FRAGMENTAÇÃO FÍSICA DE DADOS EM DATAWAREHOUSES BASEADA EM ÁRVORES R.

Rogea Rocha Silveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Claudio Esperança, Ph. D.

Prof. Geraldo Zimbrão da Silva, D. Sc.

Profª. Vanessa de Paula Braganholo, D. Sc.

RIO DE JANEIRO, RJ, BRASIL
SETEMBRO DE 2007

SILVEIRA, ROGEA ROCHA
Fragmentação Física de Dados em Data Warehouses Baseada Em Árvores
R. [Rio de Janeiro] 2007
IX, 71p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)
Dissertação – Universidade Federal do Rio de Janeiro, COPPE
1. Fragmentação Física
2. Índices Multidimensionais
3. Paralelismo
I. COPPE/UFRJ II. Título (série)

Aos meus pais e ao meu esposo Rodrigo, por todo apoio.

AGRADECIMENTOS

Agradeço ao meu orientador, professor Geraldo Zimbrão por toda sua dedicação, seu incentivo e apoio que foram fundamentais para a conclusão dessa dissertação. Sua participação em minha carreira acadêmica, ocorrida em vários momentos: quando foi meu professor de disciplinas na graduação, no mestrado e orientador de projeto final de curso, foi muito importante na minha formação e servirá de exemplo em toda minha carreira.

Agradeço aos professores Cláudio Esperança e Vanessa Braganholo por terem aceitado o convite de participação na banca. Com certeza todas as sugestões contribuirão muito para nosso trabalho.

Agradeço ao professor Jano de Souza, chefe da linha de Banco de Dados, pelo apoio tanto na carreira acadêmica quanto profissional e por se dedicar em manter o nível de excelência da linha de pesquisa.

Agradeço ao professor Blaschek pelo apoio durante os anos de projeto COPPETEC, onde pude aplicar meus conhecimentos acadêmicos e desenvolver meu lado profissional.

Agradeço à Patrícia Leal, secretária de linha de banco de dados, e à Solange Santos, secretária acadêmica do PESC/UFRJ por estarem sempre dispostas a ajudar.

Agradeço à minha família por todo o carinho. Aos meus pais por privilegiarem minha educação e terem investido no meu futuro. Tudo que sou hoje devo a eles.

Agradeço ao meu esposo Rodrigo pelo apoio incondicional, pelas palavras de carinho nos momentos em que eu mais precisava e pela paciência e compreensão nos momentos em que não pude estar presente. Essa conquista também é sua. Obrigada por tudo!

Agradeço aos meus amigos, Vinicios Pereira, Amanda Varella, Vinicius Vonheld e Camille Furtado pelo incentivo e apoio em todos os momentos. A amizade de vocês também foi fundamental nessa jornada.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

FRAGMENTAÇÃO FÍSICA DE DADOS EM DATA WAREHOUSES BASEADA EM ÁRVORES R.

Rogea Rocha Silveira

Setembro/2007

Orientador: Geraldo Zimbrão Da Silva

Programa: Engenharia de Sistemas e Computação

Data warehouses (DW) integram e acumulam grande volume de dados históricos provenientes das transações operacionais das organizações e de várias fontes externas para dar suporte ao processo de tomada de decisão das organizações. Consultas complexas, de alto custo e que exigem grande poder de processamento do SGBD são executadas sobre esse repositório para extração de informações estratégicas para a organização. Esta dissertação tem como objetivo melhorar o desempenho desse tipo de consulta levando-se em consideração a forma como a modelagem multidimensional organiza os dados de um DW. O esquema estrela gerado por essa modelagem é composto por tabelas dimensão e por tabelas fato e pode ser visto como um espaço multidimensional chamado de cubo de dados. Nós propomos uma abordagem que particiona esse espaço através de uma estrutura de indexação multidimensional, no nosso caso, a árvore R, utilizada para fragmentar fisicamente a tabela fato de um esquema estrela. A alocação dos fragmentos em um ambiente distribuído possibilita que as consultas sejam processadas através do paralelismo intra-consulta, que pode ajudar a reduzir consideravelmente o tempo de execução das consultas individualmente. Para validar nossa proposta, desenvolvemos um protótipo onde rodamos experimentos em um ambiente distribuído de 8 nós usando o *benchmark* TPC-H. Os resultados mostram que nossa proposta permite limitar o volume de dados a ser acessado por determinadas consultas, reduzindo desta forma seu tempo de execução.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PHYSICAL PARTITIONING OF DATA WAREHOUSES BASED IN R-TREES

Rogea Rocha Silveira

September/2007

Advisor: Geraldo Zimbrão Da Silva

Department: Engenharia de Sistemas e Computação

Data warehouses (DW) integrate and accumulate large volumes of historic data from operational transactions of organizations and from various external sources, to support the organization's decision making process. Complex queries, costly and the one's that demand high processing power from the database engine, are run over the aforementioned repository to extract reports of strategic information for the organization. The objective of this work is to improve the performance of this type of queries taking into account the way in which multidimensional modeling organizes the data of a DW. The star scheme generated by this modeling is composed of dimension and fact tables and can be seen as a multidimensional space called data cube. Our proposed approach is to partition this space through a multidimensional indexing structure, in this case the R-tree that we use to physically fragment the fact table of a star scheme. The fragment allocation in a distributed environment allows queries to be processed through intra-query parallelism, which can help reduce considerably the execution time of queries individually. To validate our approach, we implemented a prototype where we ran experiments in an 8 nodes distributed environment using the TPC-H benchmark. The results have shown that our approach allows limiting the volume of data to be accessed by certain queries, thus reducing its execution time

ÍNDICE

LISTA DE FIGURAS

1. Introdução

Um *data warehouse* é um repositório de dados que integra e acumula dados históricos gerados por transações operacionais. Ele é especialmente modelado para apoiar o processo de tomada de decisão nos negócios de uma organização (INMON, 2005). Aplicações OLAP (*On-line analytical processing*) acessam esse repositório e disponibilizam consultas onde os analistas de negócio podem extrair diversas informações estratégicas sobre o negócio da organização. Essas consultas caracterizam-se por serem complexas, de alto custo e por acessarem grandes volumes de dados. Garantir eficiência no processamento desse tipo de consulta é fundamental para que os analistas de negócio tenham acesso às informações rapidamente e assim possam manter a organização competitiva frente à sua concorrência.

Entre as estratégias utilizadas para alcançar maior desempenho nesse cenário, destacam-se a utilização de processamento distribuído, onde ganho de desempenho é alcançado através de um bom projeto de distribuição de dados que inclui a fragmentação dos dados e a alocação desses fragmentos entre os nós do ambiente distribuído. Os sistemas de computação distribuída consistem em um conjunto de elementos de processamento independentes que estão interconectados por uma rede de computadores e que cooperam entre si na execução de suas tarefas atribuídas (ÖZSU e VALDURIEZ, 1999). O processamento distribuído se baseia na idéia de que é mais eficiente dividir um problema complexo em fragmentos menores, de resolução mais simples, que podem ser processados por diferentes dispositivos computacionais.

Em sua maioria, os sistemas distribuídos representam um método mais econômico para alcançar maior poder computacional. Um exemplo que vem ganhando cada vez mais destaque são os agrupamentos de computadores pessoais. Um agrupamento de computadores pessoais é um sistema local que consiste em um conjunto de computadores independentes interconectados em uma rede de alta velocidade (STERLING, 2001). Atualmente é considerada uma plataforma bem atraente economicamente para processamento paralelo e aceleração de transações, comparado aos servidores de arquiteturas paralelas com multiprocessadores fortemente acoplados, que além do hardware especializado, exigem também software especial, o que encarece muito essa opção (CECCHET, MARGUERITE et al., 2004). Além disso, os sistemas distribuídos são mais flexíveis, pois permitem um crescimento incremental através da

inclusão de novos dispositivos computacionais caso seja necessário alcançar maior desempenho.

O projeto de distribuição de dados consiste em determinar a maneira pela qual os dados são distribuídos pelos nós de uma rede (ÖZSU e VALDURIEZ, 1999). Ele é feito em duas etapas. Primeiramente, é definido o projeto de fragmentação da base, onde as tabelas são decompostas em partes menores, chamados de fragmentos. Em seguida, o projeto de alocação é responsável por definir como esses fragmentos serão alocados entre os nós da rede.

A distribuição dos dados permite que as consultas sejam processadas paralelamente. Vários trabalhos empregaram o processamento paralelo em diferentes cenários e alcançaram bons resultados ((LIMA, 2004), (FURTADO C., 2006), (MATTOSO, ZIMBRÃO, LIMA, 2005), (MYSQL, 2006), (SWAMINATHAN, 2005), (ORACLE, 2003)). O processamento paralelo pode ser feito de duas maneiras: através do paralelismo inter-consulta ou do paralelismo intra-consulta. O paralelismo inter-consulta consiste em executar paralelamente consultas distintas nos nós da rede. Seu objetivo é aumentar a vazão do sistema. Já o paralelismo intra-consulta consiste em subdividir uma mesma consulta em sub-consultas que são executadas em paralelo em nós distintos da rede. Seu objetivo é reduzir o tempo de execução das consultas individualmente. Por isso, o paralelismo intra-consulta é mais apropriado para obter ganho de desempenho em consultas OLAP, que são de alto custo e demandam um tempo de execução considerável (AKAL, BÖHM et al.), (RÖHM, BÖHM et al., 2000),(LIMA, 2004).

O trabalho de (RÖHM, BÖHM et al., 2000) compara duas propostas de projetos de distribuição de dados do *benchmark* TPC-R (TPC, 2005c) em um agrupamento de computadores pessoais. Em um dos projetos, propõe a replicação total, onde a base de dados inteira é replicada em todos os nós. No segundo, propõe um projeto híbrido que combina fragmentação física e replicação parcial, onde tabelas maiores são fragmentadas fisicamente e as outras tabelas são totalmente replicadas entre os nós. Os experimentos executados com consultas típicas de consultas OLAP e OLTP (*On-line Transaction Processing*) mostraram que a replicação total conseguiu aceleração linear com respeito ao número de nós e aumento da vazão de consultas executadas. Entretanto, o desempenho do projeto híbrido superou a abordagem de replicação total, alcançando aceleração superlinear. A pesquisa verificou que a fragmentação física da tabela maior em fragmentos menores fez com que o SGBD gerasse planos de execução mais

eficientes sobre essas tabelas menores, pois o tamanho reduzido da tabela permite reduzir o custo da junção além de melhor utilização do *cache* (RÖHM, BÖHM et al., 2000). Logo, a estratégia de reduzir o tamanho das tabelas maiores através da fragmentação física aliado ao paralelismo intra-consulta se mostra como uma ótima alternativa para melhorar o desempenho de consultas típicas de ambiente OLAP.

Entretanto, poucos trabalhos levam em consideração a forma como os dados são organizados em *data warehouses* durante o projeto de fragmentação. Nossa proposta é aproveitar a característica da modelagem multidimensional e utilizar um índice multidimensional para fragmentar fisicamente a tabela fato de um esquema estrela com o objetivo de reduzir o tempo de processamento de consultas OLAP.

A modelagem multidimensional é a técnica mais utilizada por projetos de *data warehouses*. O esquema de dados produzido por essa técnica é conhecido como esquema estrela. O esquema estrela é formado por dois tipos de tabela: tabelas fato e dimensão. A tabela fato armazena as medidas numéricas do negócio modelado e as chaves estrangeiras das dimensões que a descrevem. A organização dos dados de um esquema estrela pode ser vista como um espaço multidimensional conhecido como "cubo de dados" (GARCIA-MOLINA, ULLMAN et al., 2001). As arestas do cubo representam as dimensões e a combinação de dimensões define um ponto do espaço multidimensional e representa uma tupla da tabela fato.

Os índices multidimensionais por sua vez dividem um espaço multidimensional em sub-regiões para tornar mais eficiente a recuperação de objetos multidimensionais. Nossa estratégia consiste em utilizar como índice multidimensional a árvore R (árvore de região) (GUTTMAN, 1984) para particionar o espaço definido por uma tabela fato e suas dimensões relacionadas. Posteriormente, a própria árvore R é utilizada para descobrir o conjunto de fragmentos que possuem tuplas relevantes para determinada consulta. Dessa forma, nossa abordagem permite reduzir o número de fragmentos a serem processados por determinada consulta além de permitir utilizar paralelismo intra-consulta para processamento de consultas e assim atingir nosso objetivo.

A avaliação da nossa proposta foi feita através da implementação de um protótipo com a base de dados e consultas do TPC-H (TPC, 2005a) em um ambiente distribuído com 8 nós cada um executando uma instância do SGBD PostgreSQL (POSTGRESQL, 2006). Os resultados mostram cenários em que obtivemos ganho tanto em ambientes monoprocessados como em ambientes paralelos.

A dissertação está organizada da seguinte forma: no capítulo 2 apresentamos a revisão da literatura com os trabalhos relacionados. No capítulo 3 apresentamos nossa proposta de fragmentação física e o capítulo 4 apresenta os resultados de desempenho obtidos com nossa técnica. Finalmente, o capítulo 5 conclui nossa dissertação e apresenta as possibilidades de trabalhos futuros.

2. Indexação Multidimensional e Fragmentação Física

O objetivo deste capítulo é apresentar os conceitos básicos que são necessários para o entendimento da solução nos quais nossa proposta é baseada. Iniciamos o capítulo falando a respeito das características da modelagem multidimensional na seção 1. Na seção 2 descrevemos os conceitos da indexação multidimensional onde abordamos em especial a árvore R. Na seção 3 apresentamos o algoritmo de clusterização que utilizamos para agrupar os objetos na árvore R. Em seguida, na seção 4 abordamos sobre sistemas de bancos de dados distribuídos e suas vantagens. Na seção 5 discorremos sobre projeto de distribuição de dados, que inclui o projeto de fragmentação (física ou virtual) e o projeto de alocação dos fragmentos em um ambiente distribuído. Nesta seção também abordamos o processamento de consultas em paralelo propiciado pelo projeto de distribuição de dados além dos projetos de distribuição específicos para *data warehouses*. Na seção 6 analisamos alguns trabalhos relacionados e finalmente na seção 7 concluímos o capítulo.

2.1. Modelagem Multidimensional

Um *data warehouse* (DW) é um repositório de dados projetado para apoiar o processo de tomada de decisão nos negócios de uma organização (INMON, 2005). Ele acumula o histórico de dados gerado pelas transações operacionais da organização. Através de consultas feitas sobre esse repositório, os analistas de negócio podem extrair diversas informações sobre a organização, como por exemplo, o perfil de seus clientes e de sua concorrência e o desempenho dos negócios da organização nos últimos meses.

Os principais requisitos de um DW é prover facilidade de acesso a informação organizacional, garantir que os dados estejam consistentes e ser flexível a mudanças através de fácil adaptação, sendo dessa forma a base para uma constante melhora do processo de tomada de decisão da organização (KIMBALL e ROSS, 2002). O modelo dimensional se encaixa nesse contexto na medida em que organiza os dados em um formato que provê facilidade de entendimento e de absorção de evoluções provenientes do banco operacional.

Atualmente, o esquema dimensional é amplamente utilizado, sendo a técnica mais viável para apresentar, armazenar e acessar dados para um DW. O objetivo da modelagem dimensional é dispor os dados de maneira simples e de fácil compreensão.

Além disso, o esquema estrela é projetado de forma a suportar consultas que contenham operações de agregação, *roll-ups* e *drill downs* utilizados pelas aplicações OLAP.

O esquema de dados utilizado para dar suporte às transações operacionais é geralmente modelado através do modelo entidade-relacionamento (ER), composto por diagramas que representam o relacionamento entre as tabelas. A modelagem dimensional também pode ser representada através do modelo ER, pois consiste de tabelas relacionais interligadas. A diferença chave entre a modelagem utilizada no ambiente operacional e a modelagem dimensional é o nível de normalização (KIMBALL e ROSS, 2002). Na modelagem apropriada ao ambiente operacional, os dados são altamente normalizados, ou seja, eles são divididos em várias entidades com o objetivo de eliminar redundância. Essa modelagem é muito boa para o processamento operacional, pois as operações de atualizações e inserções só precisam ser feitas em um local do esquema de dados. Entretanto, ela não se encaixa nos requisitos do DW, pois o modelo altamente normalizado geralmente não é intuitivo para o usuário e não permite processar eficientemente as consultas típicas de DW.

O modelo dimensional contém a mesma informação que o modelo normalizado, mas os dados são organizados de forma que garanta desempenho de consultas, flexibilidade para evoluções e facilidade de entendimento para o usuário (KIMBALL e ROSS, 2002).

Quando o modelo dimensional é baseado em um banco de dados relacional, seu conjunto de tabelas é referido como esquema estrela. O esquema estrela é composto por dois tipos de tabelas: a tabela fato e a tabela dimensão. A tabela fato é a tabela central do modelo dimensional. Ela armazena as medidas numéricas do negócio. Cada linha da tabela fato representa uma medida numérica do negócio modelado. Cada medida numérica é descrita por um conjunto de dimensões que determina o nível de granularidade da tabela fato e contextualiza a métrica em questão. A figura 1 ilustra um exemplo de esquema estrela, que descreve as vendas diárias dos produtos de uma rede de lojas. A tabela central é a tabela fato descrita pelas dimensões produto, loja e tempo.

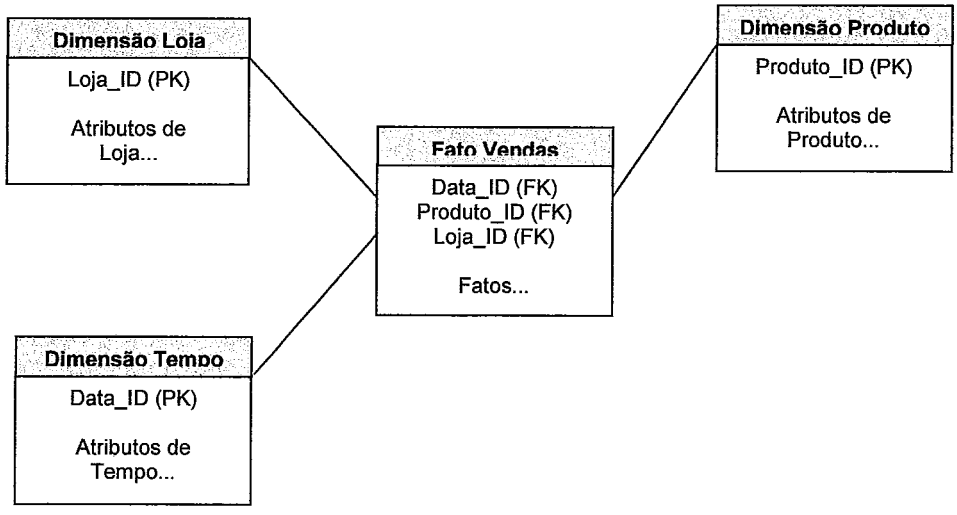


Figura 1 – Modelo dimensional (KIMBALL e ROSS, 2002)

As tabelas fato possuem uma ou mais chaves estrangeiras conectadas às chaves primárias das dimensões as quais se refere. A chave primária da tabela fato é geralmente composta pelo conjunto dessas chaves estrangeiras. As tabelas fatos expressam relacionamentos muitos para muitos.

