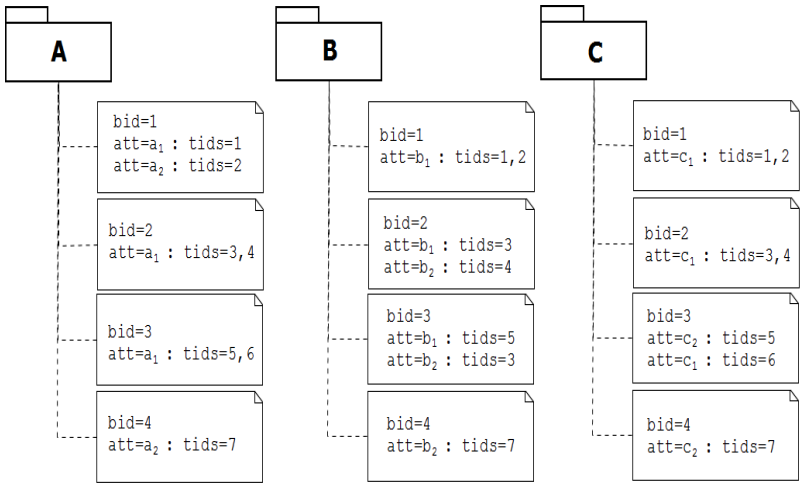

FIGURA 6.1 – Particionamento de dados na abordagem bCubing

Ao receber uma consulta do usuário, são executados interseções e uniões usando BIDs e TIDs, gerando assim resultados bCubing (bCqR).

Suponha a consulta onde se almeja as ocorrências (COUNT) de quando os valores de atributos a_1 e b_1 ocorrem juntos (agregação a_1, b_1, *). A partir da Tabela 6.5, é verificado que o valor de atributo a_1 ocorre nos blocos 1, 2 e 3 e o valor de atributo b_1 ocorre também nos blocos 1, 2 e 3. A interseção das duas listas de BIDs resulta na lista *{1, 2, 3}*. Em cada um dos blocos verifica-se em quais TIDs existe a ocorrência conjunta de a_1 e b_1.

Acessando a estrutura da Tabela 6.4, é possível obter que no bloco 1 a lista de TIDs de a_1 é *{1}*, b_1 tem a lista *{1, 2}* de TIDs. Assim através da interseção das duas listas de TIDS,

sabemos que a_1 e b_1 ocorrem em tid_1. No bloco 2, a_1 tem a lista de TIDs *{3, 4}* e b_1 possui a lista de TIDs *{3}*. Assim, sabemos a_1 e b_1 ocorrem em tid_3. No bloco 3, a_1 possui a lista de TIDs {5, 6} enquanto b_1 tem a lista de TIDs *{5}*. Com isso, a_1 e b_1 estão presentes em tid_5. Por fim, o resultado da consulta é que o COUNT para a agregação $a_1, b_1, *$ é 3, já que tal agregação ocorre nas tuplas tid_1, tid_3 e tid_5.

A lista resultado de TIDs contendo tid_1, tid_3 e tid_5, é processada de tal maneira que é recuperado os valores de medida usando cada um dos TIDs da lista resultado e possibilita a realização de cálculos estatísticos variados com os valores de medida. A execução de cálculos estatísticos inclui cálculos a partir de medidas holísticas como moda e rank, assim como as medidas desvio padrão ou variância e muitas outras.

Uma vantagem significativa da abordagem bCubing é que, diferentemente da abordagem HIC, qualquer valor de atributo de *R* pode se tornar cada vez mais frequente na relação sem necessidade de atualizações em blocos já armazenados em memória externa. Basta associar tal valor de atributo a novos blocos formados a partir de novos intervalos de TIDs referentes às novas tuplas atualizadas na base de dados. Desta forma, evitam-se operações de acesso a memória externa, algo extremamente eficiente para cenários onde atualizações são recorrentes.

Outra característica interessante da abordagem bCubing é que ao realizar consultas apenas poucas listas de TIDs são carregadas em memória principal. Isto é devido aos blocos estarem associados a conjuntos fixos e complementares de TIDs de *R*. Não importa se um valor de atributo possui 0 ou *n* TIDs em um determinado bloco. O tamanho do bloco não se altera por conta disto. Desta forma, basta saber quais blocos um determinado valor de atributo de uma consulta está associado. Os demais valores de atributos contidos na consulta sempre estarão presentes em tais blocos. Reduzimos ainda mais o espaço de busca ao proceder interseções sucessivas de blocos usando todos os valores de atributos de uma consulta.

6.2 Definição de um cubo de dados bCubing

Formalmente, um cubo de dados bCubing (bC) pode ser definido como: dado *R* e um algoritmo de computação bCubing CB, a saída é um cubo de dados $bC=\{iD_1, iD_2, ..., iD_n:(im_1, im_2, ..., im_x)\}$, onde cada $iD_i=\{(att_1, bid_1, iTati), (att_1, bid_2, iTati), (att_2, bid_2, iTati), ..., (att_z, bid_n, iTati)\}$. Cada elemento interno *iTati* de bC representa um conjunto de tuplas invertidas de um bloco de uma dimensão específica, representada por att_i. Cada *iTat* é definido como

iTat=(tid₁, ..., tidₚ), representando os TIDs associados a um valor de atributo att_i de *R*, cujo tamanho máximo é *p*.

Um cubo de dados bC também tem $(im_1, im_2, ..., im_x)$, onde cada *im* é definida como *im=(bid, tid, mv)*, sendo uma medida também invertida em dois níveis, composta por um BID, TID e um valor de medida *mv* numérico. Note que o cubo bC é particionado de acordo com as dimensões iD_1, iD_2, e medidas im_1, im_2, ..., im_x. Desta forma, garantimos que os valores de atributos att_i de uma dimensão iD_x podem ser iguais aos valores de atributos de outra dimensão iD_y. Cada bloco bid_i é um identificador de *B*, definido no início deste capítulo.

6.3 O Algoritmo de computação bCubing

O Algoritmo 6.1 demonstra como é feito a computação de cubos parciais na abordagem bCubing. O algoritmo recebe como entrada a relação *R* e a quantidade de tuplas para os blocos *B*, definido por *tBloc*.

Algoritmo 6.1: computação de cubo de dados indexadas por blocos de índices invertidos.

Entrada: (1) *R* com um conjunto de tuplas *t*, onde $t=(tid, D_1, D_2, ..., D_n)$; e a quantidade de tuplas para os blocos *B*, definido por tBloc.

Saída: Um cubo $bC=\{iD_1, iD_2, ..., iD_n:(im_1, im_2, ..., im_x)\}$.

Método:

```
1. bCount ← 0;
2. while R has tuples {
3.  generates bid whenever bCount is less than tBloc
4.   while t has dimensions{
5.      build with two inverted index an entry  in bCubingBloc[Dᵢ]: {att, {bid,
      tids}}}
6.     bCount ← bCount+1;
     }
7.   while t has measures{
8.      build with two inverted index an entry  in bCubingMeasures[Mᵢ]: {tid, mv}}
     }

9.   if bCount equal tBloc {
10.     serialize bCubingBloc and bCubingMeasures in bC
11.     for each D{
12.          bCubingBlocRAM[Dᵢ] ← bCubingBloc[Dᵢ]: {att, bid}}
        }
13.     reset bCubingBloc and bCubingMeasures
14.     bCount ← 0;
     }
    }
15. if processed tids smaller than tBloc {
16.    serialized bCubingBloc and bCubingMeasures in bC
    }
17.     reset bCubingBloc and bCubingMeasures
```

No algoritmo CB, `bCubingBloc` é a variável principal, responsável por armazenar todos valores de atributos da relação *R* de cada bloco até este ser armazenado em memória externa, cada valor de atributo associado a BIDs e cada BID associado as listas de TIDs. Em `bCubingMeasures` tem-se todos os valores de medidas de *R*. Para cada tupla de *R*, há uma ou mais entradas nos mapas `bCubingMeasures`. A variável `bCubingBlocRAM` representa a estrutura auxiliar que permanece em memória principal com a associação dos valores de atributos com suas respetivas listas de BIDs.

A variável `bCubingBloc`, `bCubingBlocRAM` e `bCubingMeasures` são vetores de mapas, onde cada posição do vetor representa uma dimensão para `bCubingBloc` e `bCubingBlocRAM` e uma medida para `bCubingMeasures`.

A computação inicia atribuindo 0 a variável auxiliar `bCount`, utilizada para definir em que momento cada bloco gerado é armazenado em memória externa. Em seguida, para cada tupla de *R* há a verificação se o valor da variável `bCount` é menor que `tBloc`. Caso positivo, é verificado se `bCount` é igual a 0. Caso `bCount` seja 0, significa que a computação está no início ou um bloco acabou de ser armazenado em memória externa, então BID é incrementado (linha 3).

Com base na definição do tamanho de blocos (`tBloc`), o algoritmo computa os valores de atributos de cada dimensão (linha 4) e valores de medida (linha 7) de cada tupla de um bloco. Cada vez que o número de tuplas processadas (controladas por `bCount`) for igual ao tamanho do bloco é feito o armazenamento em memória externa (linhas 9 e 10) e é atribuído à `bCubingBlocRAM` todos valores de atributos com o BID corrente (linhas 11 e 12). Na linha 13 é feito a reinicialização das variáveis `bCubingBloc` e `bCubingMeasures`.

Ao fim, são verificadas as tuplas remanescentes, ou seja, menores que `tBloc` (linhas 15 e 16). Caso existam tais tuplas, é feito o armazenamento em memória externa final.

Com o objetivo de detalhar o funcionamento do algoritmo CB passo a passo, consideramos uma relação *R* com sete tuplas, três dimensões (A, B, C) e duas medidas (M_1 e M_2). A Tabela 6.6 ilustra *R*, onde os identificadores únicos de cada tupla são representados por TIDs. Neste exemplo, considera-se que *tBloc* possui duas tuplas como limite para armazenamento em memória externa.

A Tabela 6.7 ilustra os valores das variáveis do algoritmo proposto antes de indexar a primeira tupla (tid_1).

TABELA 6.6 – Relação de Entrada R com Sete Tuplas, Três Dimensões e Duas Medidas

TID	A	B	C	M1	M2
1	a_1	b_1	c_1	2.56	10.00
2	a_2	b_2	c_2	3.14	20.00
3	a_1	b_1	c_1	2.45	10.00
4	a_1	b_3	c_3	6.70	11.00
5	a_1	b_1	c_4	9.00	-
6	a_5	b_5	c_4	1.00	1.00
7	a_1	b_2	c_3	0.80	1.00

TABELA 6.7 – Variáveis do Algoritmo de Computação bCubing após o Processamento de tid_1

Variável	Valor
`tBloc`	2
`bid`	1
`t`	`tupla com tid`$_1$
`bCount`	1

É feita a iteração em todas dimensões de `t`. É neste momento que há a inversão de índices em dois níveis para cada tupla. Para cada dimensão, computa-se o valor de atributo no bloco corrente. Caso ainda não haja uma entrada para tal valor de atributo, cria-se uma entrada em `bCubingBloc`. Caso já exista o valor de atributo, é feita uma atualização na estrutura de dados `bCubingBloc`, passando o novo BID associado a TID de `t`. No exemplo, a inversão de `t` estabelece que a_1, b_1 e c_1 estejam associados a tid_1 e a bid_1, em uma estrutura para cada dimensão de `t`, conforme ilustra a Tabela 6.8.

TABELA 6.8 – bCubingBloc após o Processamento de tid_1

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
B	1	b_1	1	1
C	1	c_1	1	1

Após iterar sobre as dimensões, realiza-se a iteração sobre as medidas de uma tupla. Para cada tupla insere-se na estrutura de dados `bCubingMeasures` o valor numérico da medida (*value*), o TID e o BID. A Tabela 6.9 ilustra o estado da variável `bCubingMeasures` após o processamento da primeira tupla de *R*.

TABELA 6.9 – bCubingMeasures após o Processamento de tid_1

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00

Então, é verificado se `bCount` tem o valor igual ao `tBloc` (linha 9). Caso `bCount` seja igual `tBloc` há o armazenamento do bloco em memória externa, incluindo as medidas

que formam tal bloco. No exemplo em questão `tBloc` é igual a 2, portanto ao computar tid_1 `bCount` tem valor 1, desta maneira se dá a inversão de índices da tupla tid_2 que faz com que a_2, b_2 e c_2 estejam associados a tid_2 e a bid_1. Com isso é feito uma inserção para cada novo valor de atributo em `bCubingBloc`. A Tabela 6.10 ilustra o estado da variável `bCubingBloc` após computar tid_2.

TABELA 6.10 – bCubingBloc após o Processamento de tid_2

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
		a_2	1	2
B	1	b_1	1	1
		b_2	1	2
C	1	c_1	1	1
		c_2	1	2

Ao finalizar a inversão dos índices de tid_2, suas medidas são armazenadas em `bCubeMeasure`, conforme ilustra a Tabela 6.11.

TABELA 6.11 – bCubingMeasures Após o Processamento de tid_2

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00
	2	3.14	20.00

Antes de computar a terceira tupla, o valor de `bCount` é igual ao valor definido para `tBloc`, portanto é atribuído 0 à `bCount` e é feito o armazenamento de `bCubingBloc` e `bCubingMeasures` em memória externa (linha 9 e 10), é criado um diretório para cada dimensão e cada medida, conforme ilustrado na Figura 6.1.

Somente a variável `bCubingBlocRAM` com todos valores de atributos associados a suas respectivas listas de TIDs permanecem em memória principal, a Tabela 6.12 ilustra o estado da variável `bCubingBlocRAM` após computar tid_2.

TABELA 6.12 – bCubingBlocRAM Após o Processamento de tid_2

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
A	a_1	1	1
	a_2	1	1
B	b_1	1	1
	b_2	1	1
C	c_1	1	1
	c_2	1	1

Após o armazenamento do bloco em memória externa e a inicialização de `bCubingBloc` e `bCubingMeasures`, inicia-se o processamento da tupla com tid_3. Neste exemplo, BID passa a ter valor 2, `bCount`, que teve seu valor definido para 0 quando ocorreu

o armazenamento do bloco com bid_1 em memória externa (linhas 9 e 10), volta a ter o valor 1 (linha 6) para auxiliar na serialização de bid_2. A tupla com tid_3 é composta pelos mesmos valores de atributos da tupla tid_1. Desta forma, a_1, b_1 e c_1 passam a estar associados também ao bloco bid_2 e tid_3, conforme ilustra a Tabela 6.13.

TABELA 6.13 – bCubingBloc após o Processamento de tid_3

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	2	a_1	1	3
B	2	b_1	1	3
C	2	c_1	1	3

As medidas de tid_3 são armazenadas em `bCubingMeasures`, já reinicializada após a serialização de bid_1, portanto só há os valores das medidas de tid_3 na Tabela 6.14.

TABELA 6.14 – bCubingMeasures após o Processamento de tid_3

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
2	3	2.45	10.00

A computação da próxima tupla, tid_4, atualiza `bCount` para o valor 2 antes de iniciar a inversão de índice. Os valores de atributos: a_1, b_3 e c_3 são associados a tid_4 e a bid_2. A Tabela 6.15 ilustra o estado de bCubingBloc após computar tid_4.

TABELA 6.15 - bCubingBloc após o Processamento de tid_4

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	2	a_1	2	3,4
B	2	b_1	1	3
		b_3	1	4
C	2	c_1	1	3
		c_3	1	4

Com a conclusão da inversão dos índices de tid_4, armazena-se suas medidas em `bCubingMeasures`, conforme ilustra a Tabela 6.16.

TABELA 6.16 – bCubingMeasures após o Processamento de tid_4

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
2	3	2.45	10.00
	4	6.70	11.00

Após o processamento da tupla com *tid4*, `bCubingMeasures` e `bCubingBloc` são armazenadas em memória externa conforme apresenta a Tabela 6.17 e Tabela 6.18 respectivamente.

TABELA 6.17 – bCubingMeasures em Memória Externa após o Processamento de tid_4

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00
	2	3.14	20.00
2	3	2.45	10.00
	4	6.70	11.00

TABELA 6.18 – bCubingBloc em Memória Externa após o Processamento de *tid4*

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
		a_2	1	2
	2	a_1	2	3,4
B	1	b_1	1	1
		b_2	1	2
	2	b_1	1	3
		b_3	1	4
C	1	c_1	1	1
		c_2	1	2
	2	c_1	1	3
		c_3	1	4

A Tabela 6.19 ilustra como os BIDs associados com valores de atributos permanecem em memória principal na variável `bCubingBlocRAM` após o processamento da tid_4.

TABELA 6.19 – bCubingBlocRAM após o Processamento de tid_4

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
A	a_1	1	1
		2	2
	a_2	1	1
B	b_1	1	1
		2	1
	b_2	1	1
	b_3	2	1
C	c_1	1	1
		2	1
	c_2	1	1
	c_3	2	1

A Tabela 6.20 ilustra o terceiro bloco com as tuplas cinco e seis. Finalizado a construção do bloco bid_3, `bCubingMeasures` recebe as medidas de tid_5 e tid_6.

TABELA 6.20 – bCubingBloc após o Processamento de tid_5 e tid_6

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	3	a_1	1	5
		a_5	1	6
B	3	b_1	1	5
		b_5	1	6
C	3	c_4	2	5,6

É importante ressaltar que no bid_3 `bCubingMeasures` recebe apenas uma medida, pois tid_5 não possui a medida M2, assim como ilustra Tabela 6.21.

TABELA 6.21 – bCubingMeasures após o Processamento de tid_5 e tid_6

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
3	5	9.00	-
	6	1.00	1.00

Com isso temos em memória externa as estruturas de `bCubingBloc` e `bCubingMeasures` atualizadas conforme Tabelas 6.22 e 6.23.

TABELA 6.22 – bCubingBloc em Memória Externa após o Processamento de tid_5 e tid_6

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
		a_2	1	2
	2	a_1	2	3,4
	3	a_1	1	5
		a_5	1	6
B	1	b_1	1	1
		b_2	1	2
	2	b_1	1	3
		b_3	1	4
	3	b_1	1	5
		b_5	1	6
C	1	c_1	1	1
		c_2	1	2
	2	c_1	1	3
		c_3	1	4
	3	c_4	2	5,6

TABELA 6.23 – bCubingMeasures após o Processamento de tid_5 e tid_6

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00
	2	3.14	20.00
2	3	2.45	10.00
	4	6.70	11.00
3	5	9.00	-
	6	1.00	1.00

Desta maneira, na Tabela 6.24 são observados os valores de atributos de todas tuplas, processados até o momento, associados com seus respectivos BIDs em memória principal na variável `bCubingBlocRAM` após o processamento da tid_6.

Finalmente, há a computação da última tupla de R (tid_7). Assim como nos demais casos, a variável `bCount` é definida com valor 1 e BID como 4. A inversão de índice para tid_7 estabelece que a_5, b_1 e c_3 estão associados ao bloco bid_4 e tid_7, conforme ilustra a Tabela 6.25.

Concluído a inversão de índice de tid_7, processam-se todos os valores de medida relacionadas a tal tupla, conforme se vê na Tabela 6.26.

TABELA 6.24 – bCubingBlocRAM após o Processamento de tid_5 e tid_6

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
A	a_1	1	1
		2	2
		3	2
	a_2	1	1
	a_5	3	1
B	b_1	1	1
		2	1
		3	1
	b_2	1	1
	b_3	2	1
	b_5	3	1
C	c_1	1	1
		2	1
	c_2	1	1
	c_3	2	1
	c_4	3	2

TABELA 6.25 – bCubingBloc após o Processamento de tid_7

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	4	a_1	1	7
B	4	b_2	1	7
C	4	c_3	1	7

TABELA 6.26 – bCubingMeasures Após o Processamento de tid_7

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
4	7	0.80	1.00

Neste momento a variável `bCount` é igual a 1, assim não é igual a `tBloc`, porém não existem mais tuplas a serem computas, desta forma finaliza-se o laço de repetição sobre as tuplas de *R* (linha 2) e realiza-se o armazenamento em memória externa do último bloco, uma vez que *R* chegou ao fim e `bCount` é maior que `0` (linhas 14 e 15).

As Tabelas 6.27, 6.28 e 6.29 ilustram o cubo bCubing gerado a partir da relação *R* apresentada na Tabela 6.6.

A Tabela 6.27 representa os blocos armazenados em memória externa, onde em cada dimensão temos todos BIDs gerados indexando os valores de atributos associados a tais blocos a suas respectivas listas de TIDs.

Na Tabela 6.28 observamos cada valor de medida, associado a cada uma das tuplas de um determinado bloco, dentre as possíveis medidas do cubo de dados.

Por fim, na Tabela 6.29 observamos a estrutura auxiliar que permanece em memória principal onde é possível obtermos todos BIDs associados a cada valor de atributo de cada dimensão do cubo de dados.

TABELA 6.27 – bCubingBloc em Memória Externa após o Processamento de tid_7

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
		a_2	1	2
	2	a_1	2	3,4
	3	a_1	1	5
		a_5	1	6
	4	a_1	1	7
B	1	b_1	1	1
		b_2	1	2
	2	b_1	1	3
		b_3	1	4
	3	b_1	1	5
		b_5	1	6
	4	b_2	1	7
C	1	c_1	1	1
		c_2	1	2
	2	c_1	1	3
		c_3	1	4
	3	c_4	2	5,6
	4	c_3	1	7

TABELA 6.28 – bCubingMeasures em Memória Externa após o Processamento de tid_7

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00
	2	3.14	20.00
2	3	2.45	10.00
	4	6.70	11.00
3	5	9.00	-
	6	1.00	1.00
4	7	0.80	1.00

TABELA 6.29 – bCubingBlocRAM após o Processamento de tid_7

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
A	a_1	1	1
		2	2
		3	1
		4	1
	a_2	1	1
	a_5	3	1
B	b_1	1	1
		2	1
		3	1
	b_2	1	1
		4	1
	b_3	2	1
	b_5	3	1
C	c_1	1	1
		2	1
	c_2	1	1
	c_3	2	1
		4	1
	c_4	3	2

Cada bloco representa um conjunto de tuplas em memória externa. O valor de atributo a_1 está associado a cinco TIDs e quatro BIDs, desta forma é o valor de atributo mais frequente

de *R*. Os valores de atributos a_2, b_3, b_5, c_2 e c_4 são os menos frequentes, estando associados a apenas um TID e um BID em *R*.

6.4 O Algoritmo de atualização bCubing

A estratégia de partição do cubo de dados em blocos se mostra eficiente para os quatro tipos de atualizações explicados na Seção 5.2. Considerando a atualização onde são inseridas novas tuplas, basta a execução do algoritmo de computação, descrito na Seção 6.3, pois basta inserir novos blocos e TIDs, persistindo os blocos quando se atinge o limite definido pelo usuário.

Para exemplificar o tipo de atualização I, considera-se a relação de atualização RA, ilustrada na Tabela 6.30.

TABELA 6.30 – Relação de Atualização RA com Três Novas Tuplas

TID	A	B	C	M1	M2
8	a_5	b_3	c_1	3.50	14.58
9	a_2	b_3	c_2	5.15	9.50
10	a_1	b_6	c_5	3.68	50.00

Três novas tuplas são inseridas em bC. Em RA há valores de atributos já computados em bC, tais como a_1, a_2, a_5, b_3, c_1 e c_2 e também a presença dos novos valores de atributos b_6 e c_5. Nas Tabelas 6.31, 6.32 e 6.33 é ilustrado o cubo bC após o processo de atualização proposto. Sendo que as Tabelas 6.31 e 6.32 representam as estruturas `bCubingMeasures` e `bCubingBloc`, respectivamente em memória externa, e a Tabela 6.33 apresenta a estrutura `bCubingBlocRAM` em memória principal.

TABELA 6.31 – bCubingMeasures Após o Processamento de tid_8, tid_9 e tid_{10}

BIDs	TIDs	Valor de Medida M1	Valor de Medida M2
1	1	2.56	10.00
	2	3.14	20.00
2	3	2.45	10.00
	4	6.70	11.00
3	5	9.00	-
	6	1.00	1.00
4	7	0.80	1.00
5	8	3.50	14.58
	9	5.15	9.50
6	10	3.68	50.00

TABELA 6.32 – bCubingBloc Após o Processamento de tid_8, tid_9 e tid_{10}

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
A	1	a_1	1	1
		a_2	1	2
	2	a_1	2	3,4
	3	a_1	1	5
		a_5	1	6
	4	a_1	1	7
	5	a_5	1	8
		a_2	1	9
	6	a_1	1	10
B	1	b_1	1	1
		b_2	1	2
	2	b_1	1	3
		b_3	1	4
	3	b_1	1	5
		b_5	1	6
	4	b_2	1	7
	5	b_3	2	8,9
	6	b_6	1	10
C	1	c_1	1	1
		c_2	1	2
	2	c_1	1	3
		c_3	1	4
	3	c_4	2	5,6
	4	c_3	1	7
	5	c_1	1	8
		c_2	1	9
	6	c_5	1	10

Para atualizações do tipo II, onde valores de atributos de *R* podem ser fundidos, deve-se considerar que tal fusão pode utilizar valores de atributos já existentes em *R* ou gerar um novo valor de atributo em *R*. Em ambos os cenários alguns blocos devem ser carregados em memória principal, tendo seus TIDs fundidos.

Como exemplo para o tipo II, onde se usa um valor de atributo existente na fusão, exemplificamos ao considerar que c_3 deva ser fundido com c_1. A Tabela 6.34 ilustra os valores de atributos da dimensão C com BIDs e TIDs atualizados em memória externa.

No exemplo, bid_2 passa a estar associado com tid_3 e também tid_4. O valor de atributo c_1 passa a estar associado ao bid_4, associações estas que não existiam anteriormente. Na Tabela 6.35 apresenta a estrutura auxiliar `bCubingBlocRAM` atualizada após a fusão do valor de atributo c_3 com c_1.

Para a atualização onde a fusão de *n* valores de atributos cria um novo valor de atributo, utilizamos o exemplo onde os valores de atributos b_2, b_3 e b_5 são alterados para b_4, não presente no cubo bC computado a partir de *R* e representado nas Tabelas 6.27, 6.28 e 6.29. De uma forma geral, atualizações do tipo (II) são simples de serem implementadas e seu custo computacional depende da frequência do valor de atributo em *R*, conforme ilustra Tabela 6.36.

TABELA 6.33 – bCubingBlocRAM Após o Processamento de tid_8, tid_9 e tid_{10}

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
A	a_1	1	1
		2	2
		3	2
		4	1
		6	1
	a_2	1	1
		5	1
	a_5	3	1
		5	1
B	b_1	1	1
		2	1
		3	1
	b_2	1	1
		4	1
	b_3	2	1
		5	2
	b_5	3	1
	b_6	6	1
C	c_1	1	1
		2	1
		5	1
	c_2	1	1
		5	1
	c_3	2	1
		4	1
	c_4	3	2
	c_5	6	1

TABELA 6.34 – bCubingBloc após a Fusão do Valor de Atributo c_3 com c_1

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
C	1	c_1	1	1
		c_2	1	2
	2	c_1	1	3,4
	3	c_4	2	5,6
	4	c_1	1	7

TABELA 6.35 – bCubingBlocRAM Após a Fusão do Valor de Atributo c_3 com c_1

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
C	c_1	1	1
		2	2
		4	1
	c_2	1	1
	c_4	3	2

TABELA 6.36 – bCubingBloc após o Processamento da Atualização onde b_2, b_3 e b_5 são alterados para b_4

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
B	1	b_1	1	1
		b_4	1	2
	2	b_1	1	3
		b_4	1	4
	3	b_1	1	5
		b_4	1	6
	4	b_4	1	7

A Tabela 6.37 apresenta a estrutura auxiliar `bCubingBlocRAM` atualizada após a atualização onde b_2, b_3 e b_5 são alterados para b_4.

TABELA 6.37 – bCubingBlocRAM Após o Processamento da Atualização onde b_2, b_3 e b_5 são alterados para b_4

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
B	b_1	1	1
		2	1
		3	1
	b_4	1	1
		2	1
		3	1
		4	1
	b_6	6	1

Atualizações do tipo III, onde novas dimensões e novas medidas podem ser adicionadas a *R*, requerem que a nova dimensão ou medida seja percorrida para que seus valores sejam associados aos BIDs e TIDs já contidos em bC. No exemplo a seguir é adicionada à dimensão D e a medida M3 em *R*, conforme ilustra a Tabela 6.38.

Os BIDs e TIDs que constam em bC são associados aos valores de D e M3 sem qualquer alteração nos blocos existentes. As Tabelas 6.39, 6.40 e 6.41 ilustram as novas associações.

TABELA 6.38 – Relação de Entrada R com a Dimensão D e Medida M3.

TID	A	B	C	D	M1	M2	M3
1	a_1	b_1	c_1	d_1	2.56	10.00	88.00
2	a_2	b_2	c_2	d_1	3.14	20.00	79.00
3	a_1	b_1	c_1	d_1	2.45	10.00	-78.50
4	a_1	b_3	c_3	d_2	6.70	11.00	-99.50
5	a_1	b_1	c_4	d_1	9.00	-	10.700
6	a_5	b_5	c_4	d_1	1.00	1.00	1.00
7	a_1	b_2	c_3	d_2	0.80	1.00	2.50

TABELA 6.39 – A Estrutura da Dimensão D em bCubingBloc Armazenada em Memória Externa.

Dimensão	BIDs	Valor de Atributo	Quantidade de TIDs	TIDs
D	1	d_1	1	1,2
	2	d_1	1	3
		d_2	1	4
	3	d_1	2	5,6
	4	d_2	1	7

TABELA 6.40 – A Estrutura da Medida M3 em bCubingMeasures Armazenada em Memória Externa.

BIDs	TIDs	Valor de Medida M3
1	1	88.00
	2	79.00
2	3	-78.50
	4	-99.50
3	5	10.70
	6	1.00
4	7	2.50

TABELA 6.41 – A Estrutura da Dimensão D em bCubingBlocRAM.

Dimensão	Valor de Atributo	BIDs	Quantidade de TIDs
D	d_1	1	2
		2	1
		3	2
	d_2	2	1
		4	1

Atualizações do tipo IV, onde existe reorganização nas hierarquias de uma dimensão, tal como para um cubo gerado pela abordagem HIC não gera impacto em bC, já que a consulta pode proceder em qualquer ordem. Pois como para a abordagem HIC para a abordagem bCubing não há diferença se uma consulta utiliza os valores de atributos ABC ou CBA de *R*. Internamente, há sempre a inicialização da consulta pelo valor de atributo de *R* associado com o menor número de BIDs e TIDs.

A concepção de dois níveis de índices permite recuperar versões do cubo bC, onde determinados TIDs e blocos devem ser considerados. Desta forma, há na abordagem bCubing o conceito de deltas, onde cada delta corresponde a um conjunto de blocos e TIDs relacionados a um tipo de alteração, portanto há possibilidade de consultar os valores de medidas M2 antes e depois da fusão de b_2, b_3 e b_5 em b_4.

Na literatura, há poucos trabalhos sobre atualização de medidas holísticas em cubos de dados e ainda não há solução para armazenar deltas oriundos das inúmeras atualizações sem a reconstrução completa do cubo. Pelo fato de usar eficientemente memória externa em pequenos blocos, é possível realizar consultas ao cubo ilustrado pelas Tabelas 6.27, 6.28 e 6.29 ou pelas Tabelas 6.31, 6.32 e 6.33, portanto é possível consultar medidas ou dimensões que mais sofreram alterações no tempo ou especular alterações para inferir tendências ou realçar anomalias usando deltas bCubing.

6.5 Consultas em bCubing

Com o cubo de dados gerado, pode-se realizar consultas pontuais e consultas com múltiplas agregações em bC, tais consultas foram descritas na Seção 5.3.

Em cada tipo de consulta a abordagem bCubing possibilita a obtenção das medidas devidamente agregadas segundo uma ou várias funções estatísticas.

O algoritmo de consulta bCubing tal como o algoritmo de consulta HIC recebe como entrada uma consulta *Q* e gera como saída três ou mais subconjuntos de TIDs derivados de *pQ*, *rQ* (consultas com filtros de intervalo) e *iQ* (consultas com filtros *inquire*). Os resultados das

três sub-consultas são consolidados num único resultado de *Q* bCqR utilizando algoritmos de interseção de conjuntos com complexidade *O(nlgn)*, onde *n* é o número de elementos do conjunto.

Formalmente, em um cubo de dados *n*-dimensional ($d_1, d_2, ..., dD_n$), uma consulta pode ser definida como $Q=\{(att_1, att_2, ..., att_n), m_i(M_{1, ..., } M_n), ..., m_n(M_{2, ..., } M_n)\}$, onde cada att_i especifica um valor para a dimensão d_i e m_i define o tipo de medida requerida para uma ou várias medidas $(M_1, M_{2, ..., } M_n)$ de *R*. Os tipos de medida são: *frequência Fr, soma Sm, max Ma, min Mn, média Me, variância Va, desvio padrão Dp, mediana Md* e *moda Mo*. Utilizamos '*' para dimensões que não devem ser consideradas na consulta.

Uma consulta *pQ* recebe um cubo de dados, bC, onde é executado um filtro *f*. O filtro *f* pode ser definido como $f:\{eq_1 \cap eq_2 \cap ... \cap eq_n\}$, onde eq_i é o operador ith EQUAL de *f* aplicada a dimensão *i* de bC. Somente operadores EQUAL são usados em consultas pontuais. Desta maneira, dada uma consulta *Q*, inicialmente se processa as sub-consultas pontuais *pQ's*.

$rQ \in Q$ representa consultas de intervalo nas diferentes dimensões. *rQ* retorna como resultado um conjunto de agregações a partir de valores de atributos presentes em *R*.

Uma consulta *rQ* recebe um cubo de dados, bC, onde é executado um filtro *f´*. Os BIDs de *pQ* são interseccionados com BIDs de *rQ*. A partir dos BIDs resultantes desta interseção, os TIDs de *pQ* são interseccionados com TIDs de *rQ*. O filtro *f´* pode ser definido como $f: \{þ_1 \cap þ_2 \cap ... \cap þ_n\}$, onde $þ_i$ é o i-ésimo operador RANGE de *f* aplicada a dimensão *i* de bC. *f* e *f´* são filtros aplicados à diferentes dimensões. *þ* pode ser classificado como: *{maior que + menor que + entre + alguns + diferente + similar}*. Cada $þ_i$ retorna uma lista de BIDs para os valores que atendem o critério definido por um operador *rOp*. Com isso, é executado um grupo de interseções das listas de BIDs para cada possível associação entre valores de atributos instanciados em cada sub-consulta, sendo estas interseções sempre iniciadas a partir dos valores de atributos com menor conjunto de BIDs.

Diferente de um filtro *pQ f*, um filtro *rQ f´* tem muitas operações de interseção e união. Mais especificamente se utilizarmos, por exemplo, o operador *between* $þ_b$ podemos ter como resultado uma ou mais agregações para a respectiva dimensão.

O terceiro tipo de consulta denomina-se *iQ*. Uma consulta *iQ* recebe um cubo de dados bC, e executa um terceiro filtro *f´´*. O BIDs de *iQ* são interseccionados com BIDs de $(pQ \cap rQ)$, então a partir dos BIDs resultantes, as listas de TIDs são interseccionadas para se obter uma lista fina de TIDs. O filtro *f´´* pode ser definido como $f'': \{\mu_1 \cap \mu_2 \cap ... \cap \mu_n\}$, onde μ_i é o operador ith INQUIRE de *f´´* aplicado a dimensão *i* de bC. *f*, *f´* e *f´´* são filtros aplicados a diferentes dimensões μ pode ser classificado como: *{subcube + distinct}*.

Um subcubo de uma dimensão é composto por todos os valores de atributos, incluindo o valor *ALL*. Assim, quando utilizado o operador *subcube* em uma consulta *Q*, composta pelas dimensões ABC, há como resultado todas as agregações possíveis: A, B, C, AB, AC, BC e ABC. Há, portanto, $\prod_{i=1}^{SC}(Ci+1)$ resultados quando *Q* possui *SC* operadores *subcube* C_i indica a cardinalidade da dimensão i.

Operadores *subcube* e *distinct* são idênticos para uma dimensão. Neste exemplo, *Q* possui *dis* operadores *distinct* e *Ci* indica a cardinalidade da dimensão *i*. Uma consulta *iQ*, utilizando operadores *distinct* nas mesmas dimensões ABC, gera como retorno somente agregações do tipo ABC, portanto uma consulta com operador *distinct* está contida em uma consulta com operador *subcube*.

A Tabela 6.42 ilustra os símbolos e a sintaxe utilizada pela abordagem bCubing para os diversos tipos de operadores de consulta implementados.

TABELA 6.42 – Operadores de consulta da abordagem bCubing

Tipo de Operador	Símbolo	Sintaxe
maior que	>	a1, b1, >c1
menor que	<	a1, b1, <c5
entre	…	a1, b1,]c3,c5[
alguns		a1, b1, (c3, c5)
diferente	!=	a1, b1, !=c5
similar	!	a1, b1, !
subcubo	?	a1, b1, ?
distinto	~	a1, b1, ~

Consideremos a consulta *Q ={(?, b_1,]c_1,c_4[), Sm(M1)}*, onde o primeiro filtro define o operador inquire, que computa todas possíveis combinações entre as tuplas recuperadas nos filtros anteriores com todos os elementos da dimensão A. Já o segundo filtro define uma consulta pontual na dimensão B, onde, devem ser obtidas tuplas com o valor b_1. O terceiro filtro representa o operador de intervalo *between*, definido para tuplas que estejam entre c_1 e c_4. A medida requerida é a soma *Sm* aplicada nos valores de M1.

Para detalhar o funcionamento geral das consultas, vamos considerar dois exemplos de possíveis consultas.

Exemplo de consulta 1: dado uma consulta pontual *Q' ={*a_1, b_1, c_1, *Fr}* onde é desejado saber a frequência em que os valores de atributos a_1,b_3 e c_3 aparecem juntos no cubo de dados.

É feito a interseção das listas de BIDs, começando sempre pelas menores listas, dos valores de atributos da consulta em bC, sempre em pares de listas de BIDs.

Como observamos na Tabela 6.29, a_1 tem a seguinte lista de BIDs: *{1, 2, 3, 4}*, a lista de b_1 é composta pelos BIDs *{1, 2, 3}* enquanto a lista de c_3 é *{1, 2}*. Com isso c_3{1, 2} ∩ b_1{1,

2, 3}) = b_1c_3{1, 2}, em seguida faz-se a interseção do resultado com a próxima lista, logo *b_1c_3{1, 2} ∩ a_1{1, 2, 3, 4}) = {1, 2},* tal resultado definimos como *lBids* e *bPointSize* é a quantidade de BIDs em *lBids* que neste exemplo é 2.

Sendo *lBids {1, 2},* é recuperada da memória externa a lista de TIDs de cada valor de atributo associada a cada BID, assim sabemos que *a_1* em *bid_1* possui os TIDS *{1}* e no *bid_2 {3, 4},* conforme pode ser observado na Tabela 6.27. O valor de atributo *b_1* tem apenas o TID *{1}* no *bid_1* e *{3}* no *bid_2*. Por fim, *c_1* possui as listas de TIDs *{1}* e *{3}* no *bid_1* e *bid_2* respectivamente.

Em seguida é feito a interseção entre os TIDs de cada bloco em pares de lista, assim temos *a_1{1}∩b_1{1}={1}, a_1b_1{1}∩c_1{1}={1}, a_1{3,4}∩b_1{3}={3}* e *a_1b_1{3}∩c_1{3}={3},* por fim temos a união destes resultados, logo *{1} U {3} = {1, 3}* é atribuído a bCqR.

Como a consulta *Q'* estabelece que seja retornado a frequência da agregação *a_1, b_1, c_1* e bCqR possui *{1, 3}* o resultado é frequência igual a 2.

Para execução de uma consulta pontual é necessária uma lista de tamanho *tPontualRAM* que conterá o resultado da interseção entre as listas de BIDs contidas em *bCubingBlocRAM* que tem tamanho *tAuxSize*, conforme apresentado no início deste capítulo. No pior caso *tPontualRAM* tem tamanho *tPontualRAM=nBlocos x 4 bytes*.

Dado *lBids*, é recuperado da memória externa cada bloco de cada BID contido em *lBids*, assim é necessário manter temporariamente em memória principal cada um dos blocos recuperado. Para armazenar o resultado das interseções das listas de TIDs obtidas de *tVariavelAtt* de cada bloco, que foi recuperado da memória externa, é necessária mais uma lista de tamanho *tPointE* sendo *tPointE=nTuplas x 4* considerando 4 bytes para cada TID, no pior caso que é quando para cada lista o valor de atributo ocorre em todas as tuplas de todos os blocos. Caso o usuário necessite saber o resultado de algum cálculo algébrico é necessário manter temporariamente em memória principal *tVariavelM* que é recuperada da memória externa para que os cálculos possam ser realizados, logo o resultado de cada cálculo necessita de mais 8 bytes. Na Seção 6.6 é discutido o impacto do espaço necessário para a execução de consultas na definição da quantidade de tuplas por blocos.