As tabelas dimensão contêm as descrições textuais no negócio. Seus atributos são utilizados em restrições de consultas, agrupamentos e descritores de campos de relatórios. Geralmente possuem muitos atributos, mas comparadas com a tabela fato não são muito populosas. Cada dimensão possui uma chave primária que é referenciada por uma chave estrangeira de uma tabela fato.

As medidas numéricas, também conhecidas por fatos, representam os objetos de análise. Cada medida numérica depende de um conjunto de dimensões, responsáveis em dar contexto à medida. Um exemplo de medida numérica seria a quantidade vendida e suas dimensões associadas poderiam incluir o produto vendido, a loja e a data em que a venda foi feita. As tuplas da tabela fato representam a interação entre as tuplas do conjunto de tabelas dimensão, ou seja, as tuplas da tabela fato refletem os dados da transação que envolve todas as dimensões participantes.

As tabelas dimensões freqüentemente representam relacionamentos hierárquicos do negócio. Por exemplo, na tabela dimensão produto, os produtos são agrupados por marca e categoria. Cada linha armazena a marca e a categoria referente ao produto em questão. Logo, a informação hierarquizada é armazenada de forma redundante para garantir bom desempenho das consultas. Por isso as tabelas dimensões são altamente desnormalizadas.

Os atributos das tabelas dimensão podem ser vistos como dimensões de um espaço multidimensional, conhecido como "cubo de dados", com as arestas representando as dimensões produto, loja e tempo como ilustrado na figura 2. Um ponto dentro do cubo representa a medida numérica referente à combinação dos valores dessas dimensões. Logo, cada tupla da tabela fato corresponde a um ponto no espaço multidimensional (GARCIA-MOLINA, ULLMAN et al., 2001). As consultas típicas de DW agrupam os dados ao longo das dimensões desejadas através de operações de agregação. Um exemplo desse tipo de consulta é "forneça o valor total das vendas de sapato preto para cada loja e cada semestre dos anos de 2005 e 2006. No cubo, as consultas são como operações que cortam em cubos (*slice and dice)* os dados ao longo das dimensões do cubo.

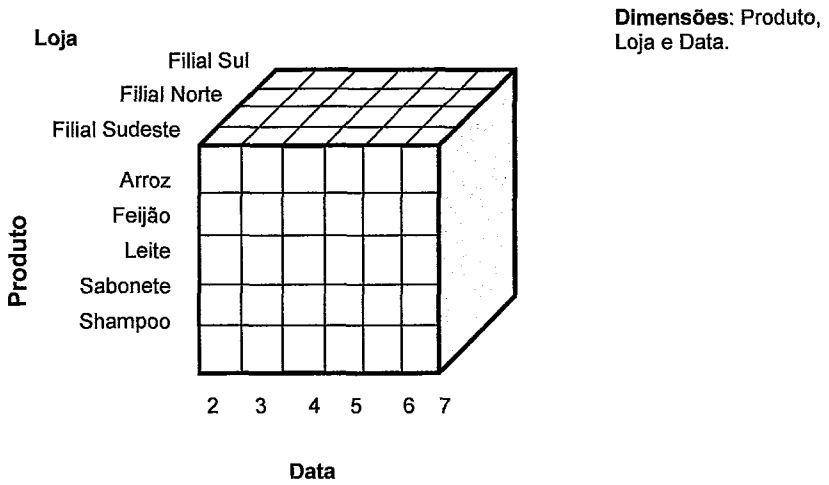


Figura 2 – Cubo de Dados

A simplicidade do modelo dimensional permite que otimizadores de banco de dados processem a consulta mais eficientemente, pois a desnormalização dos dados diminui a quantidade de junções necessárias. Além disso, sua simplicidade também permite que os usuários entendam e naveguem pelo modelo mais facilmente. Freqüentemente os analistas de negócio reconhecem seu negócio no modelo multidimensional final. Por essas razões, a modelagem dimensional é frequentemente utilizada em projetos de DW.

Na próxima seção apresentamos os conceitos básicos da indexação multidimensional e apresentamos em particular a árvore R, uma estrutura de dados muito utilizada para indexação multidimensional.

2.2. Indexação Multidimensional e a Árvore R

A principal idéia da indexação multidimensional é melhorar o acesso para recuperação de dados multidimensionais, evitando consultar todos os elementos de um espaço multidimensional para determinar aqueles que devem ser realmente recuperados. Para isso, as estruturas de indexação multidimensional organizam o espaço em sub-regiões baseando-se principalmente em duas idéias de indexação espacial: o particionamento do espaço baseado em regiões disjuntas ou aglomeração baseada em proximidade espacial, ou seja, em regiões não necessariamente disjuntas.

Geralmente o particionamento do espaço é feito de modo que as sub-regiões não possuam mais que um determinado número de pontos. Esse número máximo de pontos é geralmente a capacidade da página de disco, ou seja, o número de registros de dados que uma página é capaz de armazenar. Logo, inserções de novos pontos podem resultar em divisão de uma região em novas regiões (disjuntas ou não). Essa divisão é feita criando um ou mais hiperplanos que subdividem a região

A árvore R (árvore de retângulos) (GUTTMAN, 1984) é uma estrutura de dados inspirada na árvore B^+ para pontos ou regiões multidimensionais. Assim como a árvore B^+, ela é uma estrutura balanceada, ou seja, todos os nós folhas estão presentes no mesmo nível. Além disso, garantem utilização de pelo menos uma fração fixa do espaço, normalmente menos que 50%. Ela foi projetada pra representar pontos agrupados em regiões de duas ou mais dimensões, chamadas de hiper-retângulos. Logo, a árvore R organiza um espaço multidimensional em hiper-retângulos que agrupam objetos espacialmente próximos.

A estrutura de dados consiste de nós intermediários e nós folhas. Cada nó da árvore R possui uma quantidade variável de entradas, com tamanho máximo e mínimo pré-definidos, com exceção da raiz. Os nós intermediários são responsáveis pela indexação e cada entrada armazena um ponteiro para o nó filho e o hiper-retângulo mínimo que engloba totalmente todas as entradas do nó filho correspondente. Os nós folhas armazenam os hiper-retângulos mínimos que representam os objetos multidimensionais (GUTTMAN, 1984). A Figura 4 mostra a estrutura de uma árvore R, onde os retângulos pontilhados representam as sub-regiões. As sub-regiões não cobrem todo o espaço, o que implica que as regiões de dados que residem nas regiões maiores estão inteiramente contidas dentro de uma das regiões menores. A figura ilustra também

a sobreposição que pode ocorrer entre as regiões, embora o ideal seja minimizar as sobreposições.

O algoritmo de busca proposto por (GUTTMAN, 1984) decompõe o espaço de busca em sub-regiões e desce na árvore eliminando as regiões irrelevantes do espaço indexado até os nós folhas, onde encontra os objetos desejados. O algoritmo recebe como entrada um hiper-retângulo e o nó onde a busca deve ser feita. O objetivo é encontrar todos os registros cujo hiper-retângulo possui interseção com o hiper-retângulo de busca. A busca inicia-se pela raiz, onde cada entrada é examinada para determinar aquelas que possuem interseção com o hiper-retângulo de busca. Para todas as entradas que possuem interseção, o algoritmo de busca é aplicado novamente nos seus nós filhos e assim sucessivamente. Caso o nó examinado seja um nó folha, o algoritmo retorna a entrada como um dos resultados.

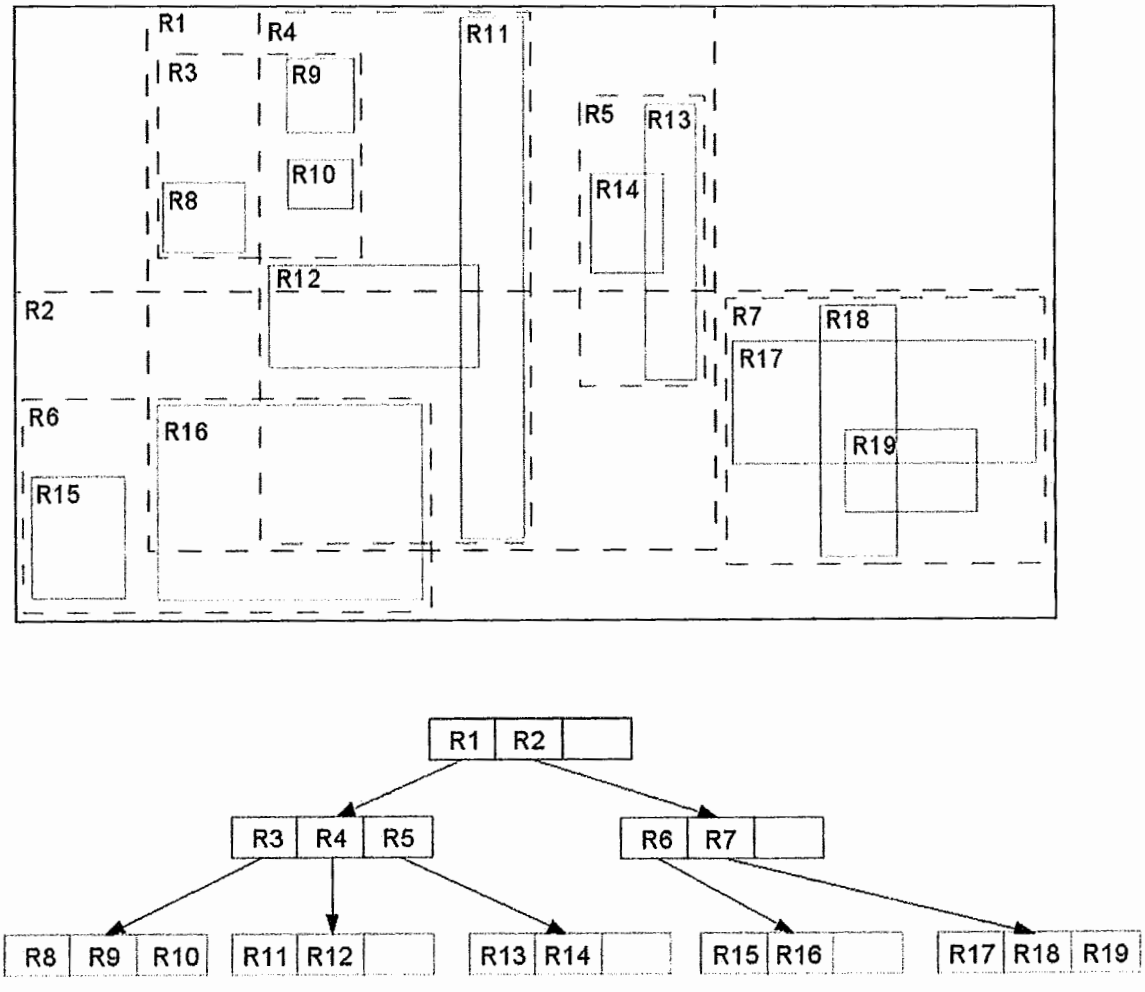


Figura 3 – Árvore R de duas dimensões.

Os algoritmos de inserção e remoção de uma entrada da árvore R utilizam os hiper-retângulos para assegurar que elementos "próximos" serão alocados no mesmo nó folha. Na inserção, as novas entradas são armazenadas nas folhas. Para escolher em que nó folha a nova entrada será armazenada, o algoritmo escolhe sempre um nó intermediário cujo hiper-retângulo necessita aumentar menos sua área para englobar a nova entrada. Esse processo é feito até que se alcance um nó folha. Se o número de entradas no nó folha ultrapassar o limite máximo de entradas definido (*overflow*), o nó sofre divisão, que consiste em dividir o nó com M + 1 entradas em dois nós, onde M é o número máximo de entradas do nó, de forma que diminua a chance de que os dois nós precisem ser examinados em buscas futuras. Para isso, o ideal é minimizar a área total dos hiper-retângulos resultantes da divisão. A divisão de um nó é propagada para os nós acima na árvore, podendo causar outras divisões até o nó raiz.

A remoção de uma entrada da árvore R ocorre através da busca do nó folha que contém a entrada para então removê-la do nó. Todo o caminho da raiz até o nó folha que sofreu a remoção deve ser atualizado. Essa atualização pode fazer com que um dos nós sofra *underflow*, ou seja, seu número total de entradas ser menor que o limite mínimo de entradas definido para a árvore. Se o *underflow* ocorrer na raiz, seu filho torna-se a nova raiz. Senão, esse nó é removido da árvore e suas entradas são reinseridas na árvore. Outra solução é redistribuir as entradas do nó entre os irmãos, de modo análogo ao que é feito na árvore B.

Em caso de estouro do número de entradas de um nó, sua divisão pode ser feita utilizando-se diferentes heurísticas para otimização do espaço. A árvore R busca minimizar a área dos nós filhos resultantes da divisão. Uma estrutura variante da árvore R é a árvore R*. Sua principal diferença é a heurística utilizada para inserção de uma nova entrada. Além de minimizar a área dos hiper-retângulos dos nós, ela também busca reduzir a margem e a sobreposição dos retângulos. (BECKMANN, KRIEGEL et al., 1990).

Nós utilizamos outra variante da árvore R proposta por (BRAKATSOULAS, PFOSER D. et al., 2002), que utiliza o algoritmo de clusterização de dados chamado k-médias para fazer a divisão de um nó que sofreu overflow. Na próxima seção falamos a respeito desse método de clusterização e suas vantagens

2.3.k-Médias

Os algoritmos de clusterização de dados são abordagens utilizadas em operações de particionamento de dados (HUANG, 1998). Os métodos de clusterização permitem particionar um conjunto de objetos em agrupamentos de forma que objetos do mesmo grupo possuam mais similaridade que objetos de outros grupos segundo algum critério pré-definido. A maior parte dos métodos de agrupamento é baseada em medidas de semelhanças ou medidas de diferença entre objetos.

Um exemplo de método para agrupar objetos é o exaustivo, que enumera todas as possíveis partições, mas sua complexidade é exponencial. Uma proposta mais viável são os métodos heurísticos, entre eles destacamos o *k-Means* (também chamado de k-Médias)

O k-Médias é um famoso algoritmo de clusterização apresentado por (MACQUEEN, 1967) muito utilizado em mineração de dados. Seu principal objetivo é classificar informações de acordo com os próprios dados, baseada em análise e comparações entre valores numéricos dos dados. Dado um conjunto de objetos numéricos e um valor inteiro k, o algoritmo busca particionar o conjunto de objetos em k grupos.

Para gerar os agrupamentos, o algoritmo calcula o centro de cada grupo, chamado de centróide e compara cada objeto com o centróide através de uma medida de distância ou similaridade, como por exemplo, a distância euclidiana ou distância de Manhattan. Os centróides são calculados pela média aritmética dos valores de cada atributo de cada objeto pertencente ao seu grupo.

O algoritmo é iniciado escolhendo-se k valores distintos para os centróides. A escolha pode ser feita aleatoriamente. Depois, todos os objetos do espaço a serem agrupados devem ser associados ao centróide mais próximo, formando agrupamentos iniciais. Para isso é preciso calcular a distância de cada ponto a todos os centróides. O próximo passo consiste em refinar o valor dos centróides, recalculando os centróides de cada um dos grupos e novamente associar os pontos aos novos centróides mais próximos. Esse último passo deve ser executado até que não haja mais alterações dos grupos, ou seja, se nenhum objeto for incorporado a um novo agrupamento diferente daquele ao qual pertencia antes. O algoritmo sempre converge a uma solução ótima local, entretanto não é garantido que alcance o ótimo global. A qualidade do resultado final do algoritmo depende dos valores escolhidos como centróides iniciais.

Uma das principais vantagens do algoritmo k-Médias é a capacidade de processar eficientemente grandes conjuntos de dados.

Nessa dissertação, utilizamos uma variante da árvore R que utiliza o k-médias para fazer a divisão do nó em caso de *overflow*. Essa variante foi proposta em (BRAKATSOULAS, PFOSER D. et al., 2002), onde eles propõe a substituição das heurísticas tradicionais utilizadas pelos algoritmos de divisão do nó pelo algoritmo de clusterização. Os resultados mostraram um aumento geral no desempenho da árvore R, com destaque para o tempo de inserção.

2.4. Sistemas de Banco de Dados Distribuído

Sistemas de computação distribuída consistem em um conjunto de elementos de processamento independentes que estão interconectados por uma rede de computadores e que cooperam entre si na execução de suas tarefas atribuídas (ÖZSU e VALDURIEZ, 1999). Os elementos de processamento são dispositivos de computação capazes de executar um programa e o conjunto desses elementos pode ser heterogêneo. O processamento distribuído surgiu da necessidade em resolver problemas grandes e complexos de maneira mais eficiente. É uma variante da estratégia "dividir para conquistar".

A idéia do processamento distribuído é dividir um problema complexo em problemas menores, mais simples de serem resolvidos. Esses fragmentos do problema complexo podem ser processados por diferentes elementos de processamento que formam um sistema onde colaboram de forma eficaz para a execução de uma tarefa comum. O fato de o problema ser dividido em partes e seu processamento ser feito por elementos distintos possibilita que ele seja resolvido paralelamente entre os componentes do sistema distribuído, o que permite um ganho maior de desempenho.

O processamento distribuído possui várias vantagens em relação ao processamento seqüencial. Os sistemas distribuídos são um método mais econômico para alcançar maior poder computacional, pois é menos custoso interconectar vários processadores do que adquirir um supercomputador, com arquitetura paralela e processadores fortemente acoplados, caros tanto em relação ao hardware quanto aos softwares específicos. Além disso, sistemas distribuídos são mais flexíveis do que máquinas isoladas, pois permitem um crescimento incremental através da inclusão de novos dispositivos computacionais para maiores desempenhos.

Os bancos de dados distribuídos são uma coleção de vários bancos de dados logicamente inter-relacionados, conectados por uma rede de computadores. Por sua vez, um sistema de banco de dados distribuído é o sistema que gerencia o banco de dados distribuído e mantém a localização dos dados transparente para as aplicações e seus usuários. Nesses sistemas, a distribuição física dos dados significa que a comunicação entre eles é feita através de uma rede, o único recurso compartilhado, e não através de memória compartilhada por exemplo (ÖZSU e VALDURIEZ, 1999).

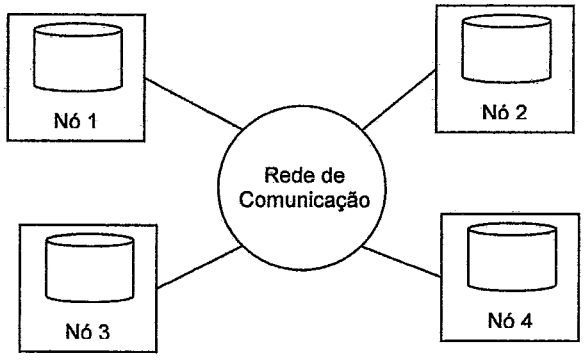


Figura 4 - Ambiente Distribuído

Um exemplo de sistema distribuído que vem ganhando destaque é o agrupamento de computadores pessoais. É considerada atualmente uma plataforma bem atraente para processamento paralelo e aceleração de transações. Eles representam uma alternativa economicamente mais interessante comparado aos servidores com arquiteturas paralelas com multiprocessadores fortemente acoplados, muito utilizados pelas aplicações que necessitam de alto poder computacional (CECCHET, MARGUERITE et al., 2004).