Para estabelecer, o número de operações de acesso a memória externa *nIO*, definimos que *l* em uma consulta pontual é uma lista ordenada por dimensões de par de valores de atributo e sua respectiva dimensão até *d_n*, logo uma consulta pontual pode ser representada como *{(att_j, d_i)}, onde att_j* representa um valor de atributo e *d_i* representa uma dimensão específica onde o ocorre *att_j*. Com isso, a consulta *Q'* pode ser apresentada como *Q'={(a_1,1), (b_1, 2), (C_1,3)}* e *l* com tamanho *tL* 3. Desta maneira $nIO = bPointSize\ x\ tL$, logo para o exemplo de consulta *Q'* $nIO = 2\ x\ 3$.

O número de interseções *nIntersec* para computação de uma consulta pontual é *nIntersec = (d-1) x bPointSize* sendo *d* a quantidade de dimensões, assim para consulta *Q'* temos *nIntersec = (3-1) x 2=4* interseções.

Exemplo de consulta 2: Seja a consulta com dois operadores inquire *Q''={?, ?, c_3, Fr}*, onde é desejado todas possíveis agregações entre os valores de atributo da dimensão A com c_3, todas agregações de todos valores de atributo da dimensão B com c_3 e todas possíveis agregações entre os valores de atributos de AB com c_3. O número de operadores *inquire* em uma consulta é definido como *tInquire* e o número de valores de atributos pontuais é definido como *tAttPontual*.

O primeiro passo para responder a consulta *Q''* é recuperar a lista de BIDs do parâmetro c_3 que é *{2, 4}* conforme ilustrado na Tabela 6.29, esta lista estabelece *lBids*. Com isso deve-se recuperar todos valores de atributos das dimensões marcadas com *?* que estejam associados aos BIDs de *lBids*, assim sabemos que podem haver agregações com os valores de atributos a_1, b_1, b_2 e b_3, o número de valores de atributo associados aos BIDs de *lBids* é definido como *nAttInquire*, estes valores de atributos estão presentes nas dimensões com os operadores *?* que estão associados aos BIDs 2 e 4, conforme observado na Tabela 6.29.

O segundo passo é recuperar da memória externa os TIDs de todos valores de atributo associados aos BIDs *{2, 4}* nas dimensões A e B. Desta maneira os valores de atributo em bid_2 com suas respetivas listas de TIDs é a_1{3,4}, b_1{3}, b_3{4} e c_3{4}; e para bid_4 temos: a_1{7}, b_2{7} e c_3{7}. Uma vez com a lista de todos os TIDs dos possíveis valores das dimensões A e B, prossegue-se com a interseção entre estas listas, assim para bid_2 temos: a_1{3,4}$\cap c_3${4}={4}, b_1{3}$\cap c_3${4}={}, a_1c_3{4}$\cap b_1${3}={}, a_1c_3{4}$\cap b_3${4}={4}, b_3{4}$\cap c_3${4}={4}. Para bid_4 temos as interseções: a_1{7}$\cap c_3${7}={7}, a_1c_3{7}$\cap b_2${7}={7} e b_2{7}$\cap c_3${7}={7}. Por fim temos a união dos TIDs de agregações comuns entre os BIDs, neste exemplo somente a_1c_3 é comum nos BIDs 2 e 4, logo *{4} U {7} = {4, 7}*.

Então para a consulta *Q'' ={?, ?, c_3, Fr}*, temos como resultado um subcubo com as seguintes agregações: **, *, c_3 : 2; a_1, *, c_3 : 2*; *a_1, b_3, c_3 : 1*; **, b_3, c_3 : 1*; *a_1, b_2, c_3 : 1* e **, b_2, c_3 : 1*; O símbolo "*" representa qualquer valor de atributo e o número depois de ":" representa a frequência da agregação. Observe por exemplo que a agregação *a_1, *, c_3* tem frequência 2 dado que a lista final de TIDs desta agregação é *{4, 7}*.

O número de operações de acesso a memória externa para consultas de subcubo é dado também por $nIO = bPointSize\ x\ tL$. Desta maneira, para o exemplo de consulta *Q''* *nIO=2x3=6*, para computação de uma consulta de subcubo temos mais interseções do que em

uma consulta pontual, assim *nIntersec*=(*nAttInquire x tInquire)* + *tAttPontual*, logo para *Q''* temos *nIntersec* = (*4 x 2)* + *1=9*.

6.5.1 Consultas incluindo o cálculo de medidas holísticas

Para o processamento das medidas baseadas em cálculos estatísticos a abordagem bCubing utiliza a biblioteca COMMONS MATH (2015). O resultado de uma consulta bCqR é utilizado pelo algoritmo de consulta bCubing a fim de obter os valores de medida, com tais valores de medidas, se aplica qualquer função estatística com a biblioteca Apache.

Como exemplo, consideremos a consulta *Q'* alterada para ao invés de retornar a frequência retorne a Moda e o Desvio padrão de M1 e M2 respectivamente, assim *Q'* passa ser $Q''=\{a_1, b_1, c_1, Mo(M_1), Dp(M_2)\}$.

Com o resultado da consulta bCqR obtido no exemplo de consulta 1, temos a lista de TIDs *{1, 3}*, assim recuperamos os valores de medidas da memória externa, que pode ser observado na Tabela 6.28, de M1 e cada um dos TIDs e fazemos o cálculo da moda, em seguida recuperamos os valores de medidas da memória externa de M2 e cada um dos TIDs e fazemos o cálculo do desvio padrão.

6.5.2 O algoritmo de consulta bCubing

Com base na discussão realizada sobre consultas na abordagem bCubing, apresentamos o Algoritmo 6.2 para consultas pontuais, intervalo e *inquire*.

Dado uma consulta *Q*, geram-se as três possíveis sub-consultas: *pQ*, *rQ* e *iQ*, onde *pQ* é a porção pontual da consulta *Q* (operador igual), *rQ* a porção com operadores do tipo intervalo *(maior que + menor que + entre + alguns + diferente + similar)* e, por fim, *iQ* a porção de *Q* com operadores do tipo *inquire* (*distinct* e *subcube*).

Algoritmo 6.2 (bCubing Query) processamento de consultas pontuais, intervalo e *inquire*.

Entrada: (1) Um cubo bC conforme descrição feita na Seção 6.2; e (2) uma consulta de usuário *Q*.

Saída: bCqR que inclui as agregações processadas pelo algoritmo de computação e finalizadas pelo algoritmo de consulta, além dos cálculos das medidas solicitadas.

Método:

```
1. for each sub-query{ //pQ, rQ or iQ
2.   for each Di{
3.      for each Di.bidi {
4.         if bidi ∩ [bid1, …, bidn]{
5.          if bidi.tids is not null{
6.            tids ← tids ∪ bidi.tids
             }
           }
         }
       }
     }

7. for each tidi{
      //point query and range query
8.    if tidi ∩ [att1, …, attn]{
9.     RQi ← tidi  ⋂ [att1, …, attn]
      }
      // inquire query
10.   if Ti ∩ [att1, att2, …, attn]{
11.    IQi ← tidi  ⋂ [att1, …, attn]
      }
   }
12. bCqR ← RQi ⋂ IQi;
13. bCqR ← calcMeasures(iCqR,Q,bCubingMeasures)
```

Inicialmente, recupera-se os BIDs (linha 3) associados aos valores de atributos instanciados em cada dimensão. Em seguida é realizada uma interseção partir do conjunto com menor número de BIDs para as possíveis agregações de cada sub-consulta (linha 4). Com o conjunto de BIDs de cada sub-consulta, recupera-se a lista de TIDs para os BIDs obtidos no processo anterior (linha 5). Em seguida, é realizado uma união das listas de TIDs associadas a BIDs (linha 6). Após a união de TIDs, é realizado a interseção entre as listas de TIDs dos valores de atributos de cada possível agregação. A interseção é realizada a partir das listas com menor número de TIDs (linha 8-12), desta forma obtemos os TIDs correspondentes às agregações que formam o resultado da consulta realizada. Por fim, são calculadas todas as medidas definidas em *Q* (linha 13).

6.6 Definição do tamanho dos blocos

Caso a quantidade de tuplas *tBloc* em um bloco for muito grande ou muito pequeno, pode resultar no problema de falta de memória de trabalho. Se o *tBloc* for muito pequeno, por exemplo 1, BID é o mesmo que TID, dessa forma o resultado das consultas pode ser obtido diretamente das ocorrências dos valores de atributos dos BIDs. Essa situação é o pior cenário e equivalente ao Frag-Cubing com as informações em memória principal (BIDs) juntamente com uma cópia em memória externa (TIDs).

Um bloco muito pequeno pode aumentar severamente o tempo de computação e a quantidade de memória principal, pois há poucas tuplas por bloco e muitos blocos para serem armazenados em memória externa. Note que quando há muitos BIDs há aumento de memória principal para computação do cubo de dados.

Em consultas onde o valor de atributo utilizado é pouco frequente e está concentrado em uma determinada partição de *R*, poucos blocos são suficientes para responder uma consulta. Desta maneira, o tempo de consulta pode se beneficiar, quando o cubo de dados é computado com pequenos blocos, ou quando o tamanho definido para o bloco agrupe de forma substancial as partições onde o atributo ocorre em *R*.

Quando poucos blocos são necessários para responder uma consulta, o espaço de busca é reduzido de forma expressiva, além de não requerer maior quantidade de memória principal do que a utilizada para computação. Finalmente, casos onde um valor de atributo é muito frequente, mas com presença uniforme em *R*, blocos muito pequenos não são adequados, pois é necessário a interseção de um número muito grande de BIDs, além do acesso a diversos blocos da memória externa. De uma forma geral, bases reais não possuem qualquer uniformidade, seja no skew, seja na distribuição da localização dos atributos em *R*.

Blocos muito grandes, por outro lado, beneficiam a computação, pois é necessário menor número de operações de entrada e saída, porém podem degradar o tempo de consulta quando valores de atributos pouco frequentes e concentrados em determinadas partições de *R* são requeridos. Nestes casos, são carregados da memória externa muitos valores de atributos e muitos TIDs não necessários em cada bloco. Contudo, em consultas onde valores de atributos são muito frequentes há um ganho com blocos grandes. Note que blocos maiores aumentam a probabilidade de obter as tuplas desejadas com menor número de acessos a memória externa.

A pergunta que surge é se existe um bom equilíbrio entre o tamanho dos blocos e o custo e o desempenho das interseções e o acesso a memória para realizar as diversas consultas possíveis. Na Seção 6.7 experimentos demonstram o impacto de diferentes definições de tamanho de blocos na computação e em consultas pontuais e de subcubo. Assim, estabelecemos uma definição do tamanho do bloco com base no pior caso que é quando temos dimensões com alta cardinalidade, sendo algumas skewed e outras com skew baixo. Também consideramos que consultas possuem normalmente valores de atributos skew em algumas dimensões e que medidas holísticas são desejadas.

Após a computação de um cubo de dados bCubing há em memória principal uma estrutura *bCubingBlocRAM* que tem tamanho $tAuxSize = (\sum_{i=1}^{D} Ci) nTuplas/tBloc$, portanto a quantidade de tuplas por bloco deve ser definida de tal forma que, além de coexistir em RAM

com a estrutura *bCubingBlocRAM,* seja possível a execução de consultas pontuais e de subcubos.

Diante disso, conforme detalhado nos exemplos de consulta 1 e 2 temos alocações de espaço em memória de trabalho definidas como *tPontualRAM* para manter a lista de BIDs que representam os atributos das consultas, sendo *tPontualRAM = nBlocos x 4 bytes*.

Além do espaço para *bCubingBlocRAM* e *tPontualRAM,* é necessário disponibilizar espaço de tamanho *tFixoTuplas* que representa o mapeamento dos valores de atributos das dimensões envolvidas na consulta com o bloco recuperado da memória externa, onde *tFixoTuplas*=$\sum_{i=1}^{D} 4\ x\ Ci\ x\ 4$. Também há o espaço para manter *tVariavelAtt* que possui a lista de TIDS de cada valor de atributo do bloco que é utilizado nas interseções, e o espaço ocupado pelos valores de medida associados com cada TID *tVariavelM*. Sendo *tVariavelAtt = (tBloc x D x 4)* e *tVariavelM= (tBloc x M x (8+4))*, é necessário mais uma lista de tamanho *tPointE* para conter os resultados das interseções de TIDs, sendo *tPointE=nTuplas x 4* e considerando 4 bytes para cada TID. Por fim, é necessário um espaço de tamanho *tResultado* onde o resultado para cada medida é mantido, sendo *tResultado = M x 8*, onde é M é o número de medidas e o resultado é representado por 8 bytes.

Sabendo que o espaço ocupado por *bCubingBlocRAM* mais a memória de trabalho necessária para consultas não pode ultrapassar a memória RAM disponível, conseguimos definir o espaço que um bloco *bSize* pode ocupar, com base no espaço de memória necessário para cada variável definida e resumidas na Tabela 6.43.

TABELA 6.43 – Operadores de consulta da abordagem bCubing

Variável	Descrição
tFixoTuplas	mapeamento dos valores de atributos das dimensões envolvidas na consulta com o bloco recuperado da memória externa
tVariavelAtt	lista de TIDS de cada valor de atributo do bloco que é utilizado nas interseções
tVariavelM	espaço ocupado pelos valores de medida associados com cada TID
tPointE	tamanho da lista que contêm os resultados das interseções de TIDs
tPontualRAM	espaço para manter a lista de BIDs que representam os atributos das consultas
tAuxSize	estrutura em memória principal que mantêm a associação dos valores de atributos com BIDs
tResultado	espaço para manter o resultado para cada medida requerida na consulta

Uma vez que cada tupla tem tamanho *tSize* e *tSize = 4 + (D × 4) + (M×8)*, o tamanho do bloco é *tBloco=bSize/tSize*. Com isso temos que a RAM necessária para não haver swapping é dada por:

RAM≥*tFixoTuplas+tVariavelAtt+tVariavelM+tPointE+tPontualRAM+tAuxSize+tResultado*

Sendo*:*

A = *tFixoTuplas* = $\sum_{i=1}^{D} 4\ x\ Ci\ x\ 4$*;*

B = *tVariavelAtt = (tBloc x D x 4);*

C= tVariavelM = (tBloc x M x (8+4));

D= tPointE = nTuplas x 4;

E=4 bytes;

F=$(\sum_{i=1}^{D} Ci)$ *nTuplas;*

K= tResultado = M x 8,;

T=nTuplas;

Temos:

$$A + tBloc.B + tBloc.C + D\ + \frac{T}{tBloc}.E + \frac{F}{tBloc} + k \leq RAM$$

$$RAM\ -\ A - D - K\ = A'$$

$$tBloc.B + tBloc.C + \frac{T}{tBloc}.E + \frac{F}{tBloc} \leq A'$$

$$\frac{tBloc.(B + C)}{1} + \frac{1}{tBloc}\ .[T.E + F] \leq A'$$

$$\frac{tBloc^2.(B + C) + \ T.E + F}{tBloc} \leq \frac{A'.tBloc}{tBloc}$$

$$tBloc^2.(B + C) -\ A'.tBloc + (T.E + F) \leq 0$$

$$bSize \cong \frac{+A' \pm \sqrt{A'^2 - 4.(B + C).(T.E + F)}}{2.(B + C)}$$

Desta maneira, o tamanho máximo que um bloco pode ocupar em memória sem fazer swapping é $bSize \cong \frac{+A' + \sqrt{A'^2 - 4.(B+C).(T.E+F)}}{2.(B+C)}$ e o valor mínimo é $bSize \cong \frac{+A' - \sqrt{A'^2 - 4.(B+C).(T.E+F)}}{2.(B+C)}$. Assim a quantidade de tuplas em um bloco é definido por: $tBloc \cong \frac{bSize}{tSize}$.

Suponha uma base com: 10^2 dimensões, onde cada dimensão com cardinalidade 10^3, três medidas, número de tuplas igual 10^9 e uma máquina com 6 GB de RAM, e $tSize = 428$ bytes, portanto *bSize* possui, no pior caso, tamanho igual a:

$B = 10^2 . 4$

$C = 3 . (8 + 4) = 36$

$F = 10^2 . 10^3 . 10^9 = 10^{15}$

$A' \cong (6.5\ x\ 10^9) - (10^2 \text{x } 4 \text{ x } 10^4 \text{ x } 4) - (4\ x\ 10^9) - (3\ x\ 8) \cong 2.43\ x\ 10^8$

$A'^2 \cong (5.89\ x\ 10^{18})$

$$bSize \cong \frac{+(2.43.10^8) + \sqrt{(5{,}89.10^{18}) - 4.((10^2.4) + 36).(10^9.4 + 10^{15})}}{2.((10^2 . 4) + 36)}$$

$bSize \cong 5.12\ x\ 10^6$

Com bSize definido, temos para o pior caso a quantidade máxima de tuplas por bloco *tBloc* (*tBloc* crítico), onde $tBloc = \frac{bSize}{tSize} = 11{,}96\ x\ 10^3$.

Para definir a quantidade mínima de tuplas por bloco bSize é definido conforme:

$$bSize \cong \frac{+(2.43.10^8) - \sqrt{(5{,}89.10^{18}) - 4.((10^2.4) + 36).(10^9.4 + 10^{15})}}{2.((10^2 . 4) + 36)}$$

$bSize \cong 0.01$

Assim, como cada tupla tem tamanho $tSize = 428$ bytes, para este cenário seria possível termos blocos com uma única tupla, tal configuração não seria adequada, já que teríamos uma representação em memória principal tal como em memória externa, pois para cada BID em memória principal teríamos um TID associado em memória externa. Desta maneira a representação em memória principal em termos de BID seria igual a abordagem Frag-Cubing o que não possibilitaria tirarmos proveito do segundo nível de indexação proposto por bCubing, para fazer interseções menores.

Caso tBloc definido seja muito maior que tBloc crítico (obtido através da expressão que define bSize), ocorrerá uma perda de desempenho para responder consultas que necessite recuperar muitos blocos, dada a necessidade de swapping, e em alguns casos ocorrerá impossibilidade de responder tais consultas por falta de memória de trabalho, tal comportamento é discutido no Capítulo 7.

O resultado da expressão que possibilita obter bSize e consequentemente tBloc crítico é aproximado, uma vez que é feito simplificações e não é considerado uso de memória do sistema operacional e seus sistemas de gerenciamento de memória, barramentos de memória e arquitetura de hardware por exemplo.

6.7 Experimentos

Com intuito de verificar a eficiência e a escalabilidade das abordagens propostas, realizamos um estudo detalhado com as abordagens bCubing, HIC e Frag-Cubing (LI; HAN; GONZALEZ, 2004), testando os algoritmos de computação e consultas disponibilizadas pelas abordagens. Os algoritmos das abordagens bCubing e HIC foram codificados em Java 64 bits (versão 8.0). Utilizamos a implementação em C++ da abordagem Frag-Cubing disponibilizada pelos autores e compilada para 64 bits (ILLIMINE, 2015).

A codificação dos algoritmos em Java 64 bits utilizam em sua maioria tipos primitivos e estruturas básicas de operações lógicas, nesse sentido tais algoritmos podem ser comparados em termos de desempenho de processamento, para validar tal conceito foi feito testes com algoritmos de iteração em listas de elementos em tipos primitivos em Java 64 bits e C++, comprovando desempenho equivalente de ambas linguagens.

Executamos experimentos com relações que cabem em memória principal e também relações que necessitam de memória externa. Os testes de computação do cubo de dados incluem tempo para operações de acesso a memória externa e tempo de CPU. Tempos de acesso a memória externa são considerados para carregar as relações de entrada para a memória principal. Nos testes de consulta para as abordagens bCubing e HIC, foi considerado o tempo de acesso a partições do cubo em memória externa.

Os algoritmos foram implementados apenas em versões sequenciais. Executamos os algoritmos em dois processadores six-core Intel Xeon com 2,4 GHz cada núcleo, cache de 12 MB e 128 GB de RAM DDR3 1333MHz. O disco em que os experimentos foram executados é SAS 15k rpm com 64MB de cache. O sistema operacional é Windows HPC (High Performance Computing) Server 2008 versão de 64 bits.

Além dos testes executados no ambiente mencionado anteriormente, realizamos experimentos com a abordagem bCubing com o objetivo de descobrir seu comportamento com diferentes tamanhos de blocos e diferentes configurações de máquinas. Desta forma, adotamos uma máquina de menor porte com processador duo-core Intel Centrino com 2,0 GHz cada núcleo, cache de 128 MB e 4 GB de RAM DDR2-SDRAM. O disco em que os experimentos foram executados é ATA 7.5k rpm com 32MB de cache. O sistema operacional é Windows 7 (*Ultimate Professional*) versão de 64 bits e há uma partição Linux Ubuntu 13.10, desta forma é possível avaliar, por exemplo, consumo de memória e tempo de execução com blocos de mesmo tamanho em sistemas operacionais distintos.

Todos os experimentos foram executados cinco vezes, removemos o maior e menor tempo de execução e a média aritmética é calculada para os três tempos de execução restantes.

Para esta Seção, definimos que D é o número de dimensões, C é a cardinalidade de cada dimensão, T é o número de tuplas em uma relação de entrada e S é o skew de dados. As relações de base sintética foram criadas usando gerador de relações fornecido pelo projeto IlliMine.

6.7.1 Respostas de consultas em cubos com diferentes tamanhos de blocos

Realizamos testes para verificar o comportamento da computação de cubos com diferentes tamanhos de blocos, na abordagem bCubing, com relações com T = 20M, 60M e 100M, D = 10, C = 10^4, e S = 2.5. Os testes foram executados nas duas máquinas descritas anteriormente nos três sistemas operacionais indicados, sendo M1 a máquina com dois processadores six-core Intel Xeon com Windows 2008 HPC, M2 e M3 representam máquinas com um processador duo-core Intel Centrino com Ubuntu 13.10 e Windows 7, respectivamente.

As Figuras 6.2, 6.3 e 6.4 ilustram o comportamento dos testes de computação com blocos de diferentes tamanhos, contudo proporcionalmente iguais entre as relações com T = 20M, 60M e 100M, respectivamente.

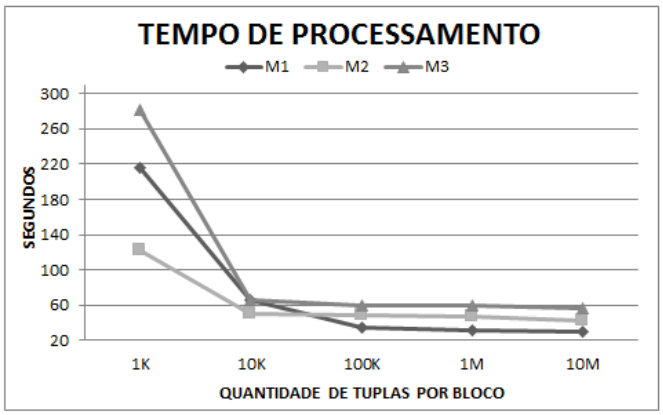

FIGURA 6.2 – Tempo de processamento e consumo de memória para computar cubos com diferentes tamanhos de blocos: D = 10, T=20M, S = 2.5, C = 10^4

Nas duas máquinas testadas com os três diferentes sistemas operacionais observamos o mesmo comportamento, apesar das diferenças de hardware. De uma forma geral, quanto maior for o número de arquivos a serem gravados em memória externa, isto é maior quantidade de tuplas por bloco, menor o tempo de processamento na computação do cubo de dados.

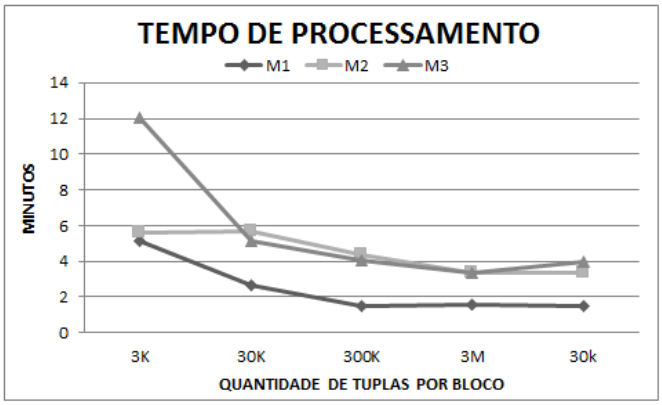

FIGURA 6.3 – Tempo de processamento e consumo de memória para computar cubos com diferentes tamanhos de blocos: D = 10, T=60M, S = 2.5, C = 10^4

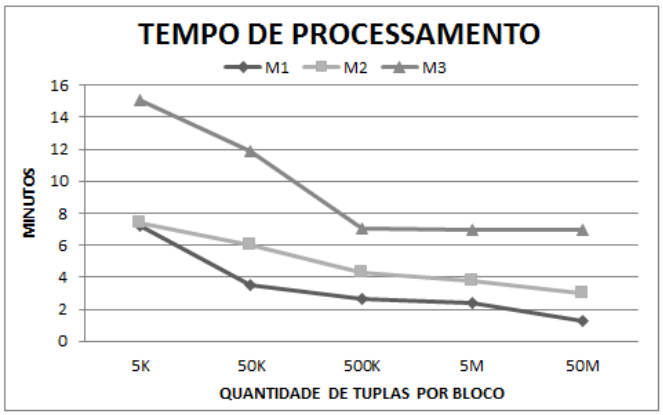

FIGURA 6.4 – Tempo de processamento e consumo de memória para computar cubos com diferentes tamanhos de blocos: D = 10, T=100M, S = 2.5, C = 10^4

Os testes de consulta a cubos com diferentes tamanhos de blocos tiveram o comportamento esperado, uma vez que em consultas onde valores de atributos muito frequentes foram definidos como parâmetros tiveram o melhor desempenho quando há menos blocos, ou seja, maior número de tuplas por bloco conforme é ilustrado nas Figuras 6.5, 6.6 e 6.7 o tempo de processamento e o consumo de memória para responder consultas onde as relações possuem T = 20M, 60M e 100M respectivamente, com dois operadores INQUIRE nas duas primeiras dimensões e um operador EQUAL, sendo EQUAL aplicado ao valor de atributo mais frequente da dimensão.

Neste cenário, o consumo de memória foi estável nas três máquinas, independentemente do tamanho do bloco. Isto se explica uma vez que quando há muitos blocos há também uma lista de BIDs extensa, logo o número de acessos a memória externa para recuperar as listas de TIDs é maior. Além do resultado final das *n* interseções ser uma lista de TIDs muito grande, quando há poucos blocos é necessário carregar da memória externa grandes listas de TIDs para executar as operações de interseção, apesar da lista de BIDs armazenada em memória principal

ser pequena. A frequência do valor de atributo mais frequente na dimensão com operador EQUAL foi 11881198 para T=20M, 35644494 para T=60M e 59405226 para T=100M.

As Figuras 6.8, 6.9 e 6.10 ilustram os testes de consulta utilizando parâmetros pouco frequentes, onde as relações possuem T = 20M, 60M e 100M respectivamente.

Em geral, o desempenho foi melhor quando o cubo de dados foi computado com blocos menores, pois, além de consumir menos espaço de memória de trabalho, foi necessário menor tempo para responder as consultas. Este comportamento ocorre, pois, apesar de existirem listas maiores de BIDs em memória principal, para responder consultas com valores de atributos pouco frequentes é necessário carregar poucas listas de TIDs de pequeno porte e, por se tratar de um valor de atributo pouco frequente, a lista de resultados das interseções também é pequena. A frequência para o valor de atributo menos presente na dimensão em que foi utilizado o operador EQUAL foi 597 para T=20M, 1837 para T=60M e 3102 para T=100M.

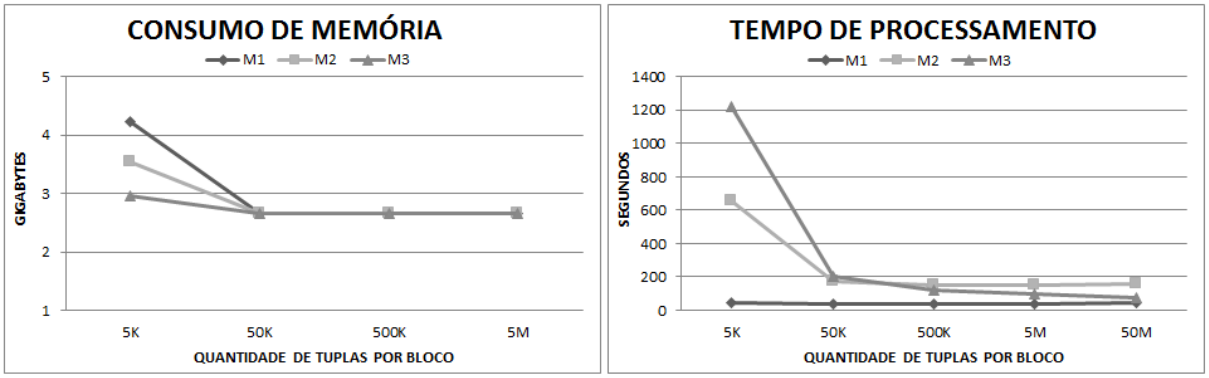

FIGURA 6.5 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=100M, S = 2.5, C = 10^4

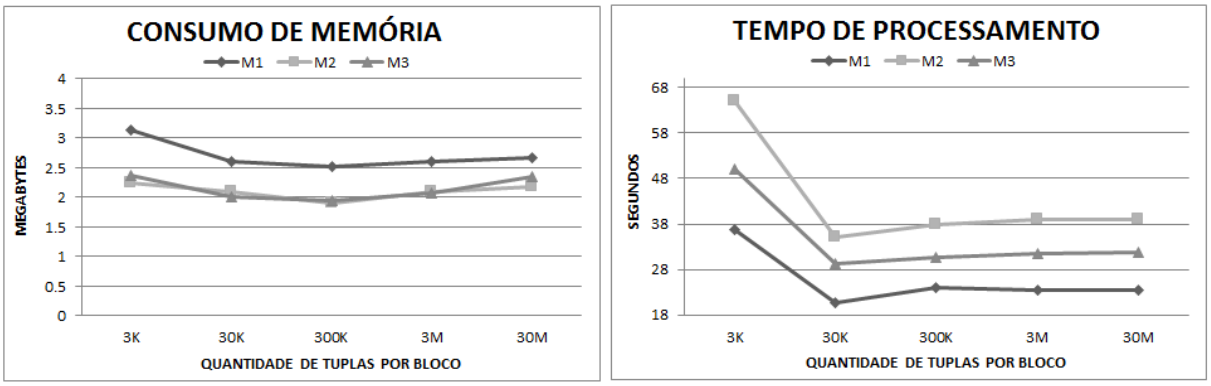

FIGURA 6.6 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=60M, S = 2.5, C = 10^4

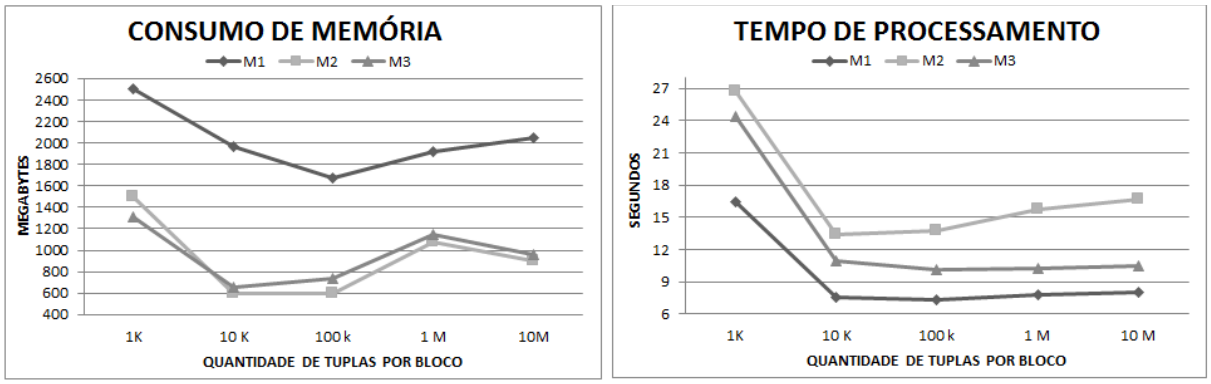

FIGURA 6.7 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=20M, S = 2.5, C = 10^4

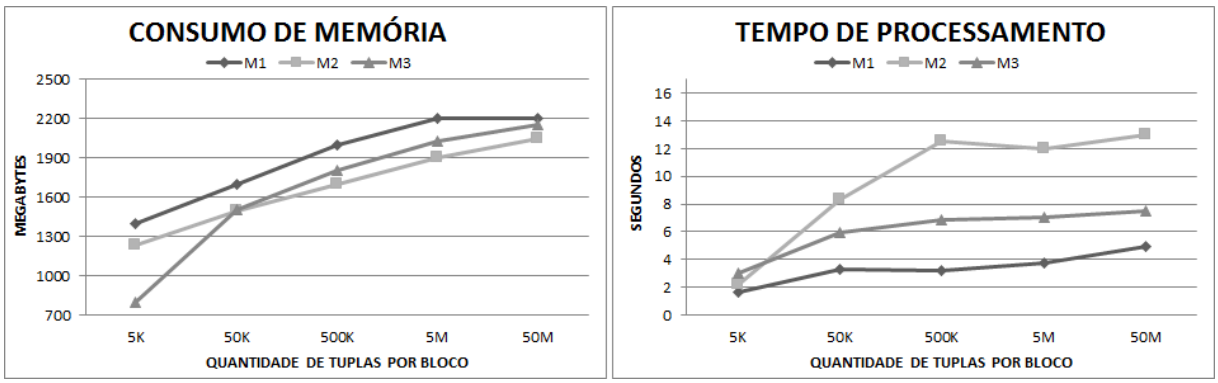

FIGURA 6.8 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=100M, S = 2.5, C = 10^4

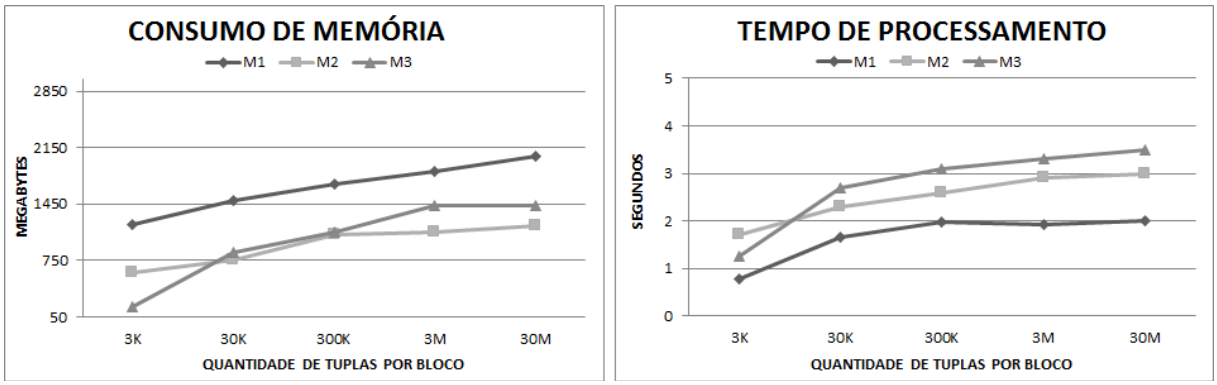

FIGURA 6.9 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=60M, S = 2.5, C = 10^4

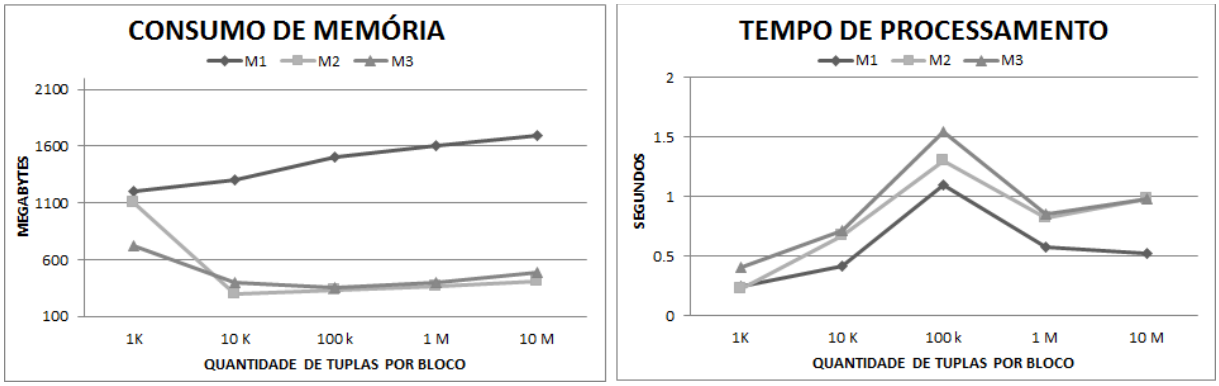

FIGURA 6.10 – Tempo de processamento e consumo de memória para responder consultas com dois operadores INQUIRE e um operador EQUAL, em cubos com diferentes tamanhos de blocos: D = 10, T=20M, S = 2.5, C = 10^4

Além dos testes apresentados, também foram realizados experimentos variando o tamanho de blocos para as três diferentes relações com T=20, 60 e 100M. Assumimos o tamanho do bloco variando entre 0.0005% e 50% de T, desta forma foram testados 25 diferentes tamanhos de blocos para cada relação em cada uma das três máquinas (M1, M2 e M3).

O comportamento foi o esperado, ou seja, em cenários onde os blocos tinham o tamanho próximo a tBloc crítico mínimo (entre 0.0005% e 0.0025% de tuplas da relação de entrada), consultas utilizado valores de atributos muito frequentes nas máquinas M2 e M3 (máquinas com 4GB de RAM) não foram respondidas. Este comportamento é explicado uma vez que tais relações não possuem o pior caso, onde teríamos uma lista com um único TID por valor de atributo em cada bloco recuperado da memória externa.

As relações testadas com blocos de tamanho entre 0.0005% e 0.0025% de tuplas da relação de entrada, possuem listas de TIDs maiores do que o pior caso, assim para responder consultas de subcubos com valores de atributos muito frequentes mais memória de trabalho é necessária para manter as listas recuperadas da memória externa. Na máquina M1 com 128 GB tais consultas foram respondidas, porém foi necessário 4 vezes mais memória do que o utilizado na computação do cubo. Neste caso não foi realizado swapping dada a disponibilidade de memória RAM, porém foi necessário a configuração do *heap* da JVM para trabalhar com a quantidade memória necessária.

Foram realizadas mais de 720 horas de testes variando a quantidade de tuplas por blocos e a execução de diversos tipos de consultas. Após os diversos testes foi observado que para as bases onde o *skew* se encontra entre 0 e 2.5 a quantidade de blocos que determina o melhor equilíbrio entre tempo de consulta e consumo de memória de consultas com valores de atributos muito e pouco frequentes estava na faixa entre 1% e 10% de T considerando a quantidade de

memória disponível. Isso demonstrou a expressão para obter tBloc crítico apesar de gerar um resultado aproximado, devido as simplificações explicadas na Seção 6.6, pode ser utilizada como parâmetro para definição de tBloc com bom equilíbrio entre tempo de consulta e consumo de memória, uma vez que considerando as relações com T=20, 60 e 100M, temos os seguintes tamanhos de blocos respectivamente $tBloc \cong 938.869, 890.991$ e 843.109. Tais valores representam 5%, 1% e 1% das respectivas relações de entrada, considerando a memória disponível.

A fim de validar tBloc crítico obtido a partir da equação proposta na Seção 6.6, uma relação com 10^3 tuplas, cardinalidade igual a 10^3 e 60 dimensões foi criada para representar o pior caso, que é quando para cada lista um valor de atributo ocorre em todas as tuplas de todos os blocos. Para tal relação, tBloc crítico é ≅ 16, considerando a memória disponível 1 GB. Tal tBloc foi o que apresentou melhor desempenho em consultas de subcubo, mesmo sendo possível realizar consultas com blocos de até 22 tuplas.

A relação real e as relações sintéticas criadas, com a abordagem bCubing comparativamente com as abordagens HIC e Frag-Cubing, não possuíam o pior caso. Desta maneira fixamos a quantidade de tuplas por bloco em 10% de T, já que este percentual de tuplas se mostrou factível de ser configurado pois não ocupou toda memória RAM disponível na máquina com 128 GB.

6.7.2 Computação de relações com diferentes números de tuplas

Testes variando quantidade de tuplas tiveram comportamento linear e estável nas três abordagens. Foram utilizadas relações com T = 1M, 25M, 50M, 75M e 100M, D = 60, C = 10^4, e S = 0. Em geral bCubing utilizou entre 6 e 14 vezes menos memória RAM do que Frag-Cubing e precisou de 3 a 5 vezes menos memória do que HIC, conforme ilustra Figura 6.11.