Um agrupamento de computadores pessoais (*cluster*) é um sistema local que consiste em um conjunto de computadores independentes interconectados em uma rede de alta velocidade (STERLING, 2001). Cada computador independente constitui um nó do agrupamento e todos os nós do agrupamento devem ser interligados por uma rede dedicada, ou seja, independente da rede que conecta o agrupamento ao mundo externo. São considerados sistemas locais, pois todos os seus componentes encontram-se fisicamente no mesmo ambiente e são gerenciados como se fosse um único sistema de computador.

Estendendo o conceito de agrupamento de computadores pessoais, o agrupamento de banco de dados consiste em uma camada intermediária de software que

controla um agrupamento de computadores onde cada nó executa um Sistema Gerenciador de Banco de Dados (SGBD) seqüencial, ou seja, não possuem nenhuma funcionalidade especial para processamento paralelo (AKAL, BÖHM et al., 2002). A camada intermediária é responsável por distribuir os dados entre os nós e gerenciar o paralelismo entre as transações.

Nas próximas seções fazemos uma análise no projeto de distribuição de dados, onde é definido como os dados devem ser distribuídos entre os nós de uma rede. Nesse contexto, falaremos também das técnicas de fragmentação e alocação de dados.

2.5. Projeto de Distribuição de Dados

O projeto de distribuição de dados consiste em determinar a distribuição de dados pelos nós de um ambiente distribuído (ÖZSU e VALDURIEZ, 1999). O processo de distribuição pode ser dividido em duas etapas: fragmentação e alocação dos dados. Na primeira etapa as tabelas são decompostas em partes menores, chamados de fragmentos. Logo, é nessa etapa que é decidida qual função de fragmentação será utilizada para dividir os dados. Em seguida, os fragmentos são alocados entre os nós da rede.

O projeto de distribuição deve ter como objetivo reduzir processamento de dados irrelevantes para a aplicação e diminuir a transferência de dados ente os nós de um sistema distribuído (NAVATHE e KARLAPALEM, 1995). Dessa forma, entre as vantagens que podem ser alcançadas temos melhoria de desempenho, autonomia, disponibilidade e confiabilidade para sistemas de bancos de dados distribuídos (ÖZSU e VALDURIEZ, 1999).

A maneira como os dados são distribuídos influencia fortemente o tipo de paralelismo a ser utilizado para o processamento de consultas, que pode ser inter-consulta, ou seja, várias consultas são executadas simultaneamente em diferentes nós da rede ou intra-consulta, quando uma única consulta é processada paralelamente por vários nós da rede (ÖZSU e VALDURIEZ, 1999).

Na próxima seção falamos a respeito da primeira etapa do projeto de distribuição de dados: a fragmentação. Descrevemos resumidamente os objetivos de diferentes técnicas de fragmentação.

2.5.1.1. Projeto de Fragmentação

A fragmentação de dados consiste em dividir uma relação em relações menores disjuntas chamadas de fragmentos. O objetivo da fragmentação é aumentar o desempenho de uma transação, na medida em que restringe o acesso apenas aos dados interessantes àquela transação. Além disso, a fragmentação de uma relação permite que uma consulta seja executada paralelamente, dividindo-a em sub-consultas que irão operar sobre os fragmentos (ÖZSU e VALDURIEZ, 1999).

Existem basicamente três tipos de fragmentação: horizontal, vertical e híbrida. Para fragmentar uma relação, são utilizados os operadores de seleção ou projeção da álgebra relacional, dependendo do tipo de fragmentação. Na fragmentação horizontal, os fragmentos são formados a partir de operações de seleção sobre a relação original. Os fragmentos possuem o mesmo esquema que a relação original. Já na fragmentação vertical, são aplicados operações de projeção sobre a relação original. Nesse caso, os esquemas dos fragmentos são diferentes do esquema da relação original. O ideal é que a fragmentação vertical seja projetada de tal forma a incluir a chave primária nos fragmentos para facilitar a reconstrução da relação original. Finalmente, a fragmentação híbrida é uma combinação da fragmentação horizontal e vertical, ou seja, os fragmentos são formados aplicando-se operações de seleções seguidas de projeções ou operações de projeções seguidas de seleções.

A fragmentação deve ser feita de tal forma que não mude a semântica do banco de dados (ÖZSU e VALDURIEZ, 1999). Para isso, três regras importantes devem ser consideradas:

1. Completude: garante que a fragmentação seja feita sem perda de dados. Uma tupla que se encontra na relação original deve ser encontrada em um ou mais de seus fragmentos.

2. Reconstrução: deve ser possível reconstruir a relação original a partir de seus fragmentos, ou seja, deve existir um operador relacional que, aplicado sobre os fragmentos, gera a relação original. Esse operador é diferente para cada tipo de fragmentação.

3. Disjunção: garante que os fragmentos horizontais são disjuntos, ou seja, se uma tupla se encontra em um dos fragmentos, ela não se encontra em nenhum outro fragmento. Na fragmentação vertical, a disjunção só se aplica aos atributos que não

fazem parte da chave primária da relação original, já que a chave se repete em todos os fragmentos.

As metodologias de fragmentação em geral se baseiam na análise do esquema de banco de dados a ser fragmentado e no conjunto de consultas típicas mais executadas sobre a base de dados. Os projetistas levam em consideração o tamanho das tabelas do esquema e a freqüência de vezes em que são acessadas além das características das consultas, os campos por elas acessados e as restrições mais utilizadas por exemplo. Todos esses quesitos são analisados para que o projeto de fragmentação seja eficiente dentro do contexto no qual se insere.

Depois do projeto de fragmentação definido, a próxima etapa consiste definir como os dados serão alocados nos nós. Na próxima seção descrevemos sucintamente os aspectos do projeto de alocação.

2.5.1.2. Projeto de Alocação

Com o projeto de fragmentação definido, a próxima etapa é definir a alocação dos fragmentos na rede de nós. Os fragmentos podem ser replicados ou pode ser alocada uma única cópia de cada fragmento na rede. A replicação dos fragmentos aumenta a disponibilidade, confiabilidade e flexibilidade do sistema. Entretanto essa opção torna mais custosa manter as réplicas consistentes quando há consultas de atualização. A replicação pode ser total ou parcial. Na replicação total, todos os fragmentos são replicados em todos os nós da rede. Dessa forma, cada nó possui uma cópia inteira a base de dados. Essa opção oferece mais disponibilidade, mas exige muito espaço em disco disponível em todos os nós para armazenar toda a base de dados. Para aplicações OLAP isso se torna mais crítico, já que as bases são consideravelmente mais volumosas.

Na replicação parcial, somente alguns fragmentos são replicados na rede. Nesse caso podemos ajustar a disponibilidade do sistema a partir do número de cópias dos fragmentos que serão replicadas. Assim como a base de dados não-replicada, a replicação parcial representa uma solução quando há restrição de espaço físico em disco.

Como já mencionado, uma das grandes vantagens da fragmentação é permitir o processamento de consultas em paralelo. Existem basicamente duas maneiras de executar o paralelismo de consultas: através do paralelismo inter-consulta e do

paralelismo intra-consulta (ÖZSU e VALDURIEZ, 1999), que serão abordados na próxima seção.

2.5.1.3. Processamento Paralelo de Consultas

O processamento paralelo é feito através de computadores com vários processadores para execução de aplicativos utilizando os processadores de modo cooperativo com o objetivo de melhorar o desempenho. Inicialmente foram utilizados para atender a necessidade de alto poder de processamento da computação científica, mas atualmente é cada vez mais utilizada para o processamento de dados (ÖZSU e VALDURIEZ, 1999).

Como citado na seção anterior, a fragmentação física permite o processamento de consultas em paralelo, que pode ser feito de duas maneiras: através do paralelismo inter-consulta e do paralelismo intra-consulta (ÖZSU e VALDURIEZ, 1999). O paralelismo inter-consulta consiste em executar paralelamente várias consultas independentes geradas por diferentes transações concorrentes independentes umas das outras. O agrupamento de computadores pessoais permite que essas consultas concorrentes sejam executadas simultaneamente, cada uma em um nó distinto da rede. Essa técnica aumenta a vazão do sistema, ou seja, há um acréscimo no número de transações executadas em certo intervalo de tempo, mas não altera o tempo de execução de nenhuma consulta em especial. Já no paralelismo intra-consulta, o objetivo é diminuir o tempo de execução de cada consulta. Para isso, essa técnica subdivide a consulta em questão em sub-consultas independentes. Cada sub-consulta é executada em paralelo em um nó distinto da rede. O término da execução da consulta será dado no momento do término de todas suas sub-consultas. O resultado da consulta original será uma composição dos resultados das sub-consultas referentes. O paralelismo intra-consulta é ideal para aceleração de execução de consultas complexas ou que acessem grandes volumes de dados.

O tipo de paralelismo a ser utilizado no processamento de consultas vai depender do projeto de banco de dados distribuído escolhido. A fragmentação física dos dados e sua alocação em diferentes nós da rede exige a utilização do paralelismo intra-consulta, já que uma consulta que precise acessar dados em mais de um fragmento da rede obrigatoriamente terá que ser subdividida em sub-consultas que irão acessar paralelamente cada fragmento relevante para a consulta. Por sua vez, a replicação total dos dados entre os nós da rede favorece a utilização do paralelismo inter-consulta, pois

como os dados estão replicados entre os nós, e cada nó armazena a base inteira, cada um deles processa uma consulta diferente.

O processamento de consultas em paralelo ainda oferece a possibilidade de melhorar o desempenho de execução de consultas através do balanceamento de carga. Essa técnica permite distribuir dinamicamente entre os processadores de uma rede o processamento de sub-consultas para evitar que a rede fique com nós sobrecarregados enquanto outros nós estão livres.

O objetivo do processamento paralelo de consultas é aumentar a capacidade de execução de consultas através do aumento de número de consultas executadas por unidade de tempo. O aumento da vazão de um sistema de banco de dados pode ser obtido tanto pela execução de consultas concorrentes em paralelo, quanto pela diminuição do tempo de resposta de uma consulta individualmente, ao ser executada em paralelo em vários processadores.

2.5.1.4. Projeto de Distribuição de Dados para Data warehouses

O projeto de fragmentação física de um banco de dados baseia-se na análise do esquema de banco de dados e nas características do conjunto de consultas típicas que são freqüentemente executadas. Seguindo o escopo da nossa dissertação, nesta seção vamos analisar as características particulares da modelagem de dados utilizada em projetos de *data warehouse* bem como suas consultas típicas.

Como já vimos na seção 2.1, a modelagem multidimensional é a mais utilizada em projetos de *data warehouse*. O conjunto de tabelas gerado por essa modelagem é referido como esquema estrela, composto por dois tipos de tabelas: tabela fato e tabela dimensão. A tabela fato é a tabela central do modelo dimensional, responsável por armazenar as medidas numéricas do negócio. As medidas numéricas são descritas por um conjunto de dimensões que determina o nível de granularidade da tabela fato e contextualiza a métrica em questão.

As consultas tipicamente executadas em ambientes de DW se caracterizam por acessarem pelo menos uma tabela fato e por serem consultas complexas, de alto custo, que acessam grande volume de dados e por isso demandam tempo de execução considerável. Dadas essas características, o objetivo do projeto de distribuição é

melhorar o desempenho das consultas individualmente. Logo, o paralelismo intra-consulta é o mais adequado para esses tipos de consultas.

A fragmentação física da base de dados possibilita diminuir o volume de dados a ser acessado por determinada consulta, pois, dependendo da consulta e do projeto de fragmentação implantado, é possível restringir o acesso da consulta a um conjunto limitado de fragmentos. No esquema estrela das aplicações OLAP, o volume de dados se concentra nas tabelas fato em detrimento das tabelas dimensão que costumam armazenar um volume menor de dados. Geralmente as tabelas dimensão ocupam menos de 10% do total de armazenamento da base (KIMBALL e ROSS, 2002). Isso permite optar pela fragmentação física somente das tabelas maiores (tabelas fato). As tabelas dimensão podem ser replicadas em todos os nós do ambiente distribuído.

O trabalho de (RÖHM, BÖHM et al., 2000) faz uma comparação entre dois projetos de distribuição distintos para agrupamento de banco de dados: um com replicação total e o outro sendo um projeto híbrido que combina replicação parcial e fragmentação física. No projeto híbrido, as relações maiores são fragmentadas entre os *n* nós do agrupamento enquanto que as outras são replicadas nos nós. Consequentemente, as consultas são processadas com paralelismo intra-consulta. Na replicação total todas as tabelas são replicadas em todos os nós do agrupamento e o paralelismo inter-consulta prevalece. Os experimentos foram executados com consultas típicas de consultas OLAP e OLTP do *benchmark* TPC-R (TPC, 2005b) em um agrupamento de banco de dados com até 6 nós executando o SGBD Oracle versão 8.0.4.

Os resultados dos experimentos mostraram que a abordagem com replicação total alcançou aceleração linear com respeito ao número de nós. Além disso, a proposta também alcançou ganho com relação à vazão de consultas processadas. Entretanto, o desempenho do projeto híbrido superou a abordagem de replicação total. O autor verificou que a fragmentação da maior tabela em fragmentos menores faz com que o SGBD gere planos de execução mais eficientes sobre essas tabelas menores. O tamanho reduzido da tabela permite que o SGBD utilize ordenação em memória, gerando planos de execução com *merge join* sempre que possível. Além da redução do custo da junção, os fragmentos menores também possibilitaram melhor utilização do *cache*. Tudo isso fez com que o projeto híbrido alcançasse aceleração superlinear (RÖHM, BÖHM et al., 2000). A estratégia do projeto híbrido em reduzir o tamanho das tabelas com grandes volumes de dados e o processamento de consultas através do paralelismo intra-consulta

se mostra como uma ótima alternativa para melhorar o desempenho de consultas típicas de ambiente OLAP.

2.6. Trabalhos Correlatos

Existem diversos trabalhos que abordam técnicas para fragmentação física de tabelas relacionais. Dentre as técnicas de fragmentação horizontal, as mais típicas são a fragmentação por função de *hashing*, a fragmentação circular e a fragmentação por faixa de valores (DEWITT e GRAY, 1992). Na fragmentação por função de *hashing*, uma função de *hashing* é aplicada sobre um atributo da relação, relacionando cada tupla a um fragmento. Na fragmentação circular, é feita uma associação seqüencial de cada tupla da relação a um fragmento sem levar em consideração o valor do atributo. Na fragmentação por faixa de valores são escolhidos um ou mais atributos baseado nas cláusulas dos predicados, sendo a faixa de valores obtida tomando como base o domínio dos atributos escolhidos.

Nas próximas seções analisaremos mais profundamente outros trabalhos correlatos. Primeiramente falamos sobre outras estruturas para indexação de dados multidimensionais: a árvore X e a técnica da pirâmide. Em seguida, descrevemos outras estratégias propostas para fragmentação de dados e processamento paralelo de consultas de aplicações OLAP, ente eles o *multi-dimensional hierarquical fragmentation* (MDHF), *Smashing Queries* (SmaQ) e *Smashing Queries Shrinking Space* (SmaQSS). Finalmente apresentamos as alternativas proprietárias atuais.

2.6.1. Estruturas Multidimensionais

Desde que (GUTTMAN, 1984) propôs seu método como uma semelhança a árvore B para o espaço multidimensional, as árvores R vêm cada vez sendo mais utilizadas em várias aplicações, desde CAD (*Computer Aided Design*) e bases de dados geográficas até nos sistemas de gerenciamento de séries de multimídia e de tempo. Com o tempo, vários pesquisadores e desenvolvedores vêm estudando artigos sobre a árvore R, implementando e testando novas variações. Apresentaremos nas próximas seções outras estruturas multidimensionais mais importantes que são derivadas da árvore R e que também poderiam ser utilizadas para fragmentar fisicamente os dados da tabela fato de um esquema estrela.

2.6.1.1. Árvore X

Assim como a árvore R, a árvore X (*eXtended node tree)* é uma estrutura para indexação de dados espaciais, mas sua característica principal é indexar dados em espaços de muitas dimensões. Seu diferencial é utilizar um algoritmo de divisão do nó que minimiza a sobreposição das regiões, utilizando o conceito de super nós. A idéia básica desse tipo algoritmo é manter a árvore tão hierárquica quanto possível e ao mesmo tempo evitar divisões que resultariam em mais sobreposição (BERCHTOLD, KEIM et al., 1996).

A sobreposição em estruturas de árvores ocorre quando certa região do espaço é coberta por mais de um hiper-retângulo dos nós de indexação da árvore. A sobreposição afeta o desempenho do processamento de consultas na medida em que elas necessitarão avaliar vários caminhos na árvore.

Em (BERCHTOLD, KEIM et al., 1996) foi feita uma avaliação de desempenho da árvore R com dados espaciais de várias dimensões. Os resultados mostraram que o desempenho da árvore R se deteriora rapidamente como aumento do número de dimensões. Para entender os efeitos que levavam a essa queda de desempenho, os autores estudaram as mais importantes características da árvore R e concluíram que a sobreposição das regiões aumenta rapidamente com o aumento do número de dimensões dos objetos.

A principal idéia da árvore X é evitar a sobreposição dos hiper-retângulos utilizando uma nova organização que é otimizada para espaços de muitas dimensões. A árvore X evita divisões que resultariam em alto grau de sobreposição. Ao invés de permitir divisões que introduzem grandes sobreposições, os nós são estendidos acima do tamanho usual do bloco resultando nos chamados super nós, onde passa a ser feita uma busca linear. A sobreposição gera uma rápida degeneração da filtragem de seletividade, fazendo com que seja necessário executar uma busca seqüencial em vários nós filhos, com a desvantagem de serem feitos vários acessos randômicos ao invés de uma leitura seqüencial nos super nós.

A árvore X consiste de três diferentes tipos de nós como ilustrado na figura 4: os nós folhas, que armazenam os dados propriamente ditos, os nós não folhas que são responsáveis pela indexação e os super nós. Os super nós são nós com capacidade de armazenamento maior que os nós não folhas normais, para evitar divisões. Em muitos casos, a criação ou extensão do tamanho do nó pode ser evitada utilizando um algoritmo

de divisão com sobreposição mínima, que determina um particionamento do nó de forma que a sobreposição dos hiper-retângulos dos dois conjuntos resultantes seja mínima.