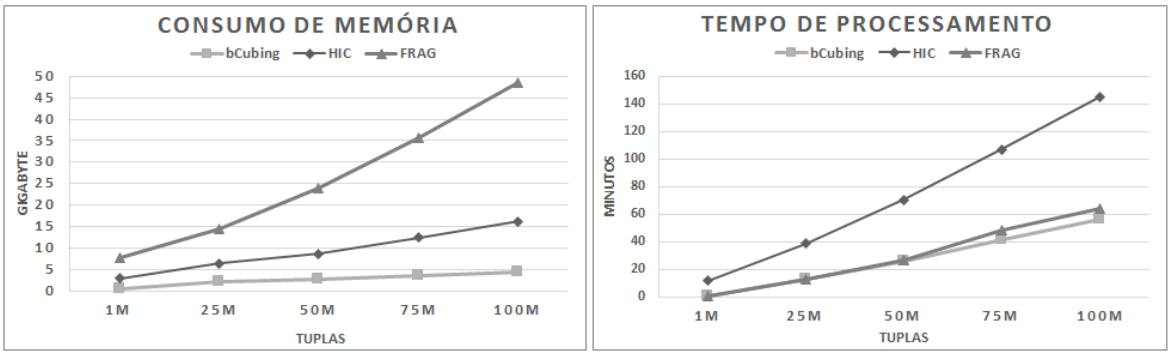

FIGURA 6.11 – Tempo de processamento e consumo de memória para computação de cubo variando o número de tuplas com as abordagens bCubing, HIC e Frag-Cubing: D = 60, S = 0, C = 10^4

Os tempos de execução bCubing para computar cubos de dados são praticamente iguais ao Frag-Cubing para as relações com T=1M, 25M e 50M e por volta de 15% mais rápido conforme aumenta o número de tuplas na relação. Este é um resultado promissor, pois demonstra que bCubing indexa relações com elevado número de tuplas sem requerer grandes quantidades de memória principal. HIC é, em média, três vezes mais lento do que bCubing e Frag-Cubing. A estratégia de dois níveis de índices de bCubing faz acesso à memória externa de forma eficiente e gera interseções com conjuntos menores de itens, portanto consegue responder consultas mais rápido.

6.7.3 Computação de relações com diferentes números de dimensões

Executamos testes variando a quantidade de dimensões em dois grupos de relações, sendo o primeiro grupo com relações definidas como D = 30, 60, 90, 120 e 150, T = 10^7, S = 0 e C = 10^4. Tais testes são ilustrados pela Figura 6.12.

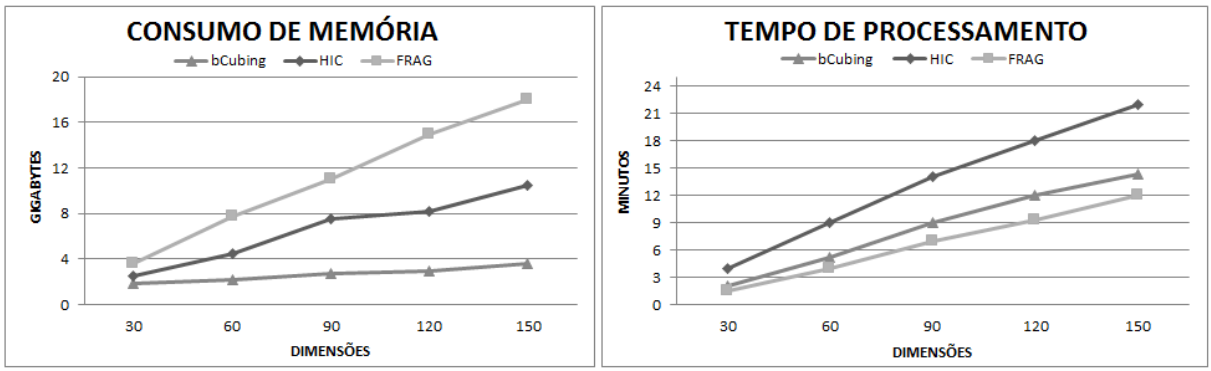

FIGURA 6.12 – Tempo de processamento e consumo de memória para computação de cubo variando o número de dimensões com as abordagens bCubing, HIC e Frag-Cubing: D = 30, 60, 90, 120 e 150, T = 10^7, S = 0 e C = 10^4

Nestes experimentos, o consumo de memória foi estável para todas as abordagens, contudo uma vez que a abordagem Frag-Cubing faz alocação de memória contígua e armazena todos vetores de TIDs em memória principal, esta necessita de 50% a 400% mais memória do que bCubing. HIC utilizou de 1.5 a 2.5 vezes mais memória do que bCubing, pois mantêm listas de TIDs dos atributos mais frequentes em memória principal e bCubing mantêm apenas listas de BIDs em memória principal.

Como pode ser observado na Figura 6.12, a abordagem Frag-Cubing tem desempenho melhor do que a abordagem bCubing, sendo entre 19% a 25% melhor, devido ao fato de não ter custo de acesso a memória externa. Já a abordagem HIC tem desempenho inferior a abordagem bCubing, sendo 50% a 89% inferior. O desempenho inferior de HIC em relação a bCubing se dá pelo maior número de acesso a memória externa feita quando existe um maior número de relações, já que existirão mais valores de atributos mantidos em memória externa.

Na Figura 6.13, é ilustrado o resultado dos experimentos do segundo grupo de relações com alta-dimensionalidade. Este grupo possui as mesmas relações apresentadas na Subseção 5.4.2 com D = 240, 480, 720, e 960, T = 10^7, e C = 10^4. A abordagem bCubing apresentou um consumo de memória expressivamente menor do que Frag-Cubing e HIC à medida que o número de dimensões cresce. A abordagem bCubing computou cubos de dados com 720 e 960 dimensões, contudo a abordagem Frag-Cubing não conseguiu computá-los devido a sua alocação contígua de memória. A abordagem bCubing teve desempenho superior a abordagem HIC, sendo entre 70% e 250% mais veloz e consumindo entre 70% e 300% menos memória.

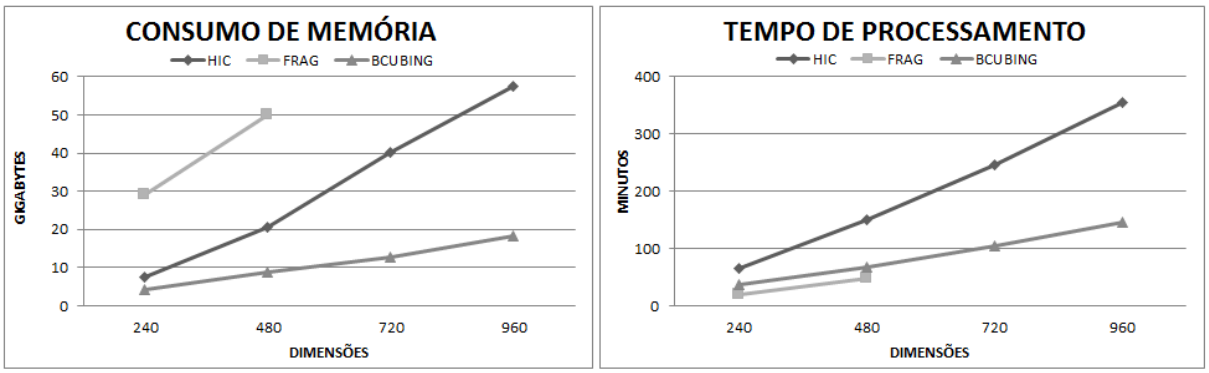

FIGURA 6.13 – Tempo de processamento e consumo de memória para computação de cubo variando o número de dimensões com as abordagens bCubing, HIC e Frag-Cubing: D = 240, 480, 720, e 960 T = 10^7, S = 0 e C = 10^4

6.7.4 Computação de relações de diferentes skew

Também testamos a uniformidade dos dados, avaliando relações com diferentes skew: S = 0, 0,5, 1, 1,5, 2, e 2,5, D = 15, T = 10^7, e C = 10^4.

Na Figura 6.14, são ilustrados o consumo de memória e tempo de execução para cada relação computada. Podemos observar na Figura 6.14 que quando o skew de uma relação é alto, todas abordagens fazem a computação do cubo de dados mais rapidamente. Isso é explicado uma vez que é necessária a alocação de menos listas de TIDs. Com S=2.5 bCubing e HIC fazem menos operações de acesso a memória externa, já que em uma relação com alto skew, em geral, a cardinalidade é reduzida, logo existem menos listas de TIDs para serem armazenadas em memória externa. A abordagem HIC leva de 1,6 a 3 vezes mais tempo do que a abordagem bCubing e Frag-Cubing que possuem desempenho muito similares.

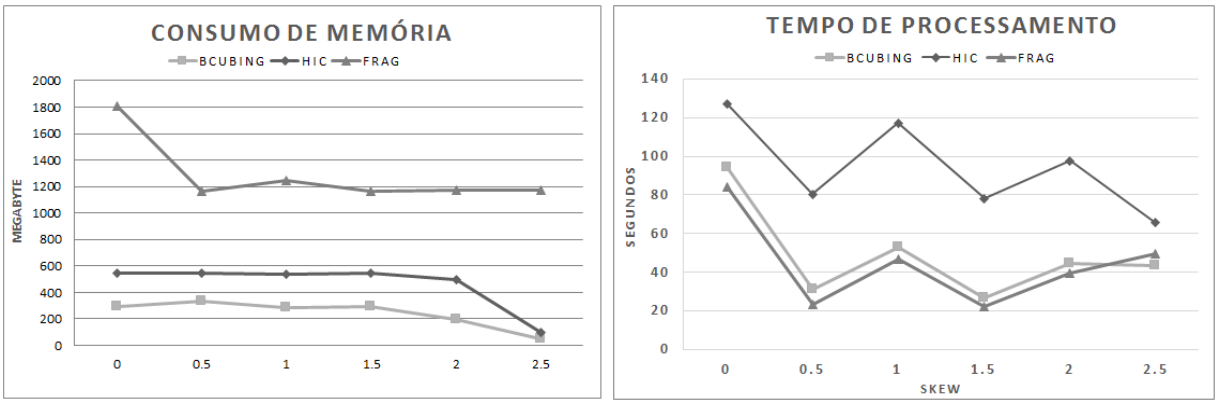

FIGURA 6.14 – Tempo de processamento e consumo de memória para computação de cubo variando o skew com as abordagens bCubing, HIC e Frag-Cubing: D = 15, T = 10^7, C = 10^4

Relações skewed onde determinados valores de atributos estão presentes em quase todas as tuplas são comuns em aplicações reais. Como HIC armazena valores de atributos frequentes na memória principal, este consome mais memória principal e menos operações de acesso a memória externa, tendo um desempenho melhor em relações com skew alto do que relações com skew mais baixo.

O melhor desempenho de bCubing em relação a HIC ocorre devido a estratégia de particionamento do cubo, em memória externa, ser mais eficiente, já que um bloco em bCubing representa um conjunto de tuplas, em HIC é feito acesso a memória externa para registrar as ocorrências de cada um dos valores de atributo com a frequência acumulada igual ou superior a frequência acumulada crítica. Diante disso, como relações skewed podem ter valores de atributo com frequência próximos a 100% em uma partição do cubo de dados a abordagem HIC tende a realizar menos acesso a memória externa, já que haverá menos valores de atributos a

serem mantidos em memória externa, por isso a diferença de desempenho diminui conforme o skew das relações aumentam.

6.7.5 Relações massivas

Foram realizados testes com seis relações massivas variando o número de tuplas, sendo as relações compostas de D = 60, T = 200M, 400M 600M, 800M, 1000M e 1200M, S=0 e C = 10^4. Não temos conhecimento de um estudo com relações deste porte na literatura OLAP, DW e correlatos.

Conforme apresentado na Subseção 5.4.5, não foi possível computar uma relação com 200 milhões de tuplas e 60 dimensões sem swap do sistema operacional com a abordagem Frag-Cubing, então para os testes com as relações massivas habilitamos operações de swap.

A Figura 6.15 ilustra o consumo de memória para cada relação computada pelas abordagens Frag-Cubing, bCubing e HIC. Realçamos o consumo de memória RAM e memória externa com as operações de swap da abordagem Frag-Cubing, portanto para a relação com T=200M a abordagem Frag-Cubing utilizou, além dos 128 GB de RAM disponíveis, mais 2GB de memória externa para operações de swap. Neste mesmo cenário, a abordagem bCubing utilizou apenas 16.5 GB RAM para indexação e a abordagem HIC necessitou de 30 GB para computar a mesma relação.

De uma forma geral, a abordagem bCubing utilizou entre 30% e 43% menos memória RAM do que a abordagem HIC. Para a relação com 1.2 bilhão de tuplas Frag-Cubing utilizou, além dos 128 GB de RAM disponíveis na máquina, mais 48 GB de memória externa para operações de swap. A abordagem bCubing utilizou apenas 84 GB de memória RAM para computar a mesma relação e HIC utilizou os 128 GB de memória RAM disponíveis.

Nos testes realizados com a abordagem bCubing os blocos foram configurados para terem tamanho igual a 10% da relação de entrada, contudo testes com blocos com apenas 2.5% das tuplas tiveram resultados com cerca de 30% menos memória para computação do cubo de dados.

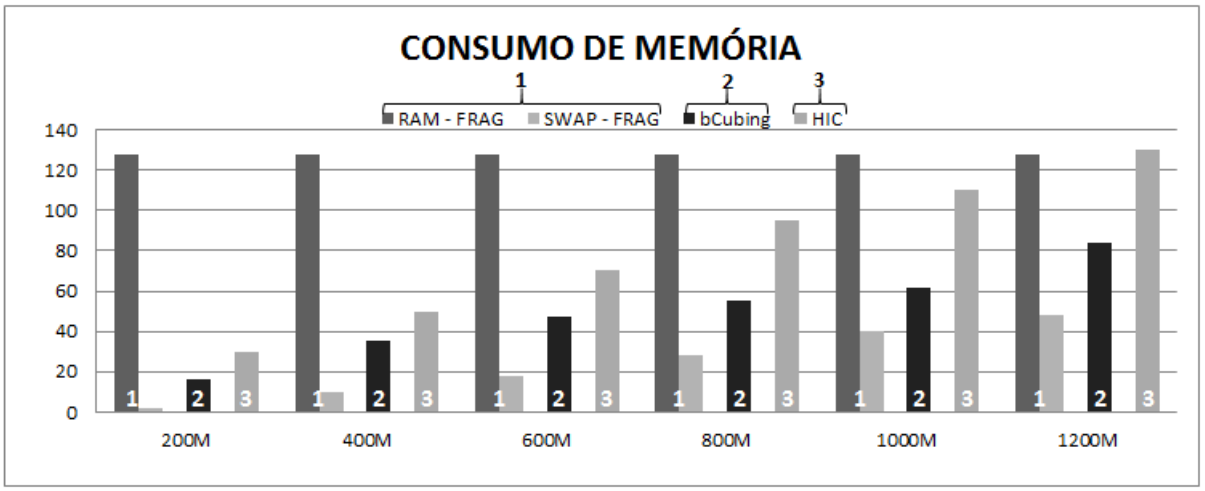

FIGURA 6.15– Consumo de memória para relações massivas variando os números de tuplas com as abordagens bCubing, HIC e Frag-Cubing: D = 60, T = 200M, 400M 600M, 800M, 1000M e 1200M, C = 10^4.

Na Figura 6.16, é ilustrado que o tempo de processamento para computação das relações massivas foi linear e estável nas abordagens bCubing e HIC, porém teve desempenho inviável na abordagem Frag-Cubing devido ao número de swaps do sistema operacional.

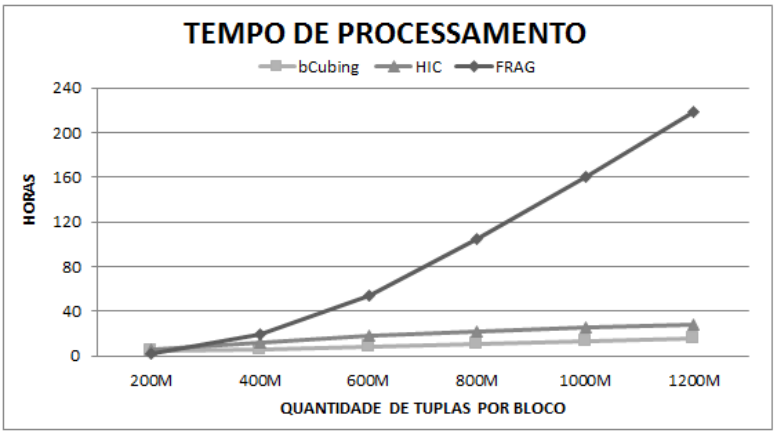

FIGURA 6.16 – Tempo de processamento para relações massivas com diferentes números de tuplas com as abordagens Frag-Cubing, bCubing e HIC: D = 60, T = 200M, 400M 600M, 800M, 1000M e 1200M, C = 10^4

A abordagem bCubing teve desempenho entre 48% e 65% melhores do que a abordagem HIC, uma vez que o número de gravações em memória externa é menor em bCubing, pois cada bloco contém todos valores de atributos da partição do cubo dividido por dimensões. Sendo assim, o número de dimensões *d* na computação de um cubo com a abordagem bCubing requer *d x nBlocos* gravações em memória externa. Sendo *nAttCritico* o número de valores de atributos com frequência crítica na abordagem HIC, teremos *d x nAttCritico* gravações em memória externa. Em geral, entre 75% e 90% dos valores de atributos possuem frequência crítica em uma relação com S=0, o que gera um elevado número de gravações de pequenos arquivos na abordagem HIC.

A abordagem bCubing foi mais veloz do que a abordagem Frag-Cubing, sendo 60% mais veloz quando T=200M, 72% mais veloz quando T=400M, 85% quando T=600M, 89% quanto T=1000M e 92% quando T=1200M. Os testes com relações massivas demonstram a eficiência da abordagem bCubing quando é desejado computar cubos de dados com elevado número de tuplas, alta-dimensionalidade e alta cardinalidade por dimensão.

6.7.6 Consultas em cubos massivos

Realizamos consultas nas relações massivas previamente computadas. Em geral, consultas com o operador subcubo foram respondidas pela abordagem Frag-Cubing somente na relação com T = 200M e mesmo assim tal abordagem teve desempenho inferior a abordagem bCubing, sendo, em média, 25% mais lenta. Nas demais relações não foi possível obter resposta às consultas subcubo por falta de memória contínua para alocação dos vetores para interseção.

A abordagem HIC respondeu somente consultas com dois operadores subcubo nas relações com T=200M, 400M e 600M. As consultas na abordagem HIC foram respondidas, em média, gastando 50% mais tempo do que bCubing. Quando utilizado apenas um operador subcubo a abordagem HIC respondeu tais consultas com desempenho 30% inferior a abordagem bCubing. O pior desempenho da abordagem HIC em relação a abordagem bCubing está diretamente relacionado ao número de acessos a memória externa e também a necessidade de realizar interseções entre listas de TIDs extremamente grandes nas consultas com dois operadores subcubo.

A abordagem bCubing respondeu consultas no cubo de dados com mais de 10^9 tuplas, utilizando dois operadores subcubo e um operador EQUAL em menos de 4 minutos. Nesta consulta foi instanciado valor de atributo de frequência mediana, assim como foram calculadas três medidas holísticas de forma exata: desvio padrão, mediana e moda. Vale ressaltar que neste teste, a abordagem bCubing necessitou utilizar apenas 2GB além do utilizado para a computação do cubo de dados.

Os cálculos de medidas holísticas não apresentaram impacto no desempenho das consultas e a quantidade de medidas holísticas por consulta também não geraram impacto conforme é ilustrado pela Figura 6.17. Experimentos mostraram que bCubing é em torno 10% a 15% mais lento do que a versão com medidas algébricas e distributivas igualmente calculadas de forma exata.

Na Figura 6.17, observamos que quando é calculado somente o desvio padrão ou, além do desvio padrão, a mediana ou se ainda adicionarmos a moda não temos impacto significativo

no desempenho. Vale ressaltar que cada cálculo foi realizado para uma medida diferente, o que representa fazer acesso a memória externa aos valores das diferentes medidas.

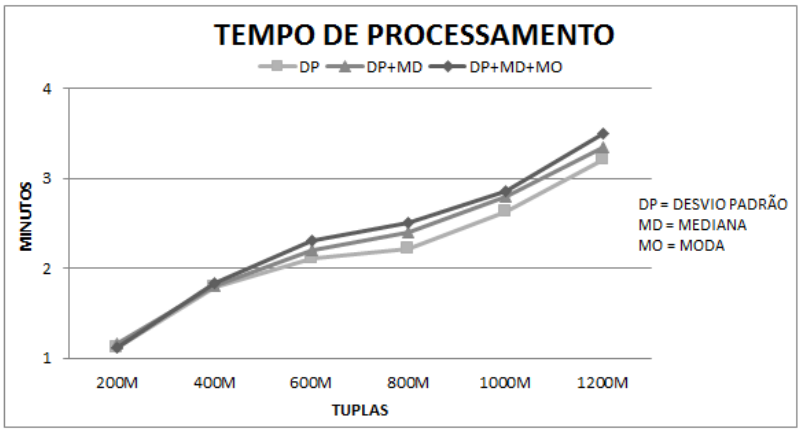

FIGURA 6.17 – Tempo de processamento para calcular medidas holísticas em relações massivas com diferentes números de tuplas com a abordagem bCubing D = 15, T = 200M, 400M 600M, 800M, 1000M e 1200M, C = 10^4

6.7.7 Experimentos com relação real

bCubing foi também experimentada com a relação que possui registros para comparações de padrões de links de páginas web. Tal relação foi obtida a partir do repositório de aprendizagem de máquina UC IRVINE (2015). Esta relação contém T = 4.599.306 D = 12. As cardinalidades são (96.633, 96.636, 55, 31, 100, 35, 2, 3, 3, 3, 3, 2). A computação do cubo de dados para esta relação utilizando bCubing gastou, em média, 12 segundos, enquanto a abordagem Frag-Cubing computou o mesmo cubo em 8.5 segundos e a abordagem HIC levou 1,3 minutos. O elevado número de acessos a memória externa feito por HIC, devido a alta cardinalidade das três primeiras dimensões da relação, ocasiona o baixo desempenho da abordagem para a computação desta relação. As três abordagens utilizaram em média 550KB de RAM.

Uma consulta inquire com duas dimensões e considerando as dimensões com cardinalidades 96.633 e 96.636 levou, em média, quatro horas para ser executada e consumiu 1.5 GB de RAM com a abordagem bCubing, já com a abordagem HIC foi necessário, em média, seis horas para executar a mesma consulta e 2 GB de RAM. Executamos a mesma consulta na abordagem Frag-Cubing, porém mesmo utilizando 128 GB de RAM não foi possível obter resultado. Diante disto, refizemos a consulta e deixamos o sistema operacional realizar swap. Desta forma, além de utilizar os 128 GB de RAM disponíveis foi necessário o uso de mais 245 GB de disco para responder tal consulta, demandando 8 horas de processamento.

Uma consulta inquire nas três dimensões com as cardinalidades mais elevadas (96.633, 96.636, 55) levou por volta de oito horas e consumiu 7.5 GB de RAM na abordagem bCubing, enquanto a abordagem HIC demandou mais de 11 horas e utilizou cerca de 10 GB de RAM. Frag-Cubing também não respondeu tal consulta.

Uma consulta inquire em três dimensões com cardinalidades baixas (3, 3, 2) foi executada pela abordagem HIC em um minuto e não necessitou memória além da utilizada na computação do cubo. Neste experimento, a abordagem Frag-Cubing teve um desempenho melhor, executando em 8 segundos apenas e utilizando 1.2 GB de RAM. Nota-se claramente que a abordagem Frag-Cubing é mais veloz para consultas com poucas agregações e que, consequentemente, consomem pouca memória de trabalho. A abordagem bCubing necessitou de 18 segundos para executar a mesma consulta sem utilizar mais memória do que a utilizada para computação do cubo de dados.

Exploramos nos testes mais algumas consultas de subcubo onde utilizamos dois parâmetros inquire para as dimensões com cardinalidade (96.633, 96.636) e variamos um valor de atributo na dimensão com cardinalidade 55. No primeiro teste utilizamos um valor de atributo com frequência igual a 2 como filtro, o que gerou como resposta um subcubo com muitas agregações. A abordagem Frag-Cubing respondeu tal consulta em 40 segundos, utilizando 73 GB de memória RAM, porém a mesma consulta na abordagem bCubing foi respondida em menos de 1 segundo e consumindo apenas 671 MB de memória RAM.

Foi utilizado um valor de atributo com frequência de 11567 que gerou um subcubo com 14802 agregações onde Frag-Cubing precisou de 2 minutos e 81 GB de RAM para responder à consulta e a abordagem bCubing precisou somente de 38 segundos e 700 MB de memória RAM.

Uma consulta que utilizou como parâmetro um valor de atributo cuja frequência era 56896 exigiu que a abordagem Frag-Cubing alocasse 87 GB de RAM e levasse 6 minutos para gerar as 57922 agregações, enquanto a abordagem bCubing gerou as mesmas agregações em apenas 2 minutos utilizando apenas 760 MB de RAM.

Por fim, executamos um cenário com um valor de atributo com frequência 329 onde a resposta da consulta é composta por um subcubo com 627 agregações. Neste teste Frag-Cubing respondeu tal consulta em 28 segundos utilizando 74 GB de memória RAM, enquanto a abordagem bCubing necessitou de apenas 6 segundos utilizando 634 MB de memória RAM.

Nos testes descritos a abordagem HIC utilizou entre 50% e 100% menos memória do que a abordagem Frag-Cubing, porém respondeu as consultas entre três e cinco vezes mais tempo do que Frag-Cubing.

A consulta onde foi utilizado uma única inquire na dimensão com cardinalidade 96.633 fez bCubing gerar todas as possíveis agregações em apenas 2 segundos e utilizando 450 MB de RAM, já Frag-Cubing levou 3 minutos e alocou 607 MB de RAM.

6.8 Resumo

Neste capítulo, detalhamos a segunda contribuição desta tese. A abordagem bCubing possui um sistema híbrido de memória para armazenar partições de cubos de dados na memória externa; portanto cubos de dados com 60 dimensões e 1.200M tuplas podem ser computados. A abordagem bCubing implementa o conceito de índices invertidos em duas fases, utilizando como base o conceito de tuplas invertidas, proposto na abordagem Frag-Cubing, em conjunto com blocos de índices invertidos, proposto neste trabalho.

Nossos experimentos demonstraram que bCubing é uma solução eficiente, uma vez que o tempo de processamento e o consumo de memória são lineares quando há aumento do número de dimensões e quantidade de tuplas. Em geral, o consumo de memória da abordagem bCubing foi inferior ao consumo de memória das abordagens HIC e Frag-Cubing. Além disto, bCubing foi mais eficiente para responder consultas pontuais e consultas com operadores subcubo em bases massivas.

Fizemos um amplo estudo sobre o comportamento da abordagem bCubing em diferentes cenários, variando quantidade de tuplas, dimensões e cardinalidades, além de variar o skew, portanto demonstramos o desempenho em geral superior de bCubing para computar cubos parciais e responder diversas consultas multidimensionais complexas. Nos testes, apresentamos resultados para relações massivas, que possuem de 200M a 1.200M de tuplas com 60 dimensões e com cardinalidade 10^4. Desconhecemos testes com relações desta grandeza na literatura sobre OLAP e DW.

Por fim, demonstramos ainda como bCubing possibilita o cálculo de medidas holísticas de forma exata, algo ainda não realizado por outras abordagens sequenciais. Apresentamos o pequeno impacto de medidas holísticas em consultas, mesmo quando o número de tuplas é extremo.

No próximo capítulo, realizamos discussão sobre temas não detalhados nesta tese que merecem algum destaque. Apontamos limitações em técnicas comuns e amplamente adotadas, assim como sugerimos alternativas de soluções para extensões de bCubing.

7 Análise e Discussões

Neste capítulo, apresentamos uma discussão sobre a aplicabilidade da abordagem bCubing, assim como outras abordagens, em diferentes cenários. A Tabela 7.1 ilustra os possíveis cenários experimentados com as abordagens Frag-Cubing, qCube, H-Frag, HIC e bCubing. As abordagens foram avaliadas quanto à computação de cubos, as quatro atualizações possíveis numa relação, consultas pontuais, consultas complexas com o operador subcubo, o cálculo exato ou aproximado de medidas holísticas e a capacidade de execução em máquinas com pouca quantidade de memória principal (RAM). As relações são classificadas quanto a sua cardinalidade, dimensionalidade, número de tuplas e *skew*.

TABELA 7.1 – Cenários onde cada abordagem é recomendada (menores consumo de memória e tempo de execução)

	Relações						
	Baixa Cardinalidade	**Baixa Dimensionalidade**	**Baixo Nº de Tuplas**	**Alta Cardinalidade**	**Alta Dimensionalidade**	**Elevado Nº de Tuplas**	**Skewed**
Computação	Frag-Cubing	Frag-Cubing	Frag-Cubing	Frag-Cubing / bCubing	bCubing	bCubing	bCubing
Atualização (4 tipos)	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing
Medidas Holísticas	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing
Consultas Pontuais	Frag-Cubing/qCube	Frag-Cubing/qCube	Frag-Cubing/qCube	HIC	HIC	HIC	HIC
Consultas a subCubos	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing
Consultas de intervalo	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing
Pouca RAM disponível	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing	bCubing

A computação de cubos pequenos favorece abordagens que utilizam apenas memória principal, devido ao custo de acessar a memória externa, desta forma Frag-Cubing é mais eficiente que as demais. Foram realizados testes com a abordagem Frag-Cubing e a abordagem bCubing para computar um cubo de dados com apenas 10^6 tuplas, 10 dimensões com cardinalidade igual a 100 em cada dimensão. A abordagem bCubing consumiu duas vezes mais memória e levou o dobro do tempo para computar tal cubo de dados.

Relações massivas impõem alta dimensionalidade e elevado número de tuplas. A indexação ou computação de cubos massivos é realizada pela abordagem bCubing de forma eficiente, pois esta se vale de um sistema híbrido de memória e dois índices invertidos e não

apenas um. Para computar um cubo de dados massivo, possível de ser armazenado apenas em memória principal, a abordagem Frag-Cubing foi de 3 a 5 vezes mais rápida do que a abordagem HIC e apenas 50% mais rápida do que a abordagem bCubing. Conforme o número de dimensões aumenta a eficiência da abordagem Frag-Cubing frente a abordagem bCubing diminui, chegando a ser de apenas 15%. Quando o limite de memória principal é atingido a abordagem bCubing passa a ser muitas vezes mais veloz do que a abordagem Frag-Cubing. Como não é possível saber antecipadamente a quantidade de memória requerida em um cubo de dados, principalmente quando consideramos as inúmeras operações de atualização que certamente ocorrerão, consideramos a abordagem bCubing a mais recomendada para cubos massivos. Os resultados da Figura 6.15 reforça esta diretriz.

A alta cardinalidade gera o aumento na combinação de valores de atributos numa relação ou diminui o grau de repetição de tuplas na relação, desta forma a cardinalidade sozinha não é responsável pelo aumento do tamanho de um cubo de dados. Tanto as abordagens Frag-Cubing, qCube e bCubing não se mostraram sensíveis a variação da cardinalidade nas relações.

Há quatro tipos de atualizações definidas neste trabalho. A tecnologia OLAP, assim como as inúmeras abordagens para cubos multidimensionais, muitas vezes simplesmente ignora avaliar o impacto de atualizações em suas inovações. Realizamos estudos com bCubing e HIC avaliando o impacto das atualizações e bCubing se mostra o mais promissor, pois considera utilização de memória externa para armazenar as recorrentes atualizações. Considerando a possibilidade do ciclo de vida de um cubo de dados ser longo, inúmeras atualizações podem ocorrer em seu ciclo de vida, portanto se justifica ainda mais um sistema de memória híbrido como o bCubing.

Medidas holísticas é outro tema ainda pouco explorado pela tecnologia OLAP, assim como pelas abordagens científicas para cubo de dados. Normalmente, o cálculo de tais medidas é aproximado e o catálogo de medidas holísticas apoiadas é pequeno. bCubing usa o cálculo de medidas holísticas de forma exata ou aproximada gastando, em média, apenas 10% mais tempo do que as mesmas consultas adotando medidas algébricas simples (Ex. COUNT, SUM). O cálculo da MODA num cubo com 10^9 tuplas demonstra a viabilidade do conceito de índices invertidos em dois níveis da abordagem bCubing, pois tal conceito garante que poucos blocos são necessários para responder consultas complexas a partir de relações com bilhões de tuplas.

Consultas pontuais a cubos pequenos demandam pouco processamento e memória, portanto bCubing, que faz uso de memória externa, nunca é mais veloz do que Frag-Cubing e qCube. HIC somente é mais rápido do que Frag-Cubing quando os valores de atributos consultados são frequentes, pois estes são armazenados em memória principal. Quando os

valores de atributos consultados com HIC são pouco frequentes e estão em memória externa Frag-Cubing tem melhor desempenho.

Para responder consultas pontuais, a abordagem bCubing teve os piores resultados, chegando a ser 10 vezes mais lenta do que a abordagem Frag-Cubing. O resultado ruim é devido ao tempo de acesso a memória externa ser maior do que o tempo gasto para se fazer poucas interseções, já que são consultas pontuais que normalmente consomem menos de 1 segundo. Consultas pontuais em cubos massivos com valores de atributos frequentes gastam muita memória e processamento, portanto muitos blocos são necessários para se responder tais consultas, o que torna a abordagem bCubing pior do que HIC que é a abordagem com melhor desempenho para tal cenário.

Cenários onde a capacidade da memória principal é atingida fazem Frag-Cubing demandar swap do sistema operacional, que não foi projetado para as otimizações necessárias a cada abordagem, desta forma Frag-Cubing passa a se degradar cada vez mais e se tornar inviável.

O operador subcubo é extremamente útil no processo decisório, pois retorna sumarizações organizadas como hierarquias a partir de um conjunto de dimensões instanciadas. O tomador de decisão passa a obter visões com as mesmas propriedades do cubo original, formado por toda a relação de tuplas, porém tal visão, denominada subcubo, possui apenas algumas dimensões instanciadas (Ex. dimensão A igual a_1) ou atribuídas para formarem novas agregações (dimensões onde utiliza-se operadores de intervalo ou *inquire*). bCubing é a mais veloz e mais promissora tanto para responder consultas de subcubos em cubos pequenos quanto para cubos massivos. Este resultado é decisivo no contexto OLAP, pois consultas rápidas com resultados hierarquizados e sumarizados é a essência de tal tecnologia.

No mesmo sentido, operadores de intervalo são fundamentais ao processo decisório. Normalmente, as abordagens implementam operadores igual e *inquire*, mas quase nunca operadores de intervalo. HIC e bCubing implementam tais operadores, contudo bCubing é mais promissora, pois é mais veloz e consome menos memória do que HIC em tais consultas.

As abordagens HIC e bCubing consomem pouca memória principal e Frag-Cubing aumenta sua utilização rapidamente à medida que o número de tuplas ou dimensões aumenta. bCubing ocupa cerca de 6 vezes menos memória do que a abordagem Frag-Cubing, demonstrando a viabilidade de utilizar máquinas de baixo custo para computação e consulta de cubos de dados massivos. Para ilustrar, há o exemplo de um cubo de dados com 10^8 tuplas e 10 dimensões onde bCubing respondeu consultas utilizando operadores subcubos, gerando 5204 agregações que ocuparam apenas 2.7 GB de RAM.

A definição da quantidade de tuplas em cada bloco tem impacto direto no desempenho da computação e das consultas na abordagem bCubing. Na Seção 6.6 apresentamos uma função quadrática para calcular a quantidade máxima e a quantidade mínima de tuplas em um bloco no pior caso e considerando as informações muitas vezes indisponíveis: tuplas, dimensões, cardinalidade e a memória RAM disponível. Como discutido anteriormente, definir um tamanho ideal de bloco depende da máquina e muitas vezes também das características adjacentes ao sistema operacional. O domínio do problema, o tipo de consulta a ser realizada em determinadas dimensões e a experiência do usuário também precisam ser levadas em conta.

Com base em vários experimentos em máquinas com diferentes tamanhos de memória, observamos que em geral o desempenho (tempo) tanto para computação do cubo quanto para consulta é pior para tamanho de bloco pequeno (entre 0.0005% e 0.0025% da relação de entrada para as máquinas experimentadas). A razão é a enorme quantidade de acesso a memória externa e interseções necessárias para responder consultas de subcubos.

O desempenho se estabiliza por um intervalo razoavelmente grande (entre 1% e 10%, nos experimentos realizados). Este intervalo está associado a quantidade total de tuplas da relação de entrada e tamanho de memória disponível. A razão é que neste intervalo existe um equilíbrio entre a quantidade de acesso a memória externa, o número de interseções e o tamanho das listas de TIDs que são intersecionadas.

Diante do comportamento observado em nossos experimentos considerando o tamanho de memória disponível de 128 GB, consideramos os tamanhos de blocos que apresentaram melhor custo benefício em termos de: (i) tempo de computação, (ii) consultas utilizando valores de atributos pouco e muito frequentes como parâmetro e (iii) consumo de memória; foram os que tinham a quantidade de tuplas entre 1% e 10% da relação de entrada. Tais percentuais podem ser explicados pelo tBloc crítico que pode ser obtido através da expressão apresentada na Seção 6.6.

A definição da expressão considera as seguintes suposições: (i) que após a computação de um cubo o espaço ocupado em memória é *tAuxSize =*$(\sum_{i=1}^{D} Ci)$*nTuplas/tBloc*, (ii) que temos alocações de espaço em memória de trabalho para representar BIDs para responder consultas com tamanho *tPontualRAM = nBlocos x 4 bytes*, (iii) que é necessário disponibilizar espaço para representação do mapeamento dos valores de atributos das dimensões envolvidas nas consultas com o bloco recuperado da memória externa, de tamanho *tFixoTuplas=*$\sum_{i=1}^{D} 4\ x\ Ci\ x\ 4$, (iv) que deve haver espaço para manter a lista de TIDS de cada valor de atributo do bloco que é utilizado nas interseções com tamanho *tVariavelAtt = (tBloc x D x 4)*, e o espaço ocupado pelos valores de medida associados com cada TID, com tamanho

tVariavelM= (tBloc x M x (8+4)), (v) que é necessário uma lista de tamanho *tPointE=nTuplas x 4 bytes*, para conter os resultados das interseções de TIDs, (vi) que é necessário um espaço de tamanho para manter cada medida calculada de tamanho *tResultado = M x 8 bytes* e (vii) que conseguimos definir o espaço que um bloco *bSize* pode ocupar, uma vez que cada tupla tem tamanho *tSize* e *tSize = 4 +(D × 4) + (M×8)*, o tamanho do bloco é *tBloco=bSize/tSize*.

Ressaltamos que tal expressão é uma simplificação conforme explicado na Seção 6.6, mas pode ser usado para definir o tamanho de bloco. Observamos que para consultas do tipo subcubo, o tamanho do bloco deve ser algum valor próximo de tBloc crítico, pois dessa forma bCubing usa o máximo de memória para armazenar as listas de BIDS e disponibiliza memória de trabalho suficiente para as computar as consultas recuperando as listas de TIDs.

As Figuras 7.1 e 7.2 apresentam o tempo de processamento e consumo de memória versus número de interseções e acesso a memória externa necessárias para responder uma consulta de subcubo com um valor de atributo muito frequente como parâmetro. É observado que blocos com poucas tuplas, gerando maior número de blocos, degradam o desempenho, já que é necessário um grande número de interseções e muitos acessos a memória externa. O consumo de memória é maior, pois é necessário manter uma lista de BIDs maior e como o valor de atributo é muito frequente também é necessário manter várias sulistas, resultado das várias interseções necessárias.

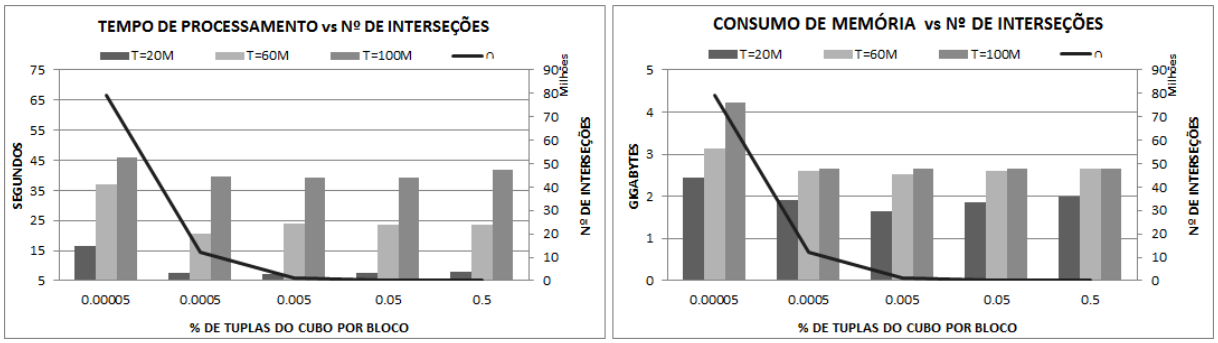

FIGURA 7.1 – Tempo de processamento e consumo de memória versus número de interseções necessárias para responder uma consulta de subcubo com um valor de atributo muito frequente como parâmetro.

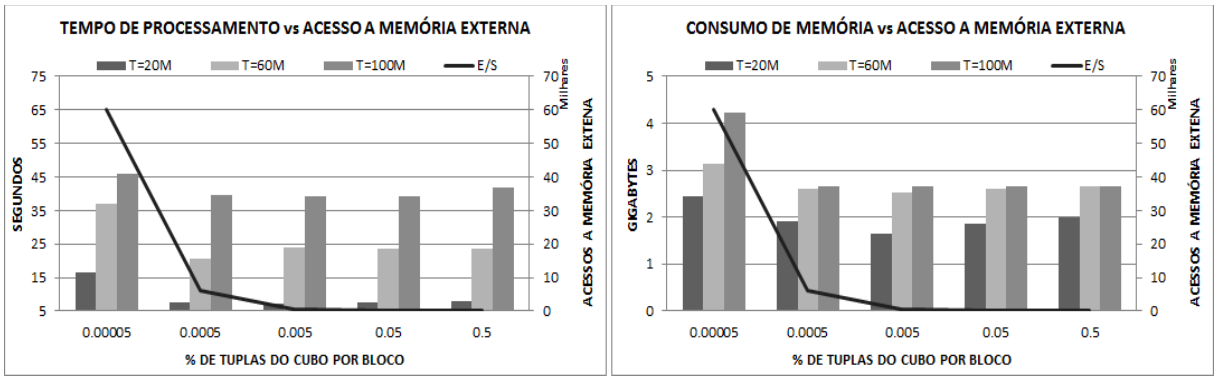

FIGURA 7.2 – Tempo de processamento e consumo de memória versus quantidade de acessos a memória necessários para responder uma consulta de subcubo com um valor de atributo muito frequente como parâmetro.