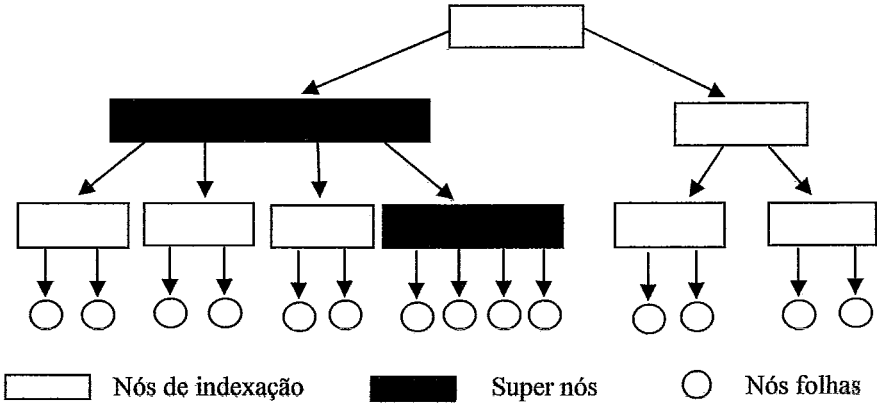


Figura 5 – Estrutura da árvore X.

A árvore X pode ser vista como uma estrutura híbrida, tanto como um vetor linear quanto como uma árvore R hierarquizada. Ela organiza dinamicamente a árvore de forma que os conjuntos de dados que poderiam produzir alto grau de sobreposição são organizados linearmente enquanto organiza hierarquicamente o conjunto de dados que puder ser organizado hierarquicamente sem muita sobreposição. Os algoritmos utilizados na árvore X são implementados para automaticamente organizar o conjunto de dados o mais hierárquico possível, resultando em uma organização híbrida do diretório. No caso em que a árvore X não possui nenhum super nó, ela está totalmente hierarquizada e se assemelha a estrutura da árvore R.

Os resultados do experimento mostraram que a árvore X obteve desempenho superior a árvore R em consultas com pontos e consultas de vizinho mais próximo. A principal limitação da solução é que os super nós podem ficar muito grandes para espaços com um número de dimensões muito grande, que pode tornar a leitura seqüencial muito custosa.

2.6.1.2. Técnica da Pirâmide

Assim como a árvore X, a técnica da pirâmide é um método de indexação multidimensional baseado em uma estratégia especial de particionamento otimizada para dados espaciais com muitas dimensões.

A idéia básica da técnica da pirâmide é transformar o espaço d-dimensional primeiramente em espaços piramidais bidimensionais, onde os topos das pirâmides estão localizados no ponto central do espaço. Em seguida é feito o particionamento das pirâmides em pedaços menores, cortando-as em fatias paralelas à sua base. Essas fatias representam as páginas de dados (BERCHTOLD, BÖHM et al., 1998). A figura 5 ilustra o particionamento de um espaço bidimensional.

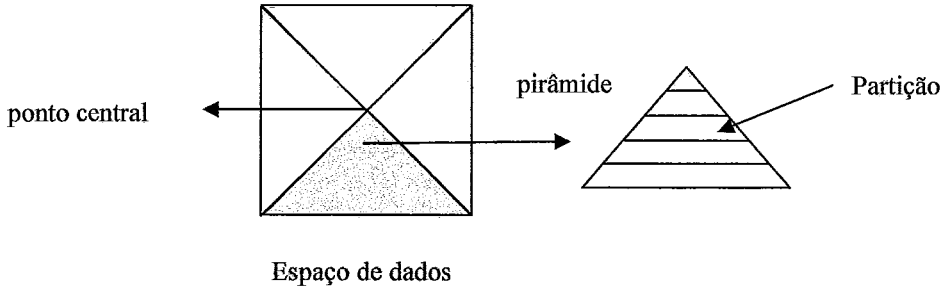


Figura 6 – Particionamento do espaço (BERCHTOLD, BÖHM et al., 1998)

Esse particionamento transforma o espaço d-dimensional em um espaço unidimensional, permitindo utilizar uma estrutura de índice eficiente como a árvore B+, que possui várias vantagens, ente elas, rapidez nas operações de inserção, atualização e remoção e facilidade de implementação. Os dados são indexados utilizando uma chave unidimensional e nas folhas da árvore são armazenados os dados d-dimensionais junto com a chave unidimensional. Dessa forma não é necessário prover uma transformação inversa.

As operações de inserção, atualização, remoção e busca, são processadas através da árvore B+. Em todos os casos, a entrada d-dimensional é transformada em um dado unidimensional para ser processado pela árvore B+. A transformação de um ponto d-dimensional em um valor é feita somando o índice da pirâmide correspondente e a altura do ponto dentro da pirâmide. Esse valor é definido como valor pirâmide do ponto. A altura de um ponto dentro de uma pirâmide é dada pela distância entre o ponto em questão e o ponto central do espaço, ou seja, o topo da pirâmide. O índice i da pirâmide correspondente ao ponto é dado pela dimensão i que possui maior desvio do centro do espaço.

Para inserção, é determinado o valor pirâmide do ponto a ser inserido. Esse valor é utilizado como chave para localização da página de dados onde o ponto deve ser

inserido na árvore B+. A chave e o ponto são armazenados na página de dados correspondente. As operações de atualização e remoção são feitas de forma análoga.

No final da criação do índice, as páginas de dados resultantes da árvore B+ contêm um conjunto de pontos que pertencem à mesma pirâmide e têm a propriedade em comum de que seus valores pirâmide estão contidos em um intervalo dado pelos valores mínimo e máximo das páginas de dados.

O processamento de consultas que determinam se um ponto p pertence ao espaço multidimensional, é feito calculando-se o valor pirâmide do ponto e consultando-se esse valor na árvore B+. O resultado da consulta é um conjunto de pontos que possuem o mesmo valor pirâmide do ponto p. Em seguida, é executada uma busca nesse conjunto para determinar se o ponto está contido no conjunto.

No caso de consultas por intervalos, ou seja, consultas que buscam os pontos no espaço que estão contidas dentro de um intervalo, o algoritmo de busca primeiramente determina as pirâmides que possuem interseção com o hiper-retângulo de busca e em seguida, dentro desse conjunto de pirâmides candidatas calculado previamente, determina os valores pirâmide que são relevantes para a consulta.

Uma extensão para a técnica da pirâmide é o tratamento de dados que não estão distribuídos uniformemente. Em um cenário onde os pontos concentram-se em uma área do espaço de dados, somente algumas pirâmides possuirão a maioria dos dados enquanto que as outras pirâmides ficarão praticamente vazias. O particionamento nesse caso não é eficiente. A proposta da extensão é transformar o espaço de dados de forma que a concentração de dados fique localizada no ponto central do espaço. Essa transformação é feita somente durante a determinação dos valores pirâmide dos pontos e não nos pontos propriamente ditos.

Os experimentos apresentados em (BERCHTOLD, BÖHM et al., 1998) mostram que a proposta possui desempenho superior a outras estratégias de particionamento no processamento de consultas de intervalos. Para consultas com baixa seletividade ou que especificam um número reduzido de atributos, a técnica ainda se mostra superior a outros métodos de indexação.

2.6.2. Projeto de Fragmentação e Processamento Paralelo para Data Warehouses

Nas próximas seções descrevemos outras propostas para fragmentação de dados e processamento paralelo de consultas específicas para aplicações OLAP, ente eles o *multi-dimensional hierarquical fragmentation* (MDHF), *Smashing Queries* (SmaQ) e *Smashing Queries Shrinking Space* (SmaQSS).

2.6.2.1. MDHF

A fragmentação hierárquica multidimensional (*multi-dimensional hierarquical fragmentation – MDHF*) é uma técnica de fragmentação e alocação desenvolvida para esquemas estrelas em um ambiente paralelo de *data warehouse*. A fragmentação é aplicada para particionar a tabela fato e seu principal objetivo é reduzir a carga de processamento e de operações de entrada e saída para diversas consultas através da redução do número de fragmentos a serem processados (STÖR, MÄRTENS et al., 2000).

A fragmentação da tabela fato é feita a partir da escolha de múltiplos atributos de fragmentação de diferentes níveis hierárquicos das dimensões. Cada atributo de fragmentação corresponde a uma dimensão diferente. Para cada atributo de fragmentação, é definido um particionamento por intervalo de valores disjuntos dentro do domínio do atributo. Por questões de simplificação, o trabalho foca na fragmentação pontual, onde cada intervalo na verdade corresponde a um valor do atributo de fragmentação. Cada fragmento consiste de todas as tuplas da tabela fato que pertencem a esse valor definido pelo atributo de fragmentação. Escolhas de atributos em um nível hierárquico maior geram fragmentos maiores enquanto que atributos em um nível hierárquico menor geram fragmentos menores.

O autor utiliza o esquema estrela baseado no *benchmark* de processamento analítico APB-1 (OLAP COUNCIL, 1998). Esse esquema provê um ambiente típico de vendas contendo uma tabela fato vendas e quatro dimensões: produto, cliente, canal e tempo. Uma fragmentação de duas dimensões que defina como atributos de fragmentação o campo mês da dimensão tempo e o campo grupo da dimensão produto, gera fragmentos que combinam todas as tuplas da tabela fato que referenciem um grupo de produto e um mês em particular. O número de fragmentos é determinado pelo produto da cardinalidade dos atributos de fragmentação utilizados. No exemplo acima,

o número de fragmentos gerados é o produto entre o total de grupos de produtos diferentes e o total de meses distintos da dimensão tempo.

A fragmentação multidimensional permite confinar a um subconjunto de fragmentos as consultas que referenciem pelo menos um atributo de fragmentação ou que utilizem a estrutura hierárquica das dimensões. Existem quatro tipos de consultas que são beneficiadas pela solução. O primeiro tipo são as consultas que referenciam diretamente o atributo de fragmentação. Consultas que referenciem todos os atributos de fragmentação podem ser restringidas ao menor conjunto de fragmentos. No melhor caso, consultas que possuem predicados que combinam exatamente com o atributo de fragmentação precisam acessar somente um fragmento. Nesse caso, o resultado da consulta é na verdade o próprio fragmento (STÖR, MÄRTENS et al., 2000).

O outro tipo de consulta é a que referencia níveis abaixo da hierarquia dos atributos de fragmentação. Da mesma forma que no tipo de consulta anterior, o número de fragmentos a serem acessados pode ser restrito a um conjunto mínimo de fragmentos. A única diferença é que somente algumas tuplas dos fragmentos são relevantes para consulta, já que ela acessa níveis inferiores da hierarquia do atributo de fragmentação.

O último tipo de consulta engloba aquelas que referenciam níveis superiores na hierarquia do atributo de fragmentação. Nesse caso, apesar de ainda possibilitar que a consulta acesse somente um subconjunto dos fragmentos, o número de fragmentos a serem acessados tende a ser maior que nos outros dois tipos de consulta, já que cada valor do atributo em uma hierarquia mais elevada corresponde a vários valores do atributo de fragmentação e, consequentemente, a um número maior de fragmentos correspondentes. A solução garante então que consultas que referenciem pelo menos um atributo de fragmentação possam ser confinadas a um conjunto de fragmentos.

A alocação física dos fragmentos foi feita utilizando o algoritmo de escalonamento *round robin*, alocando os fragmentos em discos consecutivos. Com os fragmentos alocados nos nós, outra possibilidade oferecida pela solução é o processamento paralelo de consultas. Depois da determinação dos fragmentos a serem processados através dos atributos da consulta original, uma sub-consulta é associada a cada um desses fragmentos e são então processadas paralelamente entre nós da rede.

Para validar a solução proposta no MDHF foi feita uma simulação implementada em C++ na arquitetura disco compartilhado, onde cada nó possui processador e memória próprios e tem acesso direto a uma unidade de armazenamento secundária

compartilhada. A base de dados utilizada na simulação é o *benchmark* APB-1 (*Analytical Processing Benchmark*). Um gerador de consultas passa as consultas criadas para um módulo de processamento responsável por direcionar as consultas entre os nós. Os experimentos foram feitos simulando um ambiente mono usuário, ou seja, consultas são solicitadas sequencialmente. Além disso, todas as consultas possuem um predicado de seleção sobre o mesmo atributo de fragmentação, variando randomicamente o valor do atributo.

Os autores analisaram diversas opções de fragmentação para a tabela fato *Sales* juntamente com diferentes consultas do *benchmark*. As fragmentações propostas são baseadas na característica das consultas escolhidas. Os resultados do experimento mostram um ganho próximo de linear em grande parte dos casos analisados.

As principais restrições da solução proposta é que a fragmentação ideal é obtida a partir da análise das características de um conjunto de consultas pré-estabelecidas. Ambientes OLAP com muitas consultas *ad-hoc* dificultariam a realização dessa análise prévia, podendo limitar o desempenho da solução em algumas situações.

2.6.2.2. Smashing Queries (SmaQ)

O SmaQ (LIMA, 2004) é um protótipo cuja proposta é acelerar a execução de consultas de alto custo através do paralelismo intra-consulta em um agrupamento de banco de dados. A solução utiliza a técnica de fragmentação virtual adaptativa (AVP – *Adaptative Virtual Partioning*) em uma base de dados totalmente replicada. Além da aceleração obtida através do paralelismo, a proposta aborda a implementação de um algoritmo distribuído para balanceamento dinâmico de carga entre os nós do agrupamento de banco de dados.

A técnica de fragmentação virtual consiste em reescrever uma consulta submetida ao agrupamento de banco de dados em uma série de sub-consultas, uma para cada nó, através da adição de predicados de seleção que definem intervalos correspondentes a diferentes fragmentos virtuais de uma relação. Como a base é totalmente replicada, cada sub-consulta pode então ser processada em paralelo por todos os nós do agrupamento.

A fragmentação virtual adaptativa é proposta no trabalho de (LIMA, MATTOSO et al., 2004) com objetivo de superar as desvantagens da fragmentação virtual simples (SVP – *Simple Virtual Partioning*) proposta em (AKAL, BÖHM et al.). Na SVP, o

tamanho do fragmento virtual é determinado pela cardinalidade da relação virtualmente fragmentada e pelo número de nós utilizados. Essa abordagem pode gerar fragmentos virtuais de tamanho grande o que não garante que o SGBD utilize obrigatoriamente o índice construído no atributo de fragmentação para recuperação das tuplas pertencentes a cada fragmento virtual. A utilização desse índice é um dos requisitos para garantir bom desempenho da técnica de fragmentação virtual, pois caso contrário um nó pode decidir fazer uma busca linear, recuperando toda a relação virtualmente fragmentada. A única maneira de resolver esse problema na SVP é aumentando o número de nós do agrupamento de banco de dados.

Além disso, um requisito da SVP é que os valores do atributo de fragmentação devem ser uniformemente distribuídos entre as tuplas da relação virtualmente fragmentada, o que não pode ser garantido que ocorra na prática. A distorção de dados faz com que sejam produzidos fragmentos virtuais com diferentes números de tuplas, o que pode gerar desbalanceamento inicial de carga entre os nós do agrupamento.

O objetivo da AVP é resolver esses problemas da SVP, mantendo sua flexibilidade e simplicidade. A solução proposta pela AVP é a redução dos fragmentos virtuais ajustando dinamicamente seu tamanho. Inicialmente, cada nó recebe uma sub-consulta para processar um intervalo que é calculado da mesma forma proposta pela SVP. Cada nó então subdivide seu intervalo inicial em sub-intervalos pequenos, que são processados seqüencialmente pelo SGBD.

A utilização de fragmentos menores evita o processamento de consultas através de buscas lineares e permite que um eventual desbalanceamento de carga seja resolvido alocando intervalos de um nó ocupado para um nó livre, que terminaram de processar todos os seus intervalos.

O processador de consultas do SmaQ é composto por um componente global que recebe as consultas submetidas ao agrupamento de PCs e repassa aos nós do agrupamento já fragmentada virtualmente. Por sua vez, cada nó executa sua sub-consulta utilizando a técnica da AVP e ao término da execução, retorna seu resultado ao componente global avisando que já processou todo seu intervalo. O componente global, ao ser notificado por todos os nós do agrupamento, faz a composição do resultado final e o retorna à aplicação cliente que submeteu a consulta.

Os experimentos da proposta foram realizados através da implementação de um protótipo em um agrupamento de banco de dados com 64 nós. O SGBD utilizado em todos os nós foi o banco de dados livre PostgreSQL e a base de dados e consultas do

benchmark TPC-H, específico para aplicações OLAP, que consiste de consultas de alto custo de processamento.

Os resultados obtidos mostraram ganhos com aceleração linear e superlinear em várias avaliações. Nos testes com processamento concorrente de consultas, a técnica proposta apresentou aumento superlinear de vazão em várias situações com distribuição de uniforme de dados. No cenário com distorção de dados o aumento da vazão também foi significativo. Os experimentos também mostraram que a AVP com redistribuição dinâmica de carga supera a SVP na maioria das situações e que a sobrecarga causada pelo balanceamento é desprezível.

Segundo o autor, as principais limitações da solução incluem a replicação total da base dados, pois exige consumo elevado de espaço físico em disco. Outras desvantagens são o não tratamento de atualizações de dados nesse ambiente e o grande número de mensagens trocadas pelos nós para fazer a redistribuição dinâmica de carga.

A partir desse trabalho, foi derivado o ParGRES (MATTOSO, ZIMBRÃO , LIMA, 2005), que é um software livre que funciona como um *middleware* em um agrupamento de banco de dados. Além de adotar a solução para a execução de consultas de alto custo do SmaQ, ele trata atualizações dos dados e faz o mapeamento das consultas centralizadas em sub-consultas que serão executadas em paralelo. A proposta também utiliza a replicação total da base de dados e a técnica de fragmentação virtual adaptativa (AVP) como algoritmo distribuído para balanceamento dinâmico de dados também proposto por (LIMA, 2004).

2.6.2.3. Smashing Queries Shrinking Space (SmaQSS)

Smashing Queries Shrinking Space (FURTADO C., 2006) é um protótipo originado do SmaQ, e, assim como o SmaQ (*Smashing Queries*), sua proposta é dividir as consultas em sub-consultas menores utilizando o algoritmo AVP, porém eliminando a necessidade da replicação toda da base de dados. Seu objetivo é reduzir a utilização de espaço em disco através da adoção de um projeto híbrido da base de dados (FURTADO, LIMA et al., 2006).

O particionamento híbrido (HP - *Hybrid Partitioning*) consiste em fragmentar as relações maiores por todos os nós do agrupamento de banco de dados. No caso de aplicações OLAP, a fragmentação é aplicada nas tabelas fatos e todas as dimensões são replicadas em todos os nós do agrupamento. O algoritmo do AHP consiste na utilização

do AVP com o HP para evitar a replicação total. A proposta inclui também um esquema de replicação de fragmentos com a distribuição por espalhamento encadeado de réplicas, possibilitando fazer o balanceamento dinâmico de carga entre os nós.

As tabelas são fragmentadas fisicamente de acordo com o número de nós do agrupamento de banco de dados. Um fragmento da tabela fato é definido por uma operação de seleção cujo predicado define um intervalo de valores do atributo de fragmentação. Durante a execução de uma consulta, a AHP divide novamente os fragmentos físicos em fragmentos virtuais menores, utilizando a AVP (FURTADO, LIMA et al., 2006).

Assim como no SmaQ o processador de consultas no SmaQSS, é composto por elementos de funções globais de funções locais e pelo catálogo. O catálogo, além de conter informações sobre o esquema de banco de dados, armazena informações de onde e como os fragmentos das tabelas fatos e suas cópias estão armazenados. Esses dados são utilizados para fazer balanceamento de carga entre os nós do agrupamento.

Para validar a solução proposta no SmaQSS foi implementado um protótipo de um processador de consultas em um agrupamento de PCs. Os experimentos foram realizados em um agrupamento com 32 nós com memória distribuída, utilizando o banco de dados livre PostgreSQL e os dados e consultas do *benchmark* TPC-H, específico para aplicações OLAP, consistindo-se de um conjunto de consultas tipicamente *ad-hoc* e de alto custo.

Os resultados dos experimentos mostraram que a utilização da técnica obtém aceleração linear e, muitas vezes, superlinear em vários cenários avaliados. A replicação parcial permitiu uma significativa redução de espaço em disco ocupado.

2.6.3. Alternativas Proprietárias

Nas próximas seções descrevemos os trabalhos relacionados à fragmentação de dados e execução paralela de consultas em extensões proprietárias de SGBDs famosos, entre eles o MySQL, Microsoft SQL Server, Oracle e o IBM DB2.

2.6.3.1. MySQL

MySQL (MYSQL, 2006) é um sistema gerenciador de banco de dados de código aberto muito popular, utilizado inclusive por aplicações comerciais, por prover alto desempenho aliado à facilidade de uso. Recentemente começou a atuar também na

área de paralelismo em agrupamentos de PC para ganho de desempenho e escabilidade, lançando o produto *MySQL Cluster.*

O objetivo do MySQL Cluster é oferecer as funcionalidades do SGBD MySQL em agrupamentos de PCs, provendo maior disponibilidade, aumento de vazão e escalabilidade em comparação a sistemas com um único sistema gerenciador de banco de dados.

O MySQL Cluster consiste em um conjunto de computadores, cada um executando um ou mais processos que incluem um servidor MySQL, um nó de dados, um servidor de gerenciamento e caso necessário, programas especializados de acesso a dados.

No modelo do MySQL Cluster existem três tipos de nós: nó de gerenciamento, o nó de dados, e o nó de SQL. O papel do nó gerenciamento é prover serviços de controle para os outros nós que fazem parte do agrupamento. Dessa forma, ele é responsável por disponibilizar dados de configuração, executar rotinas de backup e iniciar e parar a execução de processos. O nó de dados são os mais importantes, pois eles que são responsáveis por armazenar as tabelas fragmentadas e as réplicas. A replicação garante que caso ocorra a queda de um nó do sistema, a disponibilidade dos dados não seja afetada. Finalmente o nó de SQL é por responsável receber as requisições das aplicações cliente.

Com a replicação dos dados, o MYSQL Cluster permite que as consultas sejam processadas paralelamente. Somente o paralelismo inter-consultas é oferecido. Paralelismo intra-consulta não é suportado pelo MYSQL Cluster. Logo, a solução garante aumento da vazão das consultas, mas não provê diminuição do tempo de execução de consultas individualmente.

O MySQL Cluster trabalha os dados na memória principal, garantindo assim alto desempenho no processamento de consultas. Entretanto, restringe o tamanho máximo do banco de dados à soma da capacidade de memória principal do agrupamento de PC, pois exige que todo o banco de dados seja alocado nas memórias principais dos nós do agrupamento.

2.6.3.2. Microsoft SQL Server

O Microsoft SQL Server (SQLSERVER, 2005) é um SGBD proprietário da *Microsoft Corporation* que em sua versão 7.0 incluiu entre as suas novas características

o suporte a *data warehouse*. Entre o conjunto de novas possibilidades oferecidas para *data warehousing*, estão a fragmentação de tabelas e o processamento paralelo de consultas.

O particionamento de tabelas no SQL Server é feito através de visões particionadas, que divide a tabela fato em intervalos de dados. O conjunto de visões de uma tabela particionada é unido através de uma consulta (operador UNION) para visualização de todos os dados. A utilização de visões particionadas não impacta na complexidade da aplicação, pois a implementação física fica transparente para os métodos de acesso da aplicação (SWAMINATHAN, 2005).

O SQL Server oferece fragmentação horizontal por intervalos de dados, onde a menor unidade de particionamento é uma tupla da tabela. A aplicação permite que o usuário defina uma função de particionamento com os valores limites do intervalo de dados de cada partição. Essa função define a qual partição uma tupla ou índice da tabela pertence. Cada partição definidas por essas funções são mapeadas para um local de armazenamento através de um esquema de partição.

As consultas a serem executadas sobre tabelas particionadas podem utilizar um operador oferecido pelo SGBD chamado paralelismo sob demanda. Essa operação cria várias threads para executar varredura de tabelas, junções, ordenações e agrupamentos. O número de threads a serem criadas pela operação depende dos recursos disponíveis pelo ambiente no qual está inserido, ou seja, número de processadores disponíveis, número de transações concorrentes, memória principal disponível, entre outros.

A ferramenta permite que tabelas e índices possam ser fragmentados em até 1000 partições por tabela. Através das tabelas particionadas, o SQL Server permite fazer indexação paralela e obter aceleração em carga de dados.

2.6.3.3. Oracle Real Application Cluster 10g

A atual versão do SGBD comercial Oracle (Oracle 10g) também oferece suporte para clusters, através da extensão chamada Real Application Cluster (RAC) (ORACLE, 2003). O objetivo é prover alto desempenho e aumento da vazão oferecendo processamento de consultas com paralelismo intra-consulta e paralelismo inter-consulta. Logo é uma solução adequada tanto para ambientes OLTP quanto ambientes OLAP.

A arquitetura sugerida é a de disco compartilhado, onde todos nós do agrupamento têm acesso a um dispositivo de armazenamento de alto desempenho

chamada de *Storage Area Network* (SAN). Além disso, utiliza uma conexão de rede dedicada para comunicação interna do cluster.

Como todas as instâncias do cluster acessam a mesma base de dados armazenada na SAN, o SGBD utiliza uma tecnologia chamada *Cache Fusion* para coordenar as alterações dos dados em diferentes nós da rede. Assim, sempre que um nó requisitar um dado, o SGBD garante que ele receba a versão mais atual da informação, mesmo que o dado tenha sido alterado recentemente. A técnica de *Cache Fusion* permite trabalhar com dados em *cache* de cada nó como um *cache* global, unindo os diferentes *caches* separados fisicamente de cada nó em um *cache* único e global.

A desvantagem da solução é o alto custo necessário para implantá-la. Além do custo para adquirir a licença do SGBD, há a necessidade em hardware especializado para o disco compartilhado e para a rede de alta velocidade.

2.6.3.4. IBM DB2 Integrated Cluster Environment

O SGBD proprietário IBM DB2 também oferece uma extensão para agrupamento de PCs chamada *DB2 Integrated Cluster Environment* (ICE) (BIALEK, 2003). A solução se baseia na arquitetura de memória distribuída. Os dados são fragmentados fisicamente e alocados entre os nós do agrupamento. Para auxiliar o processo de fragmentação, o pacote disponibiliza uma ferramenta.

Um dos atrativos da solução é não exigir hardware especial, pois se baseia na arquitetura de memória distribuída. Além disso, dependendo no esquema físico da base, provê paralelismo intra-consulta e inter-consulta, o que o torna vantajoso para aplicações OLAP e OLTP. Entretanto, a solução ainda envolve consideráveis gastos com licença do produto.

2.7. Conclusão

Os sistemas distribuídos representam uma alternativa mais econômica financeiramente para alcançar maior poder computacional, em detrimento dos supercomputadores com arquitetura paralela, caros tanto em relação ao hardware e software especializados dos quais necessitam. Além disso, são opções mais flexíveis na medida em que permitem um crescimento incremental através da adição de novos dispositivos caso seja necessário adquirir maior poder computacional.

Essas vantagens fizeram que os sistemas distribuídos ganhassem destaque e seus conceitos passaram a ser utilizados em sistemas distribuídos de banco de dados. Entre os trabalhos relacionados, destaca-se o agrupamento de banco de dados, que são considerados uma plataforma bem atraente para processamento paralelo e aceleração de consultas.

Para garantir ganho de desempenho com bancos de dados distribuídos, é preciso ter um bom projeto de distribuição de dados definido. Em ambientes de *data warehouse*, o problema é que as consultas são geralmente complexas, de alto custo e acessam grandes volumes de dados. A solução mais adequada para obter desempenho com esse tipo de consulta é o processamento paralelo intra-consulta, que como vimos, oferece aceleração de consultas individualmente.

Outro fator a ser levado em consideração durante o projeto de distribuição de dados é a característica do esquema de banco de dados. Os *data warehouses* caracterizam-se por acumularem grandes volumes de dados históricos. Em *data warehouses* baseados no esquema estrela, a base de dados consiste em tabelas fato que concentram grande volume de dados e em várias tabelas dimensão. A fragmentação física da tabela fato é uma opção interessante para melhorar o desempenho das consultas OLAP, que em sua maioria acessam esse tipo de tabela, pois permite restringir o volume de dados a ser acessado pela consulta. Além disso, os fragmentos menores são processados mais eficientemente pelos SGBDs.

Logo, a fragmentação física aliada ao processamento paralelo intra-consulta se mostram boas alternativas para ganho de desempenho em ambientes de DW. Entretanto, poucos trabalhos aproveitam a característica dimensional do esquema estrela para fragmentá-lo fisicamente. Nesse contexto, surge a motivação para nossa dissertação, onde propomos utilizar uma estrutura de indexação multidimensional como método para fragmentar fisicamente a tabela fato de um esquema estrela.

No próximo capítulo apresentamos detalhadamente nossa proposta de fragmentação física utilizando a árvore R, utilizada para geração dos fragmentos físicos da tabela fato de um esquema estrela. Posteriormente, mostraremos como a utilizamos para fazer processamento intra-consulta e limitar o volume de dados a ser acessado pelas consultas.

3. Proposta de Fragmentação Multidimensional utilizando Árvore R

O propósito deste capítulo é mostrar como utilizamos uma estrutura multidimensional, no nosso caso a árvore R, para fragmentar fisicamente a tabela fato de um esquema estrela e como podemos utilizar essa estrutura para processar consultas utilizando paralelismo intra-consulta.

Iniciamos o capítulo com a seção 3.1 onde explicamos nossa proposta de fragmentação utilizando a árvore R. Na seção 3.2 abordamos como fazemos a alocação dos fragmentos gerados entre os nós da rede. Na seção 3.3 descrevemos como a fragmentação física gerada pela árvore R restringe o volume de dados a ser acessado pelas consultas e como processamos as consultas em paralelo.

3.1. Fragmentação Multidimensional

Data warehouses armazenam grande volume de dados de diversas fontes e são utilizados geralmente por sistemas de apoio à decisão. Eles têm que ser capazes de processar consultas complexas e de natureza *ad-hoc*. Além disso, tendem a ser utilizados por vários usuários o que requer a execução concorrente de várias consultas complexas. Nosso objetivo é garantir tempos de resposta melhores nesse ambiente através da fragmentação física da tabela fato e do processamento paralelo de consultas.

Nossa solução procura utilizar as características das consultas e a forma como os dados são organizados em *data warehouses*. Nós focamos em *data warehouses* relacionais que organizem seus dados através da modelagem multidimensional. Nossa abordagem consiste em aproveitar as características da modelagem multidimensional para fragmentar a tabela fato utilizando um índice multidimensional.

Os índices multidimensionais organizam um espaço de várias dimensões em sub-regiões para aumentar a velocidade de recuperação de dados multidimensionais na medida em que permitem evitar percorrer todos os elementos de um determinado conjunto para determinar aqueles que devem ser realmente recuperados. Nossa estratégia é aproveitar o particionamento feito por essas estruturas multidimensionais para fragmentar fisicamente uma base de dados multidimensional.

A estrutura multidimensional que escolhemos para implementar nossa solução foi a árvore R, por ser uma estrutura amplamente difundida e por sua simplicidade. A

árvore R é uma estrutura que organiza um espaço multidimensional em hiper-retângulos agrupando pontos próximos espacialmente (GUTTMAN, 1984). Como nosso objetivo é trabalhar com bases de grandes volumes de dados, decidimos implementar a variante da árvore R que utiliza o algoritmo de clusterização k-médias para dividir o espaço multidimensional a ser indexado (BRAKATSOULAS, PFOSER D. et al., 2002).

Nossa proposta é aproveitar a característica multidimensional da modelagem utilizada em esquemas estrelas e utilizar a árvore R para fragmentar fisicamente a tabela fato em várias tabelas menores. A fragmentação física proporcionada pela árvore R possibilita restringir o volume de dados a ser acessado por determinada consulta, pois a árvore R agrupa pontos espacialmente próximos. Além disso, a fragmentação física permite que essas tabelas sejam acessadas paralelamente. Como uma das principais características das tabelas fatos é terem um grande volume de dados, a redução do volume de dados a ser acessado e o acesso paralelo possibilita maior velocidade de execução de consultas.

O esquema estrela consiste em uma tabela de fatos e um conjunto de tabelas dimensão. A tabela fato representa a medida numérica do negócio modelado. Ela armazena também as chaves estrangeiras das dimensões referentes a ela. Geralmente, o conjunto dessas chaves estrangeiras é a chave primária composta da tabela fato.

A nossa proposta é utilizar o conjunto de chaves estrangeiras como chave de indexação da árvore R. Consideramos que cada chave estrangeira representa uma coordenada multidimensional e o conjunto dessas coordenadas compõe um ponto multidimensional. Logo, cada tupla da tabela fato corresponde a um ponto multidimensional, definido pelos valores dos atributos que são chaves estrangeiras das dimensões. Nosso algoritmo recupera da tabela fato todos os pontos multidimensionais e os insere na árvore R.

Como vimos no capítulo 2, na árvore R os nós não folhas são responsáveis pela indexação e os nós folhas armazenam os objetos espaciais propriamente ditos. No nosso caso, os objetos espaciais são pontos multidimensionais recuperados da tabela fato. Assim, cada nó folha da árvore é composto por um conjunto de atributos da tabela fato, que correspondem às chaves estrangeiras das dimensões. A figura 7 ilustra o processo de criação da árvore R, utilizando como exemplo o esquema estrela apresentado na figura 1. A tabela fato vendas está relacionada às dimensões loja, produto e tempo e possui como métricas os campos quantidade e preço unitário. Cada tupla da tabela registra as informações da venda: o produto que foi comprado, em que loja da rede foi

feita a venda, em que data, a quantidade comprada e o preço unitário do produto no momento da venda. Os campos que são chaves estrangeiras para as dimensões são transformados em um ponto tridimensional que é inserido na árvore gerando dois nós folhas que particionam o conjunto de dados do exemplo.

FATO VENDAS				
Loja_ID	**Produto_ID**	**Tempo_ID**	**Quantidade**	**Preço_Unitário**
24	9	47	2	100,00
32	24	42	1	20,50
34	49	22	5	2,20
36	5	11	6	5,60
41	9	2	10	1,30
16	31	3	1	4,65
21	33	6	20	6,45
		...		

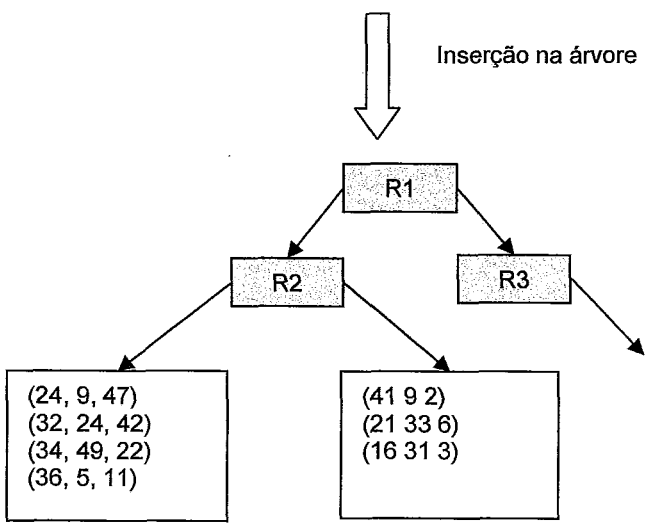
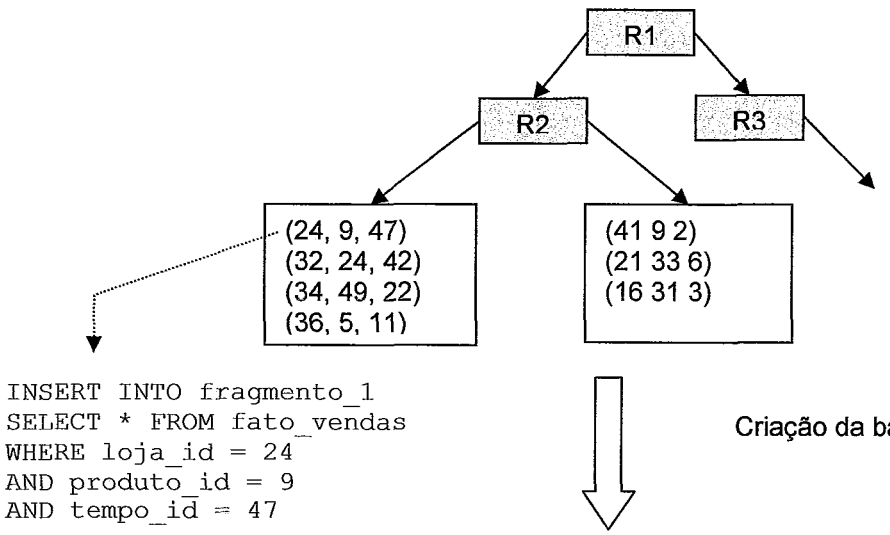


Figura 7 – Processo de criação da árvore R.

Com o particionamento dos dados definido pela árvore R, é possível criar os fragmentos físicos. Cada nó folha da árvore corresponderá a um fragmento, ou seja, para cada nó folha, será criada uma tabela contendo as tuplas armazenadas no nó. Como cada entrada do nó é um ponto multidimensional composto por chaves estrangeiras para as dimensões, é possível recuperar na tabela fato, as tuplas correspondentes a cada ponto multidimensional. Caso esses atributos definam a chave primária da tabela fato, cada ponto multidimensional corresponderá a uma única tupla. As tuplas recuperadas da tabela fato são inseridas no fragmento correspondente ao nó. Logo, o conjunto de tuplas recuperado a partir das entradas de um nó da árvore define as tuplas que devem estar

contidas em um dos fragmentos da tabela fato. Todos os fragmentos são gerados repetindo esse processo para todos os nós folhas da árvore R. O processo de criação dos fragmentos é ilustrado na figura 8, seguindo o exemplo do processo de criação da árvore para a tabela fato vendas.




```
INSERT INTO fragmento_1
SELECT * FROM fato_vendas
WHERE loja_id = 24
AND produto_id = 9
AND tempo_id = 47
```

Criação da base fragmentada

FRAGMENTO 1				
Loja_ID	**Produto_ID**	**Tempo_ID**	**Quantidade**	**Preço_Unitário**
24	9	47	2	100,00
32	24	42	1	20,50
34	49	22	5	2,20
36	5	11	6	5,60

FRAGMENTO 2				
Loja_ID	**Produto_ID**	**Tempo_ID**	**Quantidade**	**Preço_Unitário**
41	9	2	10	1,30
16	31	3	1	4,65
21	33	6	20	6,45

Figura 8 – Processo de criação dos fragmentos.