Para consultas de subcubos utilizando como parâmetro um valor de atributo pouco frequente, ao contrário do cenário anterior, mesmo havendo um maior número de acessos a memória externa e um número maior de interseções, temos melhor desempenho quando há blocos menores, gerando maior número de blocos. Esse resultado é explicado, já que interseções em listas menores são mais rápidas, portanto o baixo custo das pequenas interseções compensa o custo de acesso a memória externa. A Figuras 7.3 e 7.4 apresentam o tempo de processamento e consumo de memória versus número de interseções e acesso a memória externa necessários para responder uma consulta de subcubo com um valor de atributo pouco frequente como parâmetro.

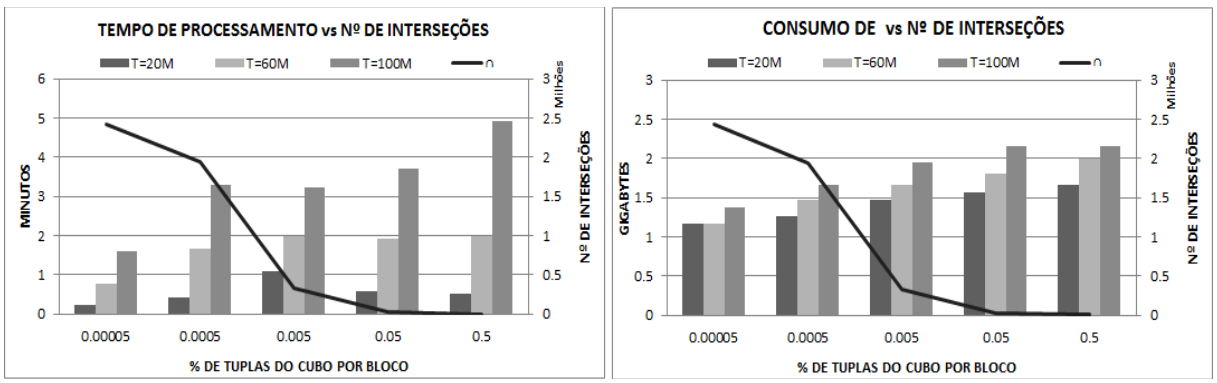

FIGURA 7.3 – Tempo de processamento e consumo de memória versus número de interseções necessárias para responder uma consulta de subcubo com um valor de atributo pouco frequente como parâmetro.

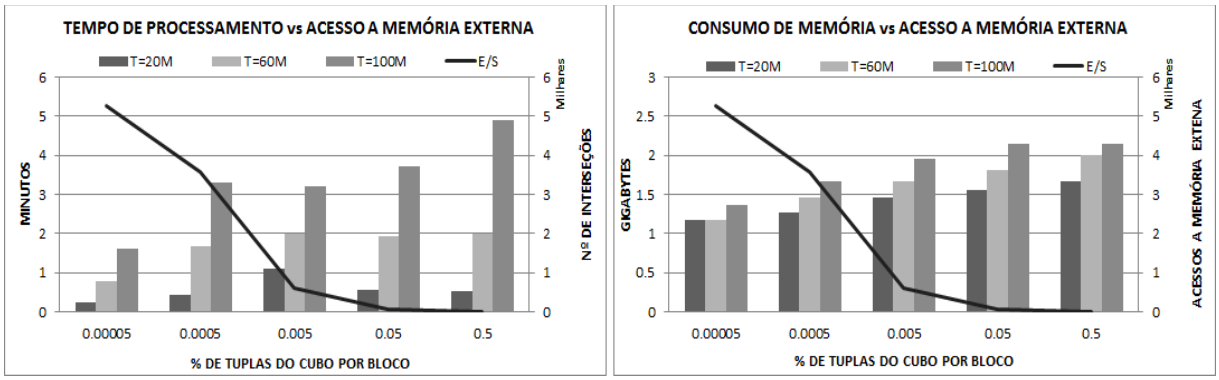

FIGURA 7.4 – Tempo de processamento e consumo de memória versus quantidade de acessos a memória necessários para responder uma consulta de subcubo com um valor de atributo pouco frequente como parâmetro.

Em domínios específicos a definição da quantidade de tuplas por bloco pode ser melhor definido, baseando-se na cardinalidade e *skew* dos valores de atributos das dimensões mais consultadas. Por exemplo, em uma relação que represente dados de uma rede social com as seguintes dimensões: *ESTADO*, *POST*, *DATA*, *PERÍODO* e *USUÁRIO*. Normalmente, há uma cardinalidade baixa e um skew muito alto nas dimensões *ESTADO* e *PERÍODO*, um skew muito baixo com cardinalidade extremamente alta na dimensão POST e um skew relativamente médio e alta cardinalidade nas dimensões *DATA* e *USUÁRIO*. Neste cenário, consultas submetidas com os atributos *ESTADO* e *PERÍODO* são beneficiadas quando maior for o bloco, pois há menos BIDs para se realizar interseções e rapidamente se recupera as listas de TIDs associadas. Contudo, se as consultas utilizarem a dimensão *POST* ou mesmo *DATA* e *USUÁRIO* é mais eficiente ter blocos menores, pois, como há baixa cardinalidade e skew baixo ou médio, as ocorrências estão associadas a poucos BIDs e estes são associados a pequenas listas de TIDs. Com isto, há menos operações de acesso à memória externa, melhorando o desempenho das interseções de TIDs.

7.1 Resumo

Neste capítulo apresentamos uma discussão sobre as abordagens Frag-Cubing, HIC e bCubing. As abordagens foram discutidas em termos do tamanho de cubos de dados, entendendo que tamanho está associado ao crescimento de tuplas e dimensões ou apenas dimensões numa relação de tuplas. Também ilustramos as características que mais impactaram nossos estudos, sejam elas consultas de diversos tipos, indexação de cubos, atualizações diversas e o consumo de memória principal. A abordagem mais promissora em cada cenário foi selecionada e as justificativas de escolha foram descritas.

Apresentamos também uma forma de definir o tamanho de bloco que deve ser usado como entrada do algoritmo de computação. A expressão é uma aproximação uma vez que é feito simplificações e não é considerado uso de memória do sistema operacional e seus sistemas de gerenciamento de memória, barramentos de memória e arquitetura de hardware por exemplo.

De uma forma geral, a abordagem bCubing é eficiente para indexar cubos massivos e é eficiente em consultar cubos de qualquer tamanho, adotando operadores subcubo. Atualizações e cálculos de medidas holísticas apontam que bCubing possui inovações promissoras.

No próximo capítulo concluímos a tese, reunindo as contribuições alcançadas e os principais e mais complexos desafios que estão pela frente.

8 Conclusão e Trabalhos Futuros

Nesta tese, apresentamos duas abordagens concebidas como estudos inicias para cubos de dados. Tais abordagens culminaram nas principais contribuições dessa tese que são outras duas abordagens para atualização e consulta eficiente de cubos parciais com dados uniformemente distribuídos ou não. Nossas abordagens são projetadas para cubos com alta dimensionalidade e elevado número de tuplas. Medidas holísticas múltiplas podem ser calculadas e mantidas eficientemente pelas abordagens implementadas.

Os estudos de desempenho apresentam um ganho no tempo de processamento e a redução do consumo de memória tanto para computar como para realizar consultas em cubos de dados massivos quando comparado com a abordagem Frag-Cubing. A abordagem bCubing cria um segundo nível de indexação chamado BID que é mantido em memória principal e que representa listas de TIDs que são mantidas em memória externa. O segundo nível de indexação possibilita que menos espaço seja necessário em memória principal para representar parcialmente um cubo de dados e viabiliza que diversos tipos de consulta possam ser respondidos eficientemente acessando da memória externa. Propomos também uma forma prática de definir o tamanho de bloco. O tamanho de bloco deve ser próximo do tBloco crítico pois garante a representação parcial do cubo e disponibilidade de memória de trabalho para execução de qualquer possível consulta de subcubos.

Testes com cubos de elevado número de tuplas, alta-dimensionalidade e alta cardinalidade inéditos na literatura comprovam a viabilidade da abordagem bCubing para calcular de forma exata medidas holísticas sem muito custo extra. Também foi realizado estudo para definição do segundo nível de indexação bCubing, ou seja, a quantidade de TIDs que deve formar um bloco representado por um BID.

Uma análise e discussões sobre as abordagens Frag-Cubing, HIC e bCubing, em termos do tamanho de cubos de dados, foi feita no Capítulo 7.

Este trabalho produziu três artigos de pesquisa completos ((SILVA; LIMA; HIRATA, 2013) (SILVA; LIMA; HIRATA, 2015a)) e (SILVA; LIMA; HIRATA, 2015b), sendo dois em veículos periódicos e um aceito para publicação em 2016. Ainda resta a publicação da abordagem bCubing, que esperamos acontecer em algum periódico estrato restrito da classificação Qualis.

A Tabela 8.1 ilustra as principais características de qCube, H-Frag, HIC e bCubing e as compara aos trabalhos correlatos. A Tabela 8.1 enfatiza a quantidade de dimensões, número de

tuplas e cardinalidade dos cubos de dados que cada abordagem computou. Também apresentamos se a abordagem permite calcular medidas holísticas, fazer atualizações e se utilizam memória externa.

TABELA 8.1 – Principais Características dos Trabalhos Correlatos

	Dimensões	Tuplas	Cardinalidade	Calcula Medidas Holísticas	Realiza Alterações	Memória Externa
bCubing	>15	10^9	10^4	Sim	Sim	Sim
HIC	>15	10^9	10^4	Não	Sim	Sim
H-Frag	>15	10^9	10^4	Não	Sim	Sim
qCube	>15	10^6	10^4	Não	Sim	Não
Frag-Cubing	>15	10^6	10^4	Não	Sim	Não
Compressed Bitmap Index Based	>15	10^5	10^2	Não	Não	Não
Word-aligned Hybrid	>15	10^6	10^3	Não	Não	Não
BitCube	>15	10^6	10^2	Não	Não	Não
S-OLAP	>15	10^5	10^2	Não	Não	Não
Shell Fragment Mini-Cube	>15	10^6	10^2	Não	Não	Não
Sampling Cube	>15	10^6	10^2	Não	Não	Não
H-Cubing	15	10^5	10^2	Não	Não	Sim
Concise Samples e Counting samples	15	10^5	10^2	Aproximada	Sim	Não
Optimization for Queries with Holistic Functions	>15	10^6	10^2	Aproximada	Não	Não
Addset Data Structure e Sliding Window	15	10^5	10^2	Aproximada	Sim	Não

Uma discussão sobre possíveis extensões das abordagens HIC e bCubing são apresentados nas próximas seções.

8.1 Cubo de dados textual

Busca de palavras-chave em um cubo de dados com dimensões textuais é um problema explorado na literatura (LIN; et al., 2008). Quando cubos de texto são construídos cada linha é associado com alguns dados de texto (por exemplo, um documento) e outras dimensões estruturais (atributos). Uma célula no cubo de texto agrega um conjunto de documentos organizados em hierarquias textuais tais como as construídas a partir de fragmentos de texto como título, parágrafo, sentença, *n*-gama de palavras e muitos outros fragmentos. Há também a possibilidade de implementação de um segundo tipo de hierarquia textual denominada

hierarquia de tópicos, onde cada tópico representa um conjunto de palavras ou itens fundamentais a diferentes contextos. Técnicas como (ZHANG; et al., 2009; ZHANG; ZHAI; HAN, 2011; OUKID; et al. 2013) buscam detectar de forma automática tópicos a partir de coleções de documentos, portanto construir a hierarquia passa a ser um desafio semântico.

A estrutura de dois níveis de índices utilizada pela abordagem bCubing poderia ser utilizada em conjunto com subcubos textuais, permitindo hierarquias textuais, assim como cálculo de medidas textuais para cubos com alta dimensionalidade. Tópicos e fragmentos de texto, sumarizados ou não, podem ser associados a BIDs e TIDs, assim como novas tabelas para novas medidas textuais podem ser incorporadas a abordagem bCubing. Neste novo cenário valores de atributos, assim como tópicos e fragmentos de texto podem ser consultados respeitando sempre suas particularidades.

8.2 Cubo fechados e cubos quocientes

Cubos quocientes foram propostos para eliminar redundâncias de células que possuem mesma medida. Suponha um cubo com apenas uma tupla t_1 composta por 100 *dimensões* $t_1=\{d_1, d_2, ..., d_{100}, m_1\}$, onde m_1 é uma medida qualquer. Independente da medida, todas as células agregadas neste cubo possuem a mesma medida m_1, ou seja, $t_2=\{*,*, ... d_{100}, m_1\}$, $t3=\{d_1,*,* ... m_1\}$, etc. compartilham m_1, portanto t_1 é suficiente para representar as demais 2^{100} outras tuplas. A solução de cubos fechados pode ser aplicada a versões bCubing que computam 2D, 3D, 4D ou mesmo 10D cuboides, pois assumimos que a representação de cubos fechados é bem compacta. Certamente haverá ganhos para responder consultas multidimensionais complexas em pouco tempo, contudo há o problema de atualização.

Cubos fechados são sensíveis a atualizações. Sua representação pode trazer desafios complexos, pois algumas células que não eram fechadas podem se tornar e outras que eram podem deixar de ser. A proposta de cubos quocientes é fazer uma compressão extrema dos dados, porém seu custo de consulta é tão elevado que não pode ser utilizado em muitas aplicações. Assim como cubos fechados, cubos quociente sofrem com atualizações recorrentes.

8.3 Paralelismo e distribuição

A abordagem bCubing pode ser paralelizada ou mesmo distribuída em máquinas de um cluster. Uma vez que tal abordagem particiona a relação verticalmente e também horizontalmente com a adoção de blocos, este particionamento facilita a distribuição. O desafio

dessa pesquisa reside no princípio da localidade dos dados, portanto faz parte de nossas pesquisas tais algoritmos responsáveis por decidir onde blocos residem no cluster. A adoção de placas gráficas é de fundamental importância. Sua arquitetura para instruções únicas e múltiplos dados (SIMD) facilita, por exemplo, a execução das inúmeras interseções em consultas complexas como as que adotam o operador subcubo. Também acreditamos que consultas a cubos com alta dimensionalidade acelerarão com novos algoritmos projetados para arquiteturas massivamente paralelas. Na literatura há estudos tanto para computação distribuída ou paralela, incluindo adoção de placas gráficas (LAUER, 2010; KACZMARSKI; RUDNY, 2011; WITTMER; LAUER; DATTA, 2010), contudo nenhum endereça soluções para bases com alta dimensionalidade e elevado número de tuplas.

8.4 Cubos espaciais e multimídia

A adoção de índice invertido facilita a indexação de tipos geométricos como linhas, pontos ou polígonos, pois estes possuem um geo-identificador único que pode ser inicialmente indexado como dado alfanumérico. As interseções e demais operações de bCubing passam a ocorrer também para a porção espacial, contudo se postergarmos tais interseções para o final da consulta conseguimos selecionar apenas os objetos espaciais alvo da consulta.

Cubos espaciais requerem mais do que integração de geo-identificadores a abordagens tradicionalmente alfanuméricas. Desta forma, bCubing precisa ser estendida para dar suporte aos diferentes tipos de hierarquias espaciais, tais como hierarquias de geo-objetos oriundos de temas geopolíticos, hierarquias de resolução, permitindo navegar em mapas temáticos com diferentes resoluções e, consequentemente, dinâmicas espaço-temporais. Há também hierarquias para espaços regulares, compostos por inúmeras células de tamanho regular e com regras de vizinhança bem estudadas. Inúmeros trabalhos foram propostos (ALZATE; MORENO; ECHEVERRI, 2012; MORENO; ARANGO; FILETO, 2009; SHEKAR; et al., 2001; BIMONTE; et al., 2011; ZAAMOUNE, et al., 2013) desde o artigo seminal de HAN, STEFANOVIC e KOPERSKI (1998) que introduziu tipos clássicos de dimensões, medidas e hierarquias espaciais.

Cubos multimídia conseguem hierarquizar dados alfanumérico estruturados e texto, assim como dado espacial, imagem, áudio e vídeo. A primeira pergunta que surge é se demandaremos tal complexidade. Redes sociais são exemplos de relações onde há espacialidade em eventos como publicar no Facebook ou mesmo postar no Twitter. Os artefatos publicados

são recorrentemente multimídia, ou seja, há texto não estruturado e há conteúdo de imagem, áudio ou vídeo. Prover hierarquizações diversas a partir de relações tão heterogêneas é o primeiro passo, contudo será que conseguiremos integrá-las? As medidas e suas particularidades também poderão operar em conjunto e de forma transparente ao usuário final? São perguntas como esta que justificam novos desafios para a área OLAP.

8.5 Redução das listas de TIDs

Explorar como as listas de TIDs podem ser reduzidas é algo ainda importante. Valores de atributos muito frequentes sequencialmente pode gerar problemas de espaço de armazenamento em memória externa quando esta for limitada a poucos gigabytes. Listas de TIDs muito grandes podem também onerar a consulta, uma vez que tais listas precisam ser carregadas para memória principal para depois realizar operações de interseção. Com isto, explorar diferentes representações para listas de TIDs é algo relevante para inúmeras abordagens baseadas em índice invertido.

Uma possível solução seria representar intervalos de TIDs para valores de atributos contínuos, ou seja, que aparecem em tuplas consecutivas na relação. Normalmente, relações com dimensões com baixa cardinalidade e skew alto podem conter valores de atributos contínuos. Dada uma dimensão com o atributo SEXO em uma relação de dados que armazena registros de um salão de beleza. Nesta relação a frequência do valor de atributo F é alta e representa grandes intervalos de tuplas, assim uma relação como a apresentada pela Tabela 8.2 pode ser reduzida conforme ilustra Tabela 8.3.

TABELA 8.2 – Relação de clientes

TID	Nome	Sexo
1	CARLA	F
2	JOANA	F
3	RUTH	F
4	NADIR	F
5	LAIS	F
6	JOÃO	M
7	DALVA	F
8	DOLORES	F

TABELA 8.3 – Relação com dimensão sexo representada por intervalo de TIDs

Dimensão	Valor de Atributo	BID	TID
SEXO	F	1	1..5
	M	2	6
	F	2	7..8

Um problema desta estratégia são as atualizações. Na alternativa de solução proposta a ideia para que o intervalo não tenha que ser reconstruído seria manter a Tabela 8.2 inalterada e marcar a tupla alterada em uma tabela auxiliar, como ilustra a Tabela 8.4, onde o valor de atributo da dimensão SEXO que foi alterado para a tupla com tid_4 é mantido. Estas e outras estratégias para redução de TIDs devem ser investigadas e incorporadas na abordagem bCubing, tornando-a cada vez mais robusta e útil para problemas OLAP reais.

TABELA 8.4 – Relação com valores de atributos alterados indexados pelo TID

Dimensão	BID	TID	Valor de Atributo
SEXO	1	4	M

Referências

AGARWAL, S.; AGRAWAL, R.; DESHPANDE, P. M.; GUPTA, A.; NAUGHTON, J. F.; RAMAKRISHNAN, R.; SARAWAGI, S. **On the Computation of Multidimensional Aggregates**. VLDB, 1996.

ALZATE, J. C.; MORENO, F. J; ECHEVERRI, J. **A spatio-temporal extension to the map cube operator**. In AIP Conference Proceedings, v. 1479, p. 2310, 2012.

AMER-YAHIA, S.; JOHNSON, T. **Optimizing Queries on Compressed Bitmaps**, Proceedings of the 26th International Conference on Very Large Data Bases, p.329-338, September 10-14, 2000.

BEYER, K.; RAMAKRISHNAN, R. **Bottom-up computation of sparse and Iceberg CUBEs**. SIGMOD, v. 28, n. 2, p. 359-370, 1999. ISSN 0163-5808.

BIMONTE, S.; TCHOUNIKINE, A.; MIQUEL, M.; PINET, F. **When spatial analysis meets olap: Multidimensional model and operators**. Exploring Advances in Interdisciplinary Data Mining and Analytics, p. 249-277; 2011.