Nós incluímos um atributo na estrutura do nó da árvore que armazena o endereço do nó no disco. Como o endereço físico de armazenamento do nó é único, esse atributo funciona como um identificador do nó. Devido a essa característica, nós o utilizamos na composição do nome dos fragmentos. O nome de cada fragmento é composto pela concatenação do nome da tabela fato com o endereço físico do nó. Esse esquema permite manter a associação entre um nó folha da árvore e sua tabela fragmento correspondente. Como veremos com mais detalhes na próxima seção sobre

processamento de consultas, essa informação será utilizada para saber em que fragmentos certa consulta deve ser executada.

O tamanho dos fragmentos impacta diretamente no desempenho obtido na execução de consultas na base fragmentada. A geração de vários pequenos fragmentos não é ideal, pois aumenta excessivamente os custos de administração dos fragmentos e deteriora o desempenho com muitas operações de entrada e saída. Por sua vez, o tamanho dos fragmentos não pode ser muito grande, pois apesar de termos menos fragmentos para processar, fragmentos muito grandes degradam o desempenho das consultas, pois o SGBD pode fazer uma busca seqüencial e não utilizar os índices das tabelas. Além disso, poucos fragmentos podem subutilizar uma rede com vários processadores e discos. O ideal é que exista pelo menos um fragmento para cada nó da rede capaz de processar consultas. Logo, o tamanho da página dos nós da árvore é um parâmetro fundamental para efetivamente alcançar um bom desempenho com a base fragmentada.

A árvore R permite definir o tamanho dos fragmentos físicos através de um dos parâmetros definidos na sua criação. Esse parâmetro define o tamanho da página no disco dos nós, ou seja, o número máximo de bytes que cada nó é capaz de armazenar. Logo, esse parâmetro permite definir o tamanho dos fragmentos. Quanto maior a página, maior é a capacidade de armazenamento de entradas do nó e, consequentemente, maior será a quantidade média de tuplas dos fragmentos. Além disso, esse parâmetro também define a quantidade de nós que a árvore possuirá. Quanto menor o tamanho da página, mais nós serão necessários para armazenar todas as entradas. Em conseqüência mais fragmentos com poucas tuplas serão gerados.

Uma estimativa inicial para o tamanho da página do nó é entre 8K e 64K, pois é o tamanho do bloco no disco. Assim, com somente um acesso podemos ler o fragmento inteiro. Além disso, aumenta as chances dos dados estarem armazenados seqüencialmente no disco. Entretanto, como esses valores são pequenos, nem sempre é possível definir o tamanho da página com eles, pois se a tabela fato a ser fragmentada armazenar um grande volume de dados, páginas de 8K vão gerar muitos fragmentos, o que pode acarretar nos problemas descritos anteriormente.

Com a base fragmentada criada, as consultas que acessam a tabela fato podem ser direcionadas para acessar somente os fragmentos que possuam dados relevantes à consulta. Para obter melhor desempenho, o ideal é que seja preciso acessar o menor número possível de fragmentos. O pior caso é uma consulta que precise acessar todos os

fragmentos, o que pode degradar seu desempenho. Logo, um dos nossos objetivos para uma boa fragmentação física é reduzir a quantidade de fragmentos acessados por consulta.

Para atingir tal objetivo, nossa proposta é sempre que possível tentar alocar dentro de um mesmo fragmento tuplas relacionadas. Como as tabelas dimensão são altamente desnormalizadas, vários valores de atributos que representam hierarquias da dimensão se repetem em várias tuplas. Voltando ao exemplo do esquema estrela de vendas, os analistas de negócio frequentemente necessitam saber o total de vendas agrupado por loja de um produto de certa marca. Para esse tipo de consulta, é interessante que todas as tuplas da tabela fato que se referem aos produtos com a mesma marca estejam alocadas em um conjunto restrito de fragmentos. Assim, poucos fragmentos precisarão ser acessados para resolver a consulta.

Precisamos então agrupar nos fragmentos as tuplas que possuam mesmos valores para alguns atributos de dimensões. No caso do exemplo acima, é interessante agrupar os produtos de mesma marca. A árvore agrupa pontos próximos espacialmente, mas somente indexamos na árvore valores numéricos associados à dimensão, que no nosso caso é o identificador da tupla. Para aumentar a probabilidade de que produtos da mesma marca fiquem próximos, a solução que desenvolvemos foi gerar novos valores para os campos identificadores das dimensões ordenando pelos campos mais frequentemente utilizados como predicados de seleção nas consultas. Dessa forma, diminuímos a chance de que os valores dos identificadores de produtos da mesma marca possuam valores totalmente díspares, evitando assim que essas tuplas fiquem espalhadas pelos fragmentos. Uma consulta que possua um predicado de seleção sobre um desses atributos de dimensão, fica restrita a um conjunto menor de fragmentos.

Ilustrando com outro exemplo, seja um esquema estrela com uma dimensão cidade, que armazene as informações referentes às cidades onde uma rede de supermercados atua, onde grande parte das consultas sobre essa dimensão faz uma operação de seleção sobre o atributo estado. Como a árvore R agrupa pontos próximos, ao gerar os identificadores dessa dimensão ordenados por esse atributo, essas tuplas ficarão mais próximas e tenderão a ficar agrupadas no nó folha e, consequentemente, no mesmo fragmento, evitando que os dados relacionados fiquem espalhados pelos fragmentos. Com os dados mais agrupados, as consultas com predicado de seleção sobre o atributo estado somente precisarão acessar um conjunto reduzido de fragmentos.

3.2. Alocação dos fragmentos

Depois de definida a maneira como a fragmentação será feita, é preciso definir como os fragmentos serão alocados entre os nós da rede. Nossa solução não prevê replicação dos fragmentos, ou seja, cada fragmento só está alocado em um nó da rede. Mas é possível estender nossa solução para suportar replicação parcial ou total dos fragmentos, tornando possível adequá-lo à necessidade desejada de disponibilidade, confiabilidade e flexibilidade do sistema.

Nós utilizamos o algoritmo de escalonamento *round-robin*, ou seja, cada fragmento é alocado sequencialmente a um nó da rede. Quando um fragmento é alocado ao último nó da rede, os próximos fragmentos são alocados iniciando novamente no primeiro nó da rede e assim sucessivamente. Fragmentos consecutivos armazenam pontos próximos espacialmente. A alocação de fragmentos consecutivos em discos diferentes permite que uma consulta que acesse uma região multidimensional, armazenada por um conjunto de nós folha, ou seja, em fragmentos consecutivos, possa ser processada paralelamente.

Nós iniciamos o processo de criação da base fragmentada percorrendo a árvore R gerada com os dados da tabela fato a ser fragmentada. Para cada nó folha, é criada uma tabela no próximo nó da rede a receber um fragmento. Para cada entrada do nó folha, recuperamos na tabela fato de origem as tuplas correspondentes ao ponto multidimensional. Dessa forma, carregamos o fragmento com todas as tuplas que pertencem à região coberta pelo menor retângulo envolvente que descreve o nó folha. Esse processo é repetido até que todos os nós folha da árvore tenham sido percorridos.

A figura 9 ilustra o processo de criação e alocação dos fragmentos, gerando a base fragmentada. A fragmentação gerada pela árvore gerou n fragmentos, alocados consecutivamente em y discos. Finalmente, como as tabelas dimensão possuem volume de dados bem menor em relação às tabelas fato, não aplicamos nenhum tipo de fragmentação sobre elas. Elas são totalmente replicadas em todos os nós da rede.

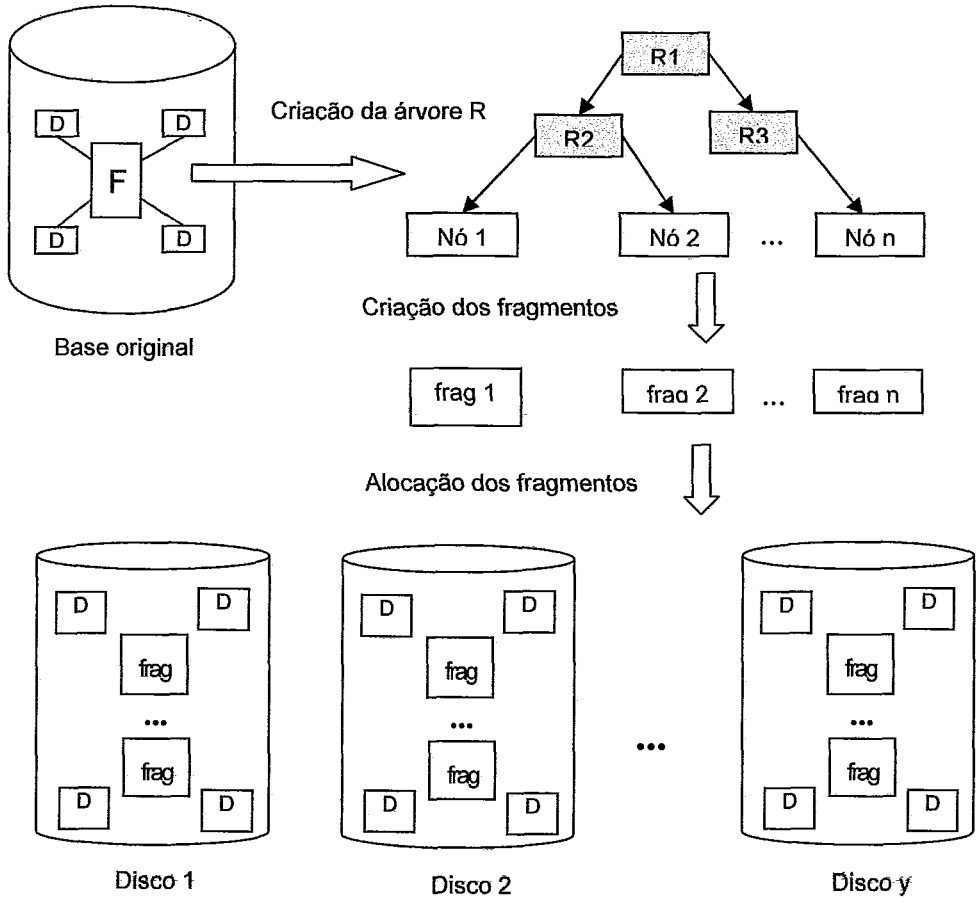


Figura 9 – Criação e alocação dos fragmentos.

3.3. Processamento de consultas

Como a árvore R foi utilizada para indexar os pontos multidimensionais da tabela fato, além de definir sua fragmentação, ela se torna a estrutura utilizada para fazer o direcionamento de uma determinada consulta entre os fragmentos. Através dela, é possível descobrir o conjunto de fragmentos que possivelmente possuirão dados relevantes para uma dada consulta. Com esse conjunto definido, basta subdividir a consulta original em sub-consultas, cada uma acessando um fragmento e disparar cada uma dessas sub-consultas na base fragmentada. Finalmente, o resultado final é composto pela consolidação do resultado de cada execução.

A limitação do conjunto de fragmentos a ser acessado por determinada consulta só é possível quando a consulta possui algum predicado de seleção sobre uma das dimensões que participaram da fragmentação da tabela fato. Caso a consulta não utilize nenhum predicado de seleção sobre alguma dimensão que foi mapeada em coordenada na árvore R ou ainda caso a consulta não utilize nenhum predicado de seleção, inevitavelmente a consulta deverá ser executada em todos os fragmentos. Essas situações representam o pior caso dentro da nossa solução, mas mesmo assim a solução

permite alcançar algum ganho, pois ela possibilita que a execução da consulta seja paralelizada. Além disso, cada sub-consulta será executada sobre tabelas menores, o que faz com que o SGBD utilize índice e dessa forma obtenha ganho de desempenho, que não ocorreria caso a consulta fosse executada na base não fragmentada. Vários SGBDs deixam de utilizar o índice quando estimam que o processamento da consulta irá acessar um número de tuplas maior que determinado limite, acessando a relação através de uma busca linear (LIMA, 2004).

O algoritmo de busca da árvore R recebe como entrada um hiper-retângulo de busca que descreve uma região multidimensional. Cada entrada do nó interno armazena um ponteiro e o hiper-retângulo mínimo que engloba todas as entradas do nó filho correspondente (MBR - menor hiper-retângulo envolvente). Os nós folhas armazenam os objetos espaciais propriamente ditos. No nosso caso, os objetos espaciais são pontos multidimensionais. Para cada menor hiper-retângulo envolvente de um nó, o procedimento de busca verifica se existe interseção com o hiper-retângulo de busca. Em caso positivo, a busca também deve ser feita no nó filho correspondente. A busca prossegue recursivamente até que todos os hiper-retângulos com interseção sejam processados. Ao alcançar um nó folha, os pontos que possuem interseção com o hiper-retângulo de busca são retornados pelo resultado da consulta na árvore. Na verdade, para nós basta saber quais nós que possuem pelo menos um ponto multidimensional que faz interseção com o hiper-retângulo de busca, pois os fragmentos correspondentes a esses nós já se tornam candidatos a possuírem dados relevantes para a consulta. Por isso, no nosso caso o procedimento de busca não precisa descer até o nível dos nós folha para chegar ao resultado. Basta descer até o último nível dos nós internos, ou seja, o penúltimo nível da árvore. Para cada nó desse nível é verificado se o menor retângulo envolvente de cada entrada possui interseção com o retângulo de busca. Em caso afirmativo, o nó filho apontado pela entrada correspondente é um candidato a possuir dados relevantes à consulta. Repetindo esse procedimento até que todos os nós candidatos sejam verificados, nosso procedimento de busca na árvore R retorna uma lista com identificadores desses nós. Como visto na seção anterior, esses identificadores são endereços físicos do nó no disco e são utilizados para compor o nome dos fragmentos. Dessa forma, temos os nomes das tabelas onde as sub-consultas deverão ser executadas.

A figura 10 ilustra o procedimento de busca na árvore. Cada entrada dos nós internos armazena o menor hiper-retângulo envolvente do nó filho correspondente. Os

nós folhas, presentes no último nível da árvore, armazenam os pontos multidimensionais que representam tuplas da tabela fato associada. A busca inicia-se no nó raiz onde para cada entrada é verificado se seu MBR possui intersecção com o retângulo de busca. Caso positivo, a busca recomeça nos nós filhos correspondentes. Os MBR que possuem interseção com o hiper-retângulo de busca estão preenchidos com a cor cinza.

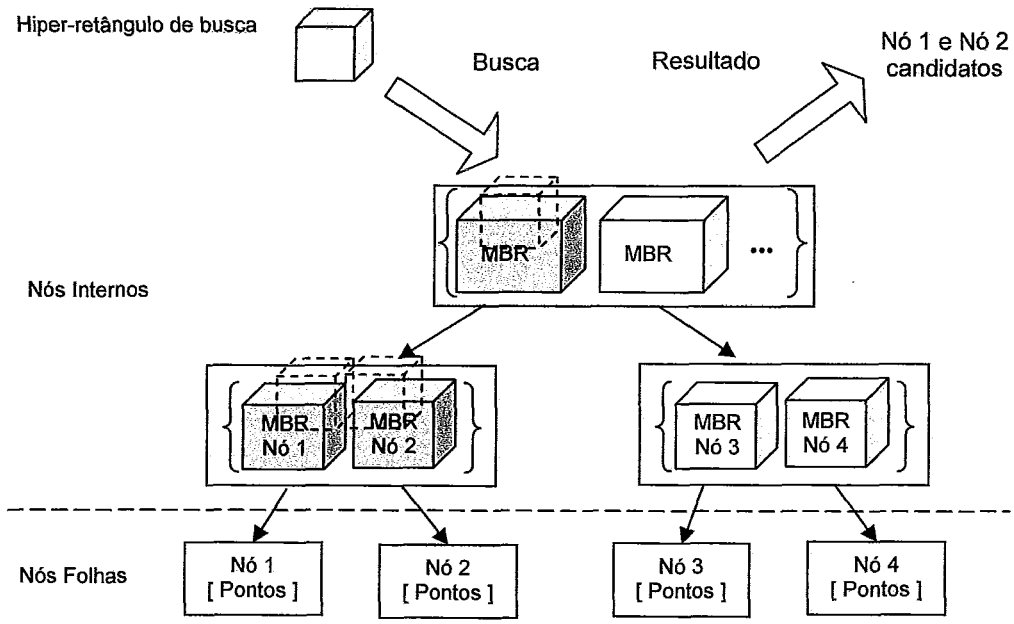


Figura 10 – Processo de busca na árvore.

Um retângulo de duas dimensões pode ser definido apenas por dois pontos: um que representa o canto superior esquerdo e o outro que representa o canto inferior direito. Os outros dois pontos podem ser calculados a partir deles. Chamamos esse dois pontos de coordenada máxima e coordenada mínima. Essa representação pode ser estendida para hiper-retângulos, pois assim como o retângulo de duas dimensões, seus lados formam ângulos retos entre si. Sempre representamos hiper-retângulos através dessas duas coordenadas. Para definir o hiper-retângulo de busca, recuperamos o hiper-retângulo do nó raiz da árvore, ou seja, o menor retângulo que engloba todo o espaço representado pela árvore e atribuímos ao nosso hiper-retângulo de busca a coordenada máxima e a coordenada mínina do hiper-retângulo da raiz da árvore.

Dessa forma, nosso hiper-retângulo de busca se restringe ao conjunto de pontos multidimensionais armazenado na árvore, que representa os dados da tabela fato que foi previamente fragmentada. Entretanto, não basta que o hiper-retângulo de busca se

restrinja ao conjunto de dados da tabela fato. Ele precisa se restringir também ao conjunto de dados relevantes para uma dada consulta, para que a busca na árvore retorne somente os nós que possuam dados que satisfazem ao critério de seleção da consulta. Logo, a definição do hiper-retângulo de busca também deve ser feita através da análise dos predicados de seleção da consulta.

Para descobrir o conjunto de fragmentos que devem ser acessados por determinada consulta, precisamos transformar a consulta original em uma consulta espacial, que é então passada como parâmetro de entrada para o algoritmo de busca da árvore R. Para transformar a consulta, verificamos se ela possui algum predicado de seleção sobre uma das dimensões que foram mapeadas em coordenada multidimensional. Em caso afirmativo, fazemos uma consulta na tabela dimensão correspondente para determinar o valor máximo e o valor mínimo do atributo em questão. Como esse atributo corresponde a uma coordenada do espaço de dados da árvore R, podemos transformar a consulta em uma região limitada ao longo da coordenada restringida pelo predicado de seleção, utilizando esses valores para determinar a coordenada mínima e a coordenada máxima do hiper-retângulo de busca.

O hiper-retângulo de busca é então formado primeiramente atribuindo-se a ele o hiper-retângulo da raiz da árvore, que restringe a consulta a todo o espaço indexado pela árvore e, em seguida, substituímos o valor máximo e o valor mínimo da coordenada que é restringida por um predicado de seleção com valores previamente calculados a partir da consulta na tabela dimensão correspondente. Essa substituição é feita em todas as coordenadas que são restringidas por algum predicado de seleção na consulta original. Com o hiper-retângulo de busca definido, basta repassá-lo como parâmetro para o algoritmo de busca da árvore R que retornará a lista de fragmentos que possivelmente possuem dados relevantes para a consulta original.

Dependendo do predicado de seleção, a consulta na tabela dimensão pode retornar um único valor ou um intervalo de valores. No caso em que a consulta retorna um único valor, é atribuído esse mesmo valor tanto à coordenada mínima quanto a coordenada máxima do hiper-retângulo de busca. E conforme explicado acima, caso a consulta retorne um intervalo de valores, atribuímos à coordenada máxima o valor máximo do intervalo e de forma análoga, atribuímos à coordenada mínima, o valor mínimo do intervalo.

No caso de consultas que possuem mais de um predicado de seleção sobre diferentes dimensões mapeadas em coordenadas e utilizadas para fragmentar a tabela

fato, para cada um desses predicados, é feita a busca na árvore. Cada busca retorna sua respectiva lista de fragmentos candidatos para consulta. A figura 11 ilustra esse processo. A consulta do exemplo acessa uma tabela fato previamente fragmentada e possui dois predicados de seleção que restringem os dados ao longo de duas dimensões que participaram da fragmentação da tabela fato. Para cada uma das duas dimensões mapeadas em coordenada, fazemos uma consulta na tabela dimensão correspondente para descobrir o intervalo delimitado pelo predicado de seleção correspondente, ou seja, o valor mínimo e o valor máximo dentre os identificadores retornados por cada consulta. Em seguida, basta utilizar esses valores para definir o hiper-retângulo de busca para descobrir que nós da árvore possuem dados relevantes para a consulta original.

Cada lista representa os fragmentos interessantes ao seu predicado de seleção correspondente. Caso a consulta seja conjuntiva, os dados relevantes para determinada consulta são aqueles que são satisfeitos por todos os predicados de seleção da consulta. Logo, a lista final de fragmentos candidatos é então composta pela interseção entre as listas de fragmentos sugeridos por cada predicado de seleção. Caso a consulta seja disjuntiva, a lista final de fragmentos candidatos é composta pela união das listas de fragmentos de cada predicado de seleção.

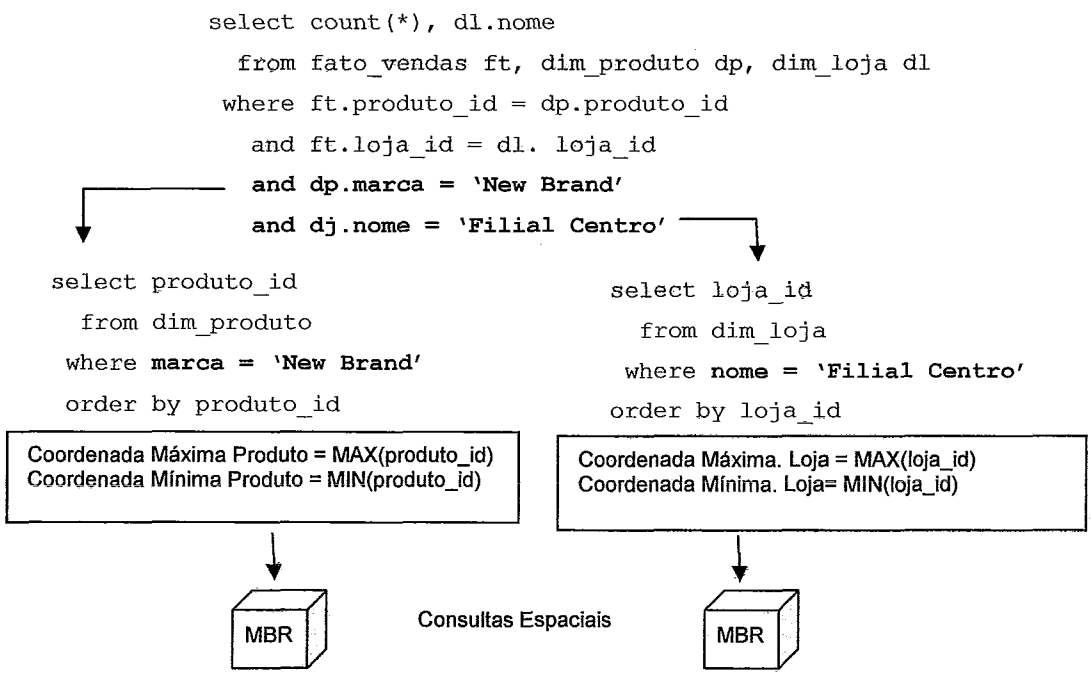


Figura 11 – Transformação da consulta original em consultas espaciais

Finalmente, com a lista de fragmentos candidatos é possível montar as sub-consultas que devem ser executadas na base fragmentada. A lista de fragmentos

armazena o identificador do nó folha de origem. O nome do fragmento é formado através da concatenação do nome da tabela fato original com o identificador do nó folha. Logo, para cada elemento da lista basta definir sua sub-consulta alterando na cláusula de FROM da consulta original o nome da tabela fato pelo nome do fragmento correspondente.

Vamos ilustrar através do exemplo da figura 12 todas as fases do processamento de consultas descrito nesta seção. A figura mostra que o processo inicia-se com a transformação da consulta original em consultas espaciais através da geração de hiper-retângulos de busca, um para cada predicado de seleção da consulta que acessa alguma das dimensões mapeadas em coordenadas. Em seguida, são feitas buscas na árvore R que particiona os dados da tabela fato. É executada uma busca para cada hiper-retângulo de busca. Cada busca na árvore retorna a lista de nós candidatos a possuírem dados que satisfazem o predicado de seleção correspondente ao hiper-retângulo de busca passado como parâmetro. A próxima etapa consiste em calcular a interseção entre as listas de nós candidatos, ou seja, estamos interessados nos nós que possuem dados relevantes para todos os predicados de seleção da consulta. Em seguida, para cada nó candidato podemos recuperar o nome da tabela do fragmento correspondente e então escrever as sub-consultas que devem ser executadas na base fragmentada. As sub-consultas são direcionadas para os discos que contém o fragmento que precisa acessar. Na medida em que as sub-consultas terminam suas execuções, os resultados parciais são agregados para finalmente compor o resultado final.

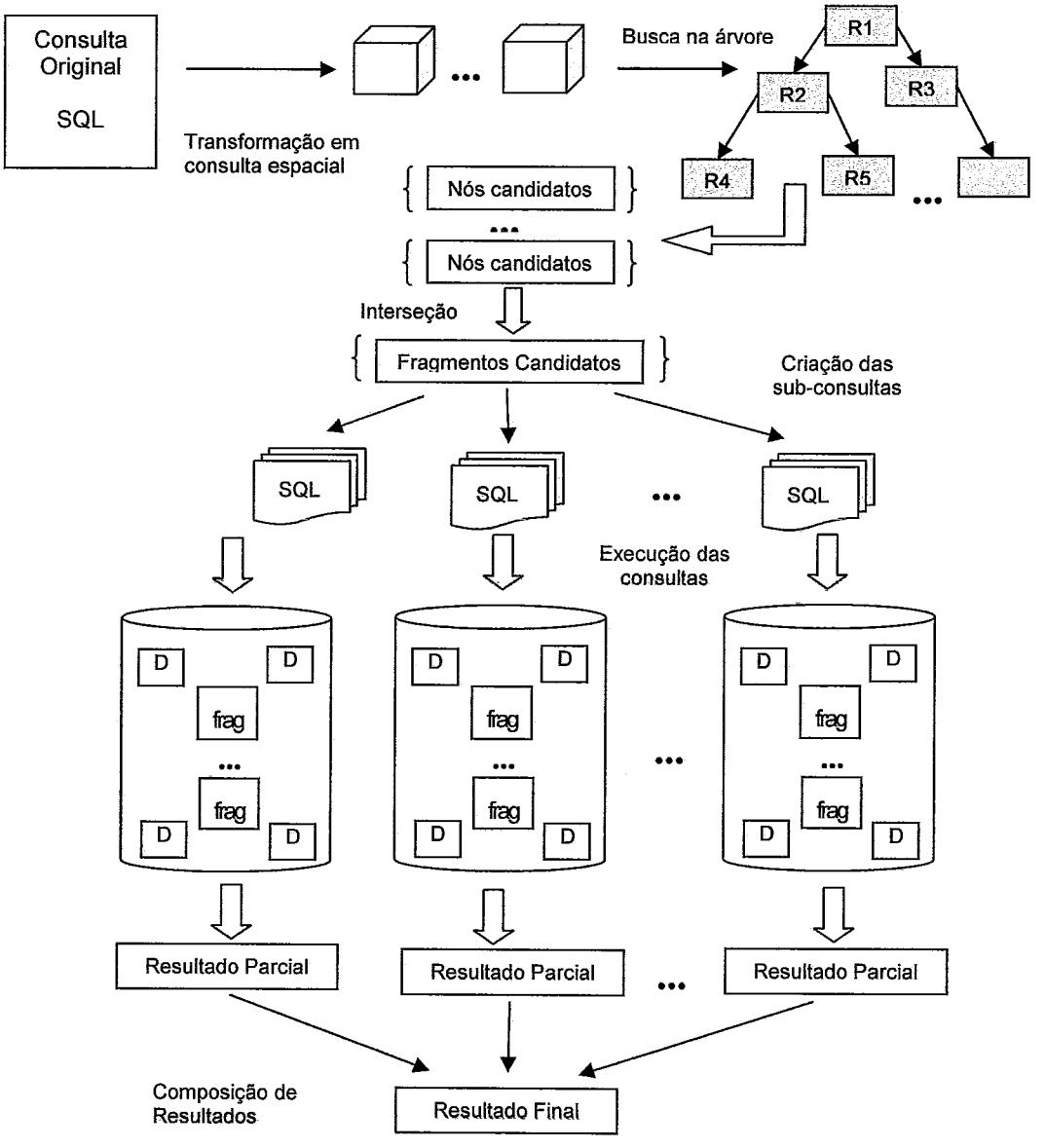


Figura 12 – Processamento de consultas

4. Validação Experimental

Neste capítulo apresentamos os detalhes da implementação do protótipo que desenvolvemos para avaliar através de experimentos a proposta da nossa dissertação. Além disso, também apresentamos e analisamos os resultados dos experimentos executados. Na seção 4.1 falamos sobre todas as configurações nas quais nossos experimentos foram executados. Em seguida, na seção 4.2 descrevemos sobre o benchmark TPC-H que utilizamos como nossa base de dados nos experimentos. Na seção 4.3 detalhamos a arquitetura do protótipo. Na seção 4.4 fazemos uma análise das consultas do TPC-H selecionadas para a execução dos experimentos e finalmente na seção 4.5 apresentamos os resultados obtidos experimentalmente.

4.1. Configuração Experimental

Para avaliar o desempenho da nossa solução, desenvolvemos um protótipo sobre a base de dados definida pelo *benchmark* TPC-H em um ambiente distribuído. A escolha do *benchmark* TPC-H foi motivada por ele oferecer uma representação de um ambiente típico de *data warehouse* além de apresentar um conjunto de consultas OLAP orientadas a suporte a decisão.

Os experimentos foram realizados em um laboratório com 8 PCs, cada um correspondendo a um nó do ambiente distribuído. Cada nó possui um processador Intel Pentium 4 HT 2.8 GHz e 512 Mb de memória RAM. Os nós estão interconectados por uma rede local de alta velocidade Ethernet de 1 Gigabit e possuem o sistema operacional Windows XP, cada um rodando uma instância do banco de dados livre PostgresSQL 8.2.4.

Nosso protótipo foi implementado em C++. A comunicação entre o banco de dados e a aplicação é feita através da biblioteca *libpq*, distribuída junto com o PostgreSQL. Essa biblioteca oferece um conjunto de funções que permite que aplicações clientes enviem consultas ao servidor de banco de dados PostgreSQL e receba o resultado dessas consultas (POSTGRESQL, 2006).

As máquinas possuem processadores com a tecnologia de *hyper-threading*, que simula a existência de dois processadores por nó, permitindo então que sejam executados dois processos simultaneamente. Para aproveitar essa característica dos

processadores durante o processamento das consultas, sempre que possível, disparamos duas *threads* para cada nó.

Alguns experimentos foram realizados com diferentes números de nós utilizados para o processamento das consultas. Para cada configuração, cada consulta foi executada 4 vezes, para minimizar a influência de carga de processos próprio do sistema operacional. Além disso, na maior parte das análises descartamos o resultado da primeira execução para considerar a influência da utilização do *cache* pelo SGBD. As avaliações são feitas calculando o tempo médio das 3 execuções de cada consulta.

4.2. TPC-H

O "*Transaction Processing Performance Council*" (TPC) é uma organização sem fins lucrativos fundada com objetivo de estabelecer *benchmarks* de processamento de transações e de banco de dados e publicar resultados confiáveis de desempenho para o mercado (TPC, 2005c). Os *benchmarks* produzidos pelo TPC avaliam o desempenho de processamento de transações e de banco de dados medindo o número de transações que determinado sistema de bancos de dados é capaz de processar por unidade de tempo.

O TPC estabelece *benchmarks* para aplicações OLTP (*On-line Transaction Processing*), aplicações OLAP (*On-line Analytical Processing*) e para aplicações de servidores. O TPC-H é um *benchmark* de suporte a decisões que consiste em um conjunto de consultas *ad-hoc* voltadas para os negócios e modificações de dados simultâneas. Ele ilustra sistemas de suporte à decisão cujo principal objetivo é fornecer suporte a questões críticas de negócios Esses sistemas acessam grandes volumes de dados e suas consultas possuem um alto nível de complexidade.

O esquema da base de dados do TPC-H é formado por oito tabelas: region, nation, supplier, part, partsupp, customer, que são as tabelas dimensão e orders e lineitem, tabelas fato. A cardinalidade das tabelas depende do fator de escala escolhido no momento da geração da base de dados, com exceção das tabelas region e nation que possuem cinco e vinte cinco tuplas respectivamente, independente do fator de escala escolhido. A cardinalidade das outras tabelas é definida pelo fator de escala da seguinte forma, sendo FS o fator de escala em questão: |Supplier| = FS * 10.000, |Customer| = FS * 150.000, |Part| = FS * 2000.000, |PartSupp| = FS * 800.000 e |Lineitem| = FS*

6.000.000. Na verdade, a cardinalidade de lineitem não é um múltiplo exato do fator de escala, pois para cada tupla de orders são gerados em média 4 tuplas em lineitem.

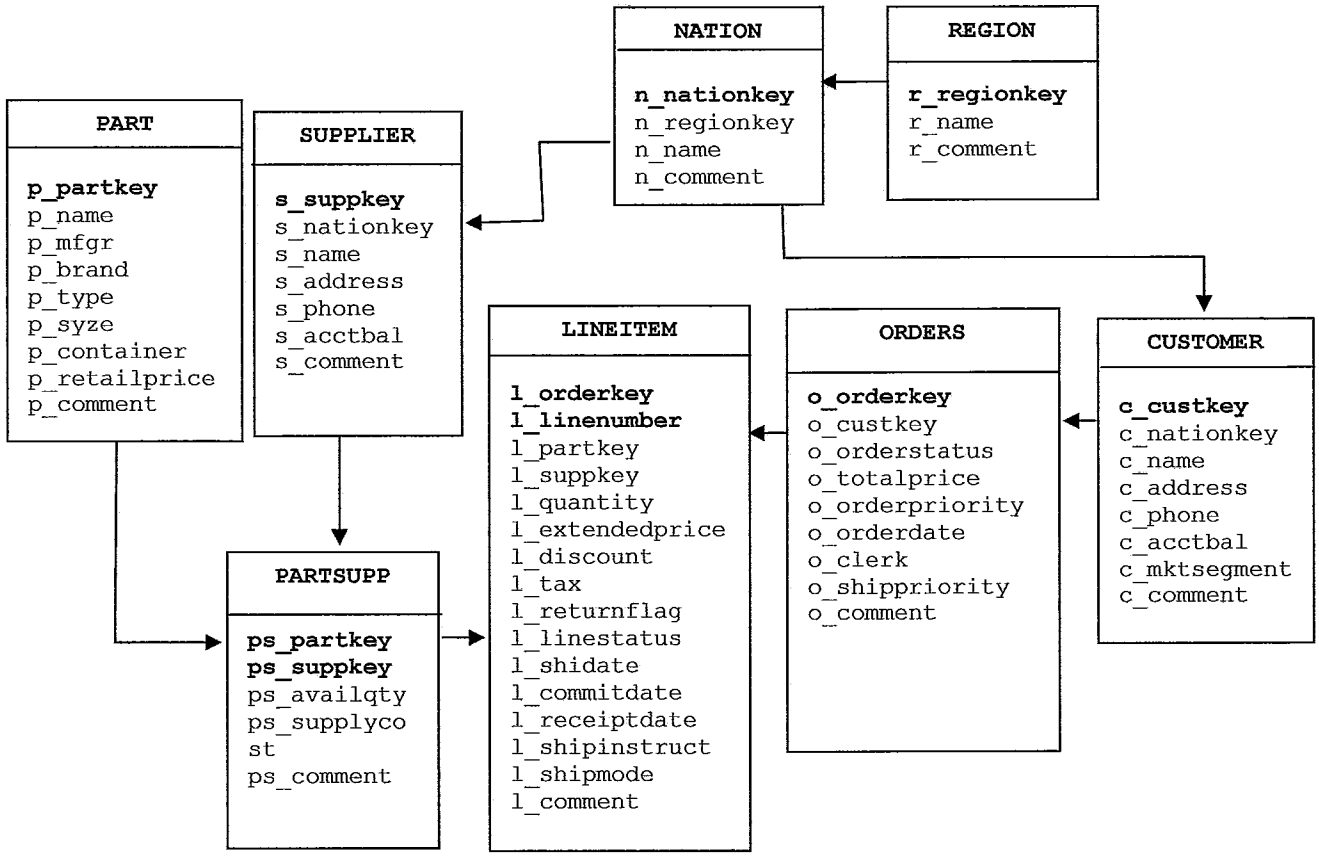


Figura 13 – Esquema Estrela do TPC-H

4.3. Arquitetura do Protótipo

A arquitetura do nosso protótipo ilustrada na figura 14 é composta por um nó principal que armazena a árvore R e é responsável por coordenar a fragmentação da tabela, a alocação dos fragmentos, a execução das sub-consultas na base fragmentada e posteriormente fazer a composição dos resultados. Os nós restantes executam uma instância do PostgreSQL que armazenarão os fragmentos. Cada Fi (1 <= i <= 6) representa um fragmento de uma tabela fato F. As dimensões são replicadas em todos os nós e os fragmentos foram distribuídos utilizando o algoritmo de escalonamento *round-robin.*

O nó principal possui um catálogo que armazena em que nó cada fragmento está alocado. Ao receber uma consulta que acessa a tabela F, o nó principal verifica através dos predicados de seleção da consulta e da árvore R, quais fragmentos possuem dados

relevantes para a consulta. Com a lista de fragmentos candidatos, é feita então uma consulta no catálogo para direcionar as sub-consultas para os nós correspondentes. No final do processamento, cada nó envia os resultados parciais de volta para o nó principal, que por sua vez os consolida.