BRAHMI, H.; HAMROUNI, T.; MESSAOUD, R. e YAHIA, S. **A new concise and exact representation of data cubes.** Advances in Knowledge Discovery and Management, Studies in Computational Intelligence (vol. 398), Springer, Berlin-Heidelberg, 2012, pp. 27–48.

BRAZ, F.; ORLANDO, S.; ORSINI, R.; RAFFAETA, A.; RONCATO, A. e SILVESTRI, C. **Approximate aggregations in trajectory data warehouses.** In Data Engineering Workshop, 2007 IEEE 23rd International Conference on, page 536-545. IEEE, 2007.

CHAN, C. Y; IOANNIDIS, Y. E. **Bitmap index design and evaluation.** In SIGMOD'98.

CHEN, Y.; DEHNE, F. K. H. A.; EAVIS, T.; RAU-CHAPLIN, A**. PnP: sequential, external memory, and parallel iceberg cube computation.** Distributed and Parallel Databases, 2008.

CHIOU, A.; SIEG, J.C. **Optimization for queries with holistic functions.** In Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on, page 327334. IEEE, 2001.

CODD, E. F. **Relational completeness of data base sublanguages**. R. Rustin (ed.), Database Systems, Prentice Hall and IBM Research Report (RJ 987), San Jose, California, 1972, 65-98.

CODD, E. F.; CODD, S. B.; SALLEY, C. T. **Providing OLAP to User-Analysts:** An IT Mandate. Technical report, E. F. Codd & Associates. 1993.

COMMONS MATH. **Commons Math: The Apache Commons Mathematics Library.** Disponível em: <http://commons.apache.org/proper/commons-math//>. Acesso em: 05 Mai. 2015.

DEHNE, F.; EAVIS, T.; RAU-CHAPLIN **Computing Partial Data Cubes for parallel Data Warehousing Application**. Proceedings of Euro PVM/MPI 01, Santorini, Greece, 2001.

DOKA, K.; TSOUMAKOS, D.; KOZIRIS, N. **Brown dwarf**: A fully-distributed, fault-tolerant data warehousing system. J. Parallel Distrib. Comput., 71:1434-1446, November 2011.

FANG, M.; SHIVAKUMAR, N., GARCIA-MOLINA, H.; MOTWANI, R. E ULLMAN, J. D. **Computing Iceberg Queries Efficiently**. VLDB, 1998.

FASTUTIL. **Fast & compact type-specific collections for Java.** Disponível em: <http://fastutil.di.unimi.it/>. Acesso em: 05 Mai. 2015.

FENG, Y.; AGRAWAL, D.; ABBADI, A. E.; METWALLY, A. **Range CUBE:** Efficient Cube Computation by Exploiting Data Correlation. In Proceedings of the ICDE Conference, page 658-670, 2004.

FERRO, A.; GIUGNO, R.; PUGLISI, P. L.; PULVIRENTI, A. **Bitcube:** A bottom-up cubing engineering. Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery, DaWaK '09. Springer-Verlag, Berlin-Heidelberg, 2009, pp. 189–203.

FRIEDMAN, E.; PAWLOWSKI, P.; CIESLEWICZ, J. **SQL/MapReduce:** A Practical Approach to Self-Describing, Polymorphic, and Parallelizable User-Defined Functions. VLDB, 2009.

GIBBONS, P. B.; MATIAS, Y. **New sampling-based summary statistics for improving approximate query answers.** In ACM SIGMOD Record, volume 27, page 331-342. ACM, 1998.

GRAY, J.; CHAUDHURI, S.; BOSWORTH, A.; LAYMAN, A.; REICHART, D.; VENKATRAO, M.; PELLOW, F.; PIRAHESH, H. **Data Cube:** A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. 1997.

HAN, J.; KAMBER, M. **Data Mining:** Concepts and Techniques. [S.l.]: Morgan Kaufmann, 2006.

HAN, J.; PEI, J.; DONG, G.; WANG, K. **Efficient Computation of Iceberg Cubes with Complex Measures.** SIGMOD, 2001.

HAN, J.; STEFANOVIC, N.; KOPERSKI, K. **Selective materialization: An efficient method for spatial data cube construction.** Research and Development in Knowledge Discovery and Data Mining, volume 1394, page 144-158. Springer Berlin Heidelberg, 1998.

HARINARAYAN, V.; RAJARAMAN, A.; ULLMAN, J. D. **Implementing Data Cubes Efficiently**. SIGMOD, 1996.

HU, K.; LING, C.; JIE, S.; QI, G., TANG, X. **Computing high dimensional MOLAP with parallel shell mini-cubes.** Proceedings of the Second international conference on Fuzzy Systems and Knowledge Discovery, August 27-29, 2005, Changsha, China.

ILLIMINE. **software and data repository from Data Mining Research Group, Data and Information Systems (DAIS) Research Laboratory, Department of Computer**

Science, University of Illinois at Urbana-Champaign. Disponível em: < http:// http://illimine.cs.uiuc.edu///>. Acesso em: 05 Mai. 2015.

INMON, W. H.; HACKATHORN, R. D. **Using the Data Warehouse, Wiley-QED** Publishing, Somerset, NJ, USA. 1994.

JANET, B.; REDDY, A. V. **Cube Index:** A Text Index Model for Retrieval and Mining, Int. Journal of Computer Applications, v.1, No. 9 (20), p. 192-330, 2010.

KACZMARSKI, K.; RUDNY, T. **Molap cube based on parallel scan algorithm**. In: SPRINGER. Advances in Databases and Information Systems. [S.l.], p. 125–138, 2011.

KIMBALL, R.; ROSS, M. **The data warehouse toolkit**: the complete guide to dimensional modeling. 2nd ed. John Wiley and Sons, 2002.

LAKSHMANAN, L., V. S.; PEI, J.; HAN, J. **Quotient cube:** How to summarize the semantics of a data cube. In Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02, page 778-789, Hong Kong, China, 2002. VLDB Endowment.

LABIO, W.; YANG, U.; CUI, Y.; GARCIA-MOLINA, H.; WIDOM, J. **Performance issues in incremental warehouse maintenance.** In Proceedings of the 26st Int'l Conference on Very Large Databases (VLDB'00), 2000.

LAUER, T.; DATTA, A.; KHADIKOV, Z.; ANSELM, C. **Exploring graphics processing units as parallel coprocessors for online aggregation.** In: ACM. Proceedings of the ACM 13th international workshop on Data warehousing and OLAP. [S.l.], p. 77–84, 2010.

LEHNER, W.; SIDLE, R.; PIRAHESH, H.; COCHRANE, R. **Maintenance of Cube Automatic Summary Tables**. In Proceedings of the ACM SIGMOD Conference, pages 512-513, 2000.

LENG, F.; BAO, Y.; YU, G.; WANG, D.; LIU, Y. **An efficient indexing technique for computing high-dimensional data cubes**. Proceedings of the International Conference on Advances in Web-Age Information Management, Berlin, Heidelberg, Germany, 2006, pp. 557–568.

LENZ, H.; THALHEIM, B. **OLAP Databases and Aggregation Functions**. Proc. 13th Int'l Conf. Scientific and Statistical Database Management, pp. 91-100, 2001.

LI, C.; CONG, G.; TUNG, A. K. e WANG, S. **Incremental maintenance of quotient cube for median.** In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, page 226235. ACM, 2004.

LI, X.; HAN, J.; GONZALEZ, H. **High-dimensional OLAP**: A minimal cubing approach. In: NASCIMENTO, M. A.; ÖZSU, M. T.; KOSSMANN, D.; MILLER, R. J.; BLAKELEY, J. A.; SCHIEFER, K. B. (Ed.). **Proceedings of the Thirtieth International Conference on Very Large Data Bases,** Toronto, Canada, August 31 - September 3 2004. [S.l.]: Morgan Kaufmann, 2004. p. 528-539. ISBN 0-12-088469-0.

LI, X.; HAN, J.; YIN, Z.; LEE, J-G.; SUN, Y. **Sampling Cube**: A Framework for Statistical OLAP over Sampling Data. Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'08), Vancouver, BC, Canada, 2008.

LIMA, J. d. C.; HIRATA, C. M. **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, p. 362-376, 2007.

LIMA, J. d. C. **Sequential and Parallel Approaches to Reduce the Data Cube Size**. 2009. 191f. Computer Science – Electronic Engineering, Aeronautical Institute of Technology, São José dos Campos.

LIMA, J. d. C.; HIRATA, C. M. **Computing data cubes using exact sub-graph matching**: The sequential mcg approach. In: SHIN, S. O. S. Y. (Ed.). ACM SAC. [S.l.]: ACM, 2009. p. 1541{1548. ISBN 978-1-60558-166-8.

LIMA, J. d. C.; HIRATA, C. M. **Multidimensional cyclic graph approach**: representing a data cube without common sub-graphs. Information Sciences 181 (13), July 2011, 2626-2655.

LIN, C. X.; DING, B., HAN, J.; ZHU, F.; ZHAO, B. **Text Cube: Computing IR Measures for Multidimensional Text Database Analysis**. In Data Mining, ICDM '08. Eighth IEEE International Conference, p.905-910, 15-19 Dec, 2008.

LIU, X.; TANG, K.; HANCOCK, J.; HAN, J.; SONG, M.; XU, R.; MANIKONDA, V.; POKORNY, B. **SocialCube:** A Text Cube Framework for Analyzing Social Media Data. In: Proceedings of ASE International Conference on Social Informatics, Washington, DC, 2012.

LO, E.; KAO, B.; HO, W-S.; LEE, S. D.; CHEUNG, D. W. **OLAP on sequence data**. Proceedings of the 2008 ACM SIGMOD international conference on Management of data, , Vancouver, Canada, June 09-12, 2008.

MOREIRA, A.A.; LIMA, J. d. C. **Full and partial data cube computation and representation over commodity PCs.** In: Information Reuse and Integration (IRI), IEEE 13th International Conference on , vol., no., pp.672-679, 8-10 Aug. 2012.

MORENO, F.; ARANGO, F.; FILETO, R. **Extending the map cube operator with multiple spatial aggregate functions and map overlay**. Proceedings of Geoinformatics 2009, Fairfax, VA, USA, August 2009.

MUMICK, I.; QUASS, D.; MUMICK, B. **Maintaince of data cubes and summary tables in a warehouse.** In Proc. Of ACM-SIGMOD Int'l Conference on Management of Data, 1997.

NANDI, A.; YU , C.; BOHANNON, P.; RAMAKRISHNAN, R. **Distributed cube materialization on holistic measures.** Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, p.183-194, April 11-16, 2011.

NG, R. T.; WAGNER, A. S. e YIN, Y. **Iceberg-cube computation with PC clusters.** Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, California, USA, May 21-24, p.25-36, 2001.

OUKID L.; ASFARI, O.; BENTAYEB, F.; BENBLIDIA, N.; BOUSSAID, O. **CXT-cube:** contextual text cube model and aggregation operator for text OLAP. Proceedings of the sixteenth international workshop on Data warehousing and OLAP, October 28-28, San Francisco, California, USA, 2013.

O'NEIL, P.; QUASS, D. **Improved query performance with variant indexes**. Proceedings of the 1997 ACM SIGMOD international conference on Management of data (SIGMOD '97), ACM, New York, NY, USA, p. 38-49. 1997.

PEREIRA, F. M. **Minerando exceções em cubos OLAP**. 2003. 163f. Cin. Ciência da Computação, Universidade Federal de Pernambuco, Recife.

POOSALA, V.; GANTI, V. **Fast approximate answers to aggregate queries on a data cube.** Proceedings of the 11th International Conference on Scientific and Statistical Database Management, p.24-33, July 28-30, 1999

ROSS, K.; SRIVASTAVA, D.; SUDARSHAN, S. **Materialized view maintenance and integrity constraint checking:trading space for time.** In Proc. Of ACM-SIGMOD Int'l Conference on Management of Data, 1996.

RUGGIERI, S.; PEDRESCHI, D.; TURINI, F. **Dcube**: discrimination discovery in databases. Proceedings of ACM SIGMOD International Conference on Management of Data, New York, NY, USA, p. 1127–1130, 2010.

SARAWAGI, S.; AGRAWAL, R.; MEGIDDO, N. **Discovery-driven exploration of OLAP data cubes.** Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology, p.168-182, March 23-27, 1998.

SILVA, R. R.; LIMA, J. d. C.; HIRATA, C. M. **qcube: efficient integration of range query operators over a high dimension data cube.** Journal of Information and Data Management 4 (3), 2013, 469–482.

SILVA, R. R.; LIMA, J. d. C.; HIRATA, C. M. **A Hybrid Memory Data Cube Approach for High Dimension Relations**. Proc. 17th International Conference on Enterprise Information Systems (ICEIS), SCITEPRESS Digital Library, Barcelona, 2015a.

SILVA, R. R.; LIMA, J. d. C.; HIRATA, C. M. **Computing BIG data cubes with hybrid memory**. Journal of Convergence Information Technology, Submitted, 2015b

SISMANIS, Y.; DELIGIANNAKIS, A.; ROUSSOPOULOS, N.; KOTIDIS, Y. **Dwarf**: shrinking the petacube. Proceedings of ACM SIGMOD International Conference on Management of Data, New York, NY, USA, p. 464–475, 2002.

SHEKAR, S., LU, C., TAN, X., CHANG, S.; VASTRAI, R. **Map Cube: A Visualization Tool for Spatial Data Warehouse**. Geographic Data Mining and Knowledge Discovery, Taylor and Francis, London, p. 74-90, 2001.

UC IRVINE. **Marchine Learning Repository.** Disponível em: <http:// www.ics.uci.edu/~mlearn >. Acesso em: 05 Mai. 2015.

VASSILIADIS, P. **Modeling Multidimensional Databases, Cubes and Cube Operations**. Proc. 10th Int'l Conf. Scientific and Statistical Database Management, pp. 53-62, 1998.

WITTMER, S.; LAUER, T.; DATTA, A. **Real-time computation of advanced rules in olap databases**. In: SPRINGER. Advances in Databases and Information Systems. [S.l.], p. 139–152, 2011.

WREMBEL, R.; KONCILIA, C. **Data Warehouses and OLAP:** Concepts, Architectures and Solutions. Edição: IRM Press. Hershey PA: Idea Group Inc., 2007.

WU, M. C.; BUCHMANN A. P. **Encoded bitmap indexing for data warehouses.** Proceedings of the Fourteenth International Conference on Data Engineering, p.220-230, February 23-27, 1998.

WU, K.; OTOO, E. J.; SHOSHANI, A. **Compressing bitmap indexes for faster search operations.** Proceedings of the Fourteenth International Conference on Scientific and Statistical Database Management, SSDBM '02. IEEE Computer Society, Washington, DC, USA, p. 99–108, 2002.

WU, K.; OTOO, E. J.; SHOSHANI, A. **A performance comparison of bitmap indexes**. Proceedings of the Tenth International Conference on Information and Knowledge Management, CIKM '01. ACM, New York, NY, USA, p. 559–561, 2001.

WU, K.; OTOO, E. J.; SHOSHANI, A. **On the performance of bitmap indices for high cardinality attributes**. Proceedings of the Thirtieth International Conference on Very Large Data Bases (vol. 30), VLDB '04. VLDB Endowment, pp. 24–35, 2004.

STOCKINGER, K.; WU, K. **Bitmap Indices for Data Warehouses**. Idea Group, Inc., Chapter VII, 179--202. LBNL-59952, 2006.

WU, K.; STOCKINGER, K.; SHOSHANI, A. **Breaking the curse of cardinality on bitmap indexes**. Proceedings of the Twentieth International Conference on Scientific and Statistical Database Management, SSDBM '08. Springer-Verlag, Berlin-Heidelberg, p. 348–365, 2008.

XIN, D.; HAN, J. e WAH, B. W. **Star-cubing**: Computing iceberg cubes by top-down and bottom-up integration. Proceedings of the 29th International Conference on Very Large Data Bases - VLDB '2003, VLDB Endowment, v.29, p. 476-487, 2003.

XIN, D.; SHAO, Z.; HAN, J.; LIU, H. **C-cubing**: Efficient computation of closed cubes by aggregation-based checking. In: LIU, L.; REUTER, A.; WHANG, K.-Y.; ZHANG, J. (Ed.). **Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA**. [S.l.]: IEEE Computer Society, p. 4-16, 2006.

ZAAMOUNE, M.; BIMONTE, S.; PINET, F.; BEAUNE, F. **A New Relational Spatial OLAP Approach for Multi-resolution and Spatio-multidimensional Analysis of Incomplete Field Data**, ICEIS 2013, p 145-152, 2013.

ZHANG, D.; ZHAI, C.; HAN, J.; SRIVASTAVA, A.; OZA, N. **Topic modeling for OLAP on multidimensional text databases: topic cube and its applications**. Statistical Analysis and Data Mining, v.2 n.5-6, p.378-395, December 2009.

ZHANG, D.; ZHAI, C.; HAN, J. **Mitexcube: Microtextcluster cube for online analysis of text cells**. Conference on Intelligent Data Understanding, p. 204-218, 2011.

ZHAO, Y.; DESHPANDE, P. M.; NAUGHTON, J. F. **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, p. 159-170, 1997.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS
TD	16 de dezembro de 2015	DCTA/ITA/TD-044/2015	144

5. TÍTULO E SUBTÍTULO:

Abordagens híbridas para cubo de dados massivos com alta dimensionalidade: HIC e bCubing

6. AUTOR(ES):

Rodrigo Rocha Silva

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):
8.
Instituto Tecnológico de Aeronáutica – ITA

9. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

1. Cubo de Dados. 2. Alta Dimensionalidade. 3. Índice Invertido.

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Celeiros de dados; Banco de dados; Memória (computadores); Recuperação da informação; Armazenamentos de dados; Complexidade computacional; Processamento em paralelo (computadores); Computação.

10. APRESENTAÇÃO: **X Nacional** **Internacional**

ITA, São José dos Campos. Curso de Doutorado. Programa de Pós-Graduação em Engenharia Eletrônica e Computação. Área de Informática. Orientador: Prof. Dr. Celso Massaki Hirata. Defesa em 27/11/2015. Publicada em 2015.

11. RESUMO:

Abordagens para computação de cubos de dados utilizando a estratégia de índices invertidos, tais como Frag-Cubing, são alternativas eficientes em relação às tradicionais abordagens para computação de cubos de dados com alta dimensionalidade, entretanto tais abordagens são limitadas pela memória principal (RAM) disponível. Neste trabalho, é apresentadado duas abordgens iniciais: qCube e H-Frag. qCube é uma extensão da abordagem Frag-Cubing que possibilita consultas de intervalo e H-Frag é uma abordagem que utiliza memória principal e memória externa a partir de definições do usuário. Com base nas abordagens iniciais, propomos duas outras que utilizam o sistema de memória composto por memória principal e memória externa, o qual chamamos de sistema híbrido de memória, para computar e manter atualizado cubos com alta dimensionalidade e elevado número de tuplas: HIC e bCubing. Em HIC, partições de cubos são armazenados em RAM e na memória externa utilizando a mesma representação de Frag-Cubing, contudo valores de atributos frequentes são armazenados em memória principal e valores de atributos pouco frequentes são armazenados em memória externa. HIC utiliza um parâmetro, chamado frequência acumulada crítica, para definir quais os valores de atributo são armazenados em memória principal ou em memória externa. bCubing particiona uma lista de identificadores de tuplas (TIDs) implementando a inversão de tuplas em dois níveis: um nível onde o identificador é o índice de bloco (BID) e o segundo nível onde o identificador é o índice da tupla (TID). As listas de TIDs dos valores de atributos são armazenadas em memória externa. As listas de BIDs são mantidas em memória principal e indexadas pelos valores de atributos. bCubing é capaz de calcular e manter atualizadas medidas holísticas de forma exata em cubos com alta dimensionalidade e elevado número de tuplas. Experimentos utilizando uma relação com 480 dimensões e 10^7 tuplas mostram que a abordagem bCubing é apenas 30% mais lenta do que Frag-Cubing para computação de cubos e aproximadamente 3 vezes mais rápida para responder consultas multidimensionais complexas a partir de tais relações. Um cubo massivo com 60 dimensões e 10^9 tuplas foi computado por bCubing usando 84 GB de RAM, enquanto o Frag-Cubing não computou tal cubo em uma máquina com 128 GB de RAM sem realizar operações de swap do sistema operacional. O impacto do cálculo de medidas holísticas em um cubo de dados com alta dimensionalidade também foi avaliado e os resultados demonstram que a abordagem bCubing gasta, em média, 10% mais tempo ao calcular medidas holísticas do que consultas com medidas COUNT. A abordagem bCubing respondeu consultas em um cubo de dados com 1.2 bilhões de tuplas em até 4 minutos, sendo uma destas consultas *Q* composta por dois operadores de subcubo e um operador EQUAL. A consulta *Q* calculou três medidas holísticas de forma exata: desvio padrão, mediana e moda.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) SECRETO**