Por motivos de simplicidade, decidimos restringir nossa implementação para dar suporte somente para consultas conjuntivas, ou seja, consultas onde os predicados de junção e seleção são combinados em expressões conjuntivas. Além disso, a composição de resultados não é automática, ou seja, o protótipo não verifica automaticamente na consulta como a composição de resultados deve ser feita (através de uma soma, totalização ou média). Também não implementamos um *parser*, que seria responsável por identificar nas consultas os predicados de seleção feitos sobre atributos de fragmentação para recuperar automaticamente o intervalo de dados correspondentes aos predicados. O escopo do nosso protótipo é avaliar o desempenho da proposta de fragmentação multidimensional, por isso deixamos a implementação do *parser* e da composição de resultados automática como trabalhos futuros.

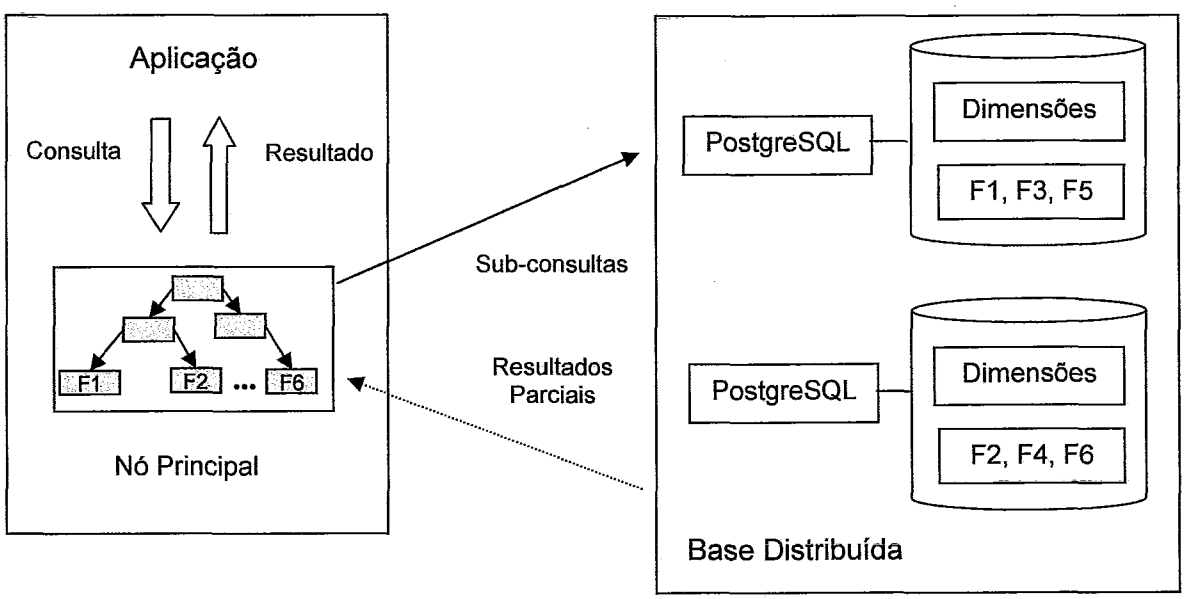


Figura 14 – Arquitetura do Protótipo

Decidimos fragmentar a tabela fato lineitem por ser a maior tabela do esquema do *benchmark* TPC-H. As outras tabelas foram replicadas por todos os nós. A base de dados foi gerada com fator de escala 1, o que resulta em aproximadamente 6 milhões de tuplas para a tabela lineitem. Utilizamos os campos que são chaves estrangeiras para

outras tabelas do esquema para compor o ponto multidimensional que será inserido na árvore R. Foram selecionados os campos l_suppkey chave estrangeira para tabela supplier, l_orderkey chave estrangeira para a tabela orders e finalmente l_partkey, chave estrangeira para tabela part. Logo, no nosso protótipo, a árvore R indexa pontos de três dimensões.

O tamanho dos fragmentos é determinado pelo tamanho da página dos nós da árvore R, que é um dos parâmetros a ser definido na criação da árvore. O tamanho de página escolhido define o número máximo de entradas que cada nó folha pode armazenar e consequentemente define o número máximo de tuplas de cada fragmento. Além disso, quanto menor o tamanho dos fragmentos, mais fragmentos serão gerados e da mesma forma, quanto maior o tamanho dos fragmentos, menor será o número de fragmentos gerados.

Para calcular o número máximo de entradas de um nó, é preciso analisar a sua estrutura:

Estrutura do nó da árvore R		
Folha (4 bytes)	Box (24 bytes)	Lista de Entradas (cada entrada possui 28 bytes)

Onde,

- Folha – atributo que indica se o nó é folha ou interno. Atributo do tipo inteiro.
- Box – armazena o menor hiper-retângulo envolvente do nó. O hiper-retângulo é representado através de dois pontos, um que indica sua coordenada máxima e outro que indica sua coordenada mínima. Como armazenamos pontos de 3 dimensões, cada ponto é representado através de um vetor de 3 posições do tipo inteiro. Logo, a estrutura Box ocupa 24 bytes.
- Vetor de entradas – Se o nó é folha, esse atributo armazena os objetos propriamente ditos. Cada entrada é uma estrutura composta por um atributo do tipo Box que armazena o ponto, e um atributo do tipo inteiro que armazena uma posição no arquivo da árvore em disco. Logo cada entrada ocupa 28 bytes de armazenamento.

Com essas informações, a árvore calcula o número máximo de entradas que o nó pode armazenar através da seguinte fórmula:

$$\text{número máximo de entradas} = \frac{(\text{tamanho do nó} - \text{tamanho folha} - \text{tamanho box})}{\text{tamanho da entrada}}$$

Por sua vez, o tamanho mínimo de cada nó é definido como 40% do tamanho máximo. Segundo experimentos realizados em (BECKMANN, KRIEGEL et al., 1990), o número mínimo de entradas definido como 40% do número máximo de entradas atingiu melhor desempenho de busca na árvore.

Nos experimentos que realizamos, geramos árvores com diferentes tamanhos de página para avaliar o desempenho das consultas com tamanhos variados de fragmentos. A tabela 1 lista os tamanhos de página que utilizamos, quantos fragmentos foram gerados com essa configuração e o número mínimo e máximo de tuplas de cada fragmento, que corresponde ao número mínimo e máximo de entradas dos nós da árvore.

Tamanho Página	# Fragmentos	# Min Tuplas	# Max Tuplas
128 Kbytes	2434	1871	4679
256 Kbytes	1228	3744	9360
512 Kbytes	613	7489	18723

Tabela 1 – Configuração com diferentes tamanhos de páginas

Para aumentar melhorar a probabilidade de que as consultas se restrinjam a um grupo pequeno de fragmentos, geramos valores seqüenciais para os campos que compõem o ponto multidimensional da árvore, ordenados sobre um dos atributos da tabela das quais esses campos são chave, ou seja, geramos novos valores para os campos s_suppkey, p_partkey e o_orderkey que são chaves primárias das tabelas supplier, part e orders respectivamente, ordenados por um dos atributos da sua tabela respectiva. A escolha do atributo utilizado como critério de ordenação foi feito a partir da análise do conjunto de consultas a serem executadas. Foram escolhidos os atributos que eram mais utilizados em predicados de seleção nas consultas. Essa escolha não interfere nas consultas onde há restrições pontuais sobre as dimensões, e quando houver consultas com restrições de faixas de valores sobre os atributos das dimensões utilizados para a ordenação das chaves teremos também faixas de valores nestas, otimizando o uso da fragmentação adotada. Na próxima seção apresentamos o conjunto

de consultas escolhidas para os experimentos e a análise feita para escolha dos atributos de ordenação.

4.4. Consultas do TPC-H

Em nossos experimentos, nós selecionamos as seguintes consultas dentre as oferecidas pelo TPC-H: Q3, Q5, Q6, Q8, Q9 e Q10, com diferentes características. Elas são listadas abaixo:

Consulta Q3:

```
select l_orderkey,sum(l_extendedprice*(1-l_discount)) as revenue
       o_orderdate, o_shippriority,
  from customer, orders, lineitem
 where c_mktsegment = 'BUILDING'
   and c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and o_orderdate < date '1995-03-15'
   and l_shipdate > date '1995-03-15'
 group by l_orderkey, o_orderdate, o_shippriority
```

Consulta Q5:

```
select n_name,    sum(l_extendedprice * (1 - l_discount)) as revenue
  from customer, orders, supplier, nation, region, lineitem
 where c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and l_suppkey = s_suppkey
   and c_nationkey = s_nationkey
   and s_nationkey = n_nationkey
   and n_regionkey = r_regionkey
   and r_name = 'ASIA'
   and o_orderdate >= date '1994-01-01'
   and o_orderdate < date '1994-01-01' + interval '1 year'
 group by n_name
```

Consulta Q6:

```
select sum(l_extendedprice*l_discount) as revenue
  from lineitem
 where l_shipdate >= date '1994-01-01'
   and l_shipdate < date '1994-01-01' + interval '1 year'
   and l_discount between 0.06 - 0.01 and 0.06 + 0.01
   and l_quantity < 24
```

Consulta Q8:

```
select o_year,
       sum(case when nation = 'BRAZIL' then volume
           else 0 end) / sum(volume) as mkt_share
  from (select extract(year from o_orderdate) as o_year,
```

```
                l_extendedprice * (1-l_discount) as volume,
                 n2.n_name as nation
            from part, supplier, lineitem, orders, customer,
                 nation n1, nation n2, region
           where p_partkey = l_partkey
             and s_suppkey = l_suppkey
             and l_orderkey = o_orderkey
             and o_custkey = c_custkey
             and c_nationkey = n1.n_nationkey
             and n1.n_regionkey = r_regionkey
             and r_name = 'AMERICA'
             and s_nationkey = n2.n_nationkey
             and o_orderdate between date '1995-01-01' and date '1996-
12-31'
              and p_type = 'ECONOMY ANODIZED STEEL') as all_nations
group by o_year
```

Consulta Q9:

```
select nation, o_year, sum(amount) as sum_profit
  from ( select n_name as nation,
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) - ps_supplycost *
                l_quantity as amount
           from part, supplier, lineitem, partsupp, orders, nation
          where s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%green%') as profit
  group by nation, o_year
```

Consulta Q10:

```
select c_custkey, c_name,
       sum(l_extendedprice * (1 - l_discount)) as revenue,
       c_acctbal, n_name, c_address, c_phone,  c_comment
 from customer, orders, nation, lineitem
where c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate >= date '1993-10-01'
  and o_orderdate < date '1993-10-01' + interval '3 month'
  and l_returnflag = 'R'
  and c_nationkey = n_nationkey
group by c_custkey, c_name, c_acctbal,
         c_phone, n_name, c_address,c_comment
```

Todas as consultas selecionadas envolvem a tabela fato lineitem. Além disso, para avaliar nossa proposta com diferentes cenários, escolhemos consultas que possuíam predicados de seleção sobre os atributos utilizados para fragmentar a tabela lineitem assim como consultas que não são restringidas por predicados de seleção, ou seja, que precisam acessar todos os fragmentos da base.

A consulta Q3 faz junção entre as tabelas fatos lineitem e orders e uma tabela dimensão. Ela realiza uma operação de agregação e, diferentemente das outras consultas, seu resultado retorna muitas tuplas. Além disso, possui um predicado de seleção sobre um dos atributos utilizados para a fragmentação da tabela fato. A consulta Q5 faz uma junção das tabelas fato e quatro dimensões: region, nation, supplier e customer. Ela também realiza apenas uma operação de agregação e possui dois predicados de seleção sobre atributos utilizados na fragmentação. A consulta Q6 acessa apenas a tabela lineitem. Ela possui uma operação de agregação e não possui nenhum predicado de seleção sobre os campos utilizados na fragmentação. Nesse caso todos os fragmentos devem ser acessados para atender a consulta. A consulta Q8 faz junção entre as duas tabelas fatos e cinco dimensões: part, supplier, customer, nation e region. Ela realiza duas operações de agregação e possui três predicados de seleção sobre atributos de fragmentação. A consulta Q9 também faz junção entre as duas tabelas fato e quatro tabelas dimensão: part, supplier, partsupp e nation, e assim como Q8, realiza duas operações de agregação e possui somente um predicado de seleção sobre o atributo de fragmentação. Finalmente a consulta Q10 também faz junção entre as tabelas fato e duas tabelas dimensão (customer e nation). Ela realiza somente uma operação de agregação e um predicado de seleção sobre um dos atributos de fragmentação.

Analisando os predicados de seleção das consultas, observamos que as consultas Q3, Q5, Q8 e Q10 possuem um predicado de seleção sobre o campo o_orderdate da tabela orders. As consultas Q5 e Q8 possuem um predicado de seleção sobre o campo r_name da tabela region. A consulta Q3 realiza uma seleção sobre o campo c_mktsegment da tabela customer. As consultas Q8 e Q9 realizam seleção sobre dois campos distintos da tabela part: p_type e p_name.

Consulta Q3:

```
and c_mktsegment = 'BUILDING'
and o_orderdate < date '1995-03-15'
```

Consulta Q5:

```
and r_name = 'ASIA'
and o_orderdate >= date '1994-01-01'
and o_orderdate < date '1994-01-01' + interval '1 year'
```

Consulta Q8:

```
and r_name = 'AMERICA'
and o_orderdate between date '1995-01-01' and date '1996-12-31'
and p_type = 'ECONOMY ANODIZED STEEL'
```

Consulta Q9:

```
and p_name like '%green%'
```

Q10:

```
and o_orderdate >= date '1993-10-01'
and o_orderdate < date '1993-10-01' + interval '3 month'
```

A partir dessa análise selecionamos os campos das tabelas dimensão que serão utilizados como critério de ordenação na geração de novos identificadores dessas tabelas. Para a tabela order, selecionamos o campo o_orderdate, utilizado como predicado de seleção em quatro consultas. Para a tabela region, escolhemos o campo r_name, utilizado em duas consultas. Para a tabela customer, foi eleito o campo c_mktsegment, que faz parte do predicado de seleção de uma das consultas. Finalmente para a tabela part, tínhamos duas opções: o campo p_type ou o campo p_name. Decidimos escolher o campo p_type, pois o predicado de seleção sobre o campo p_name envolve um operador LIKE, que no caso não representaria ganho considerável para a consulta correspondente ao utilizá-lo como critério de ordenação.

Para descobrir os fragmentos a serem acessados por certa consulta, é preciso descobrir o intervalo de dados limitado pelos predicados de seleção da consulta ao longo de uma coordenada. Para isso, executamos uma consulta na tabela acessada pelo predicado e selecionamos o valor mínimo e o valor máximo do identificador da tabela. Com esses valores, calculamos o hiper-retângulo de busca que é passado como parâmetro para o procedimento de busca da árvore. O procedimento de busca retorna então a lista de nós que possuem interseção com o hiper-retângulo de busca. Para cada consulta do TPC-H selecionada para os experimentos, listamos abaixo as consultas efetuadas para calcular seus hiper-retângulos de busca correspondentes. As consultas que possuem predicados de seleção sobre diferentes tabelas possuem mais de um hiper-retângulo de busca, um para cada predicado. Nesses casos, a lista final de fragmentos candidatos é calculada a partir da interseção da lista de fragmentos indicada por cada hiper-retângulo de busca.

Consulta Q3:

```
select o_orderkey from orders, customer
 where c_custkey = o_custkey
  and c_mktsegment = 'BUILDING'
  and o_orderdate < date '1995-03-15' order by o_orderkey
```

Consulta Q5:

```
select s_suppkey from supplier, region, nation
where r_name = 'ASIA'
  and n_regionkey = r_regionkey
  and n_nationkey = s_nationkey order by s_suppkey


select o_orderkey from orders
 where o_orderdate >= date '1994-01-01'
   and o_orderdate < date '1994-01-01' + interval '1 year'
  order by o_orderkey
```

Consulta Q8:

```
select o_orderkey from orders, customer, region, nation
 where o_orderdate between date '1995-01-01' and date '1996-12-31'
   and o_custkey = c_custkey
   and r_regionkey = n_regionkey
   and c_nationkey = n_nationkey
   and r_name = 'AMERICA' order by o_orderkey


select p_partkey from part
where p_type = 'ECONOMY ANODIZED STEEL' order by p_partkey
```

Consulta Q9:

```
select p_partkey from part
 where p_name like '%green%' order by p_partkey
```

Consulta Q10:

```
select o_orderkey from orders
 where o_orderdate >= date '1993-10-01'
   and o_orderdate < date '1993-10-01' + interval '3 month'
order by o_orderkey
```

4.5. Experimentos

Realizamos dois tipos de experimentos: primeiramente comparamos o tempo de execução das consultas na base original não fragmentada com o tempo de execução das consultas em um ambiente monoprocessado com diferentes configurações de fragmentação, onde variamos o tamanho e a quantidade dos fragmentos. Em seguida, fazemos testes em um ambiente distribuído utilizando paralelismo intra-consulta com um número crescente de nós da rede distribuída (2, 4 e 8 nós). Todas as consultas foram executadas individualmente e isoladamente. O tempo de execução das consultas na base fragmentada leva em consideração além do tempo para processar todas as sub-

consultas, o tempo gasto para calcular os hiper-retângulos de busca, efetuar a busca na árvore e consolidar os resultados.

Os experimentos avaliam além do tempo de execução das consultas, o número de fragmentos que foram necessários acessar para a execução de cada consulta. Assim podemos avaliar o quanto nossa proposta de fragmentação restringe o número de fragmentos a serem acessados.

Consulta	Página de 128 Kbytes		Página de 256 Kbytes		Página de 512 Kbytes	
	# Fragmentos	%	# Fragmentos	%	# Fragmentos	%
Q3	1343	55,18	680	55,37	328	53,51
Q5	151	6,20	74	6,03	37	6,04
Q6	2434	100	1228	100	613	100
Q8	66	2,71	49	3,99	32	5,22
Q9	2434	100	1228	100	613	100
Q10	420	17,26	211	17,18	107	17,46

Tabela 2 – Total de Fragmentos acessados por consulta e tamanho dos fragmentos.

Primeiramente vamos analisar a quantidade de fragmentos necessários para executar cada consulta. A tabela 2 lista o total de fragmentos acessados por consulta e quanto representa percentualmente sobre o total de fragmentos da base. Para as consultas Q3, Q5, Q8 e Q10 conseguimos restringir o número de fragmentos acessados. Todas elas possuem pelo menos um predicado de seleção sobre dimensões que participaram da fragmentação da tabela fato e que através de suas restrições, possibilitaram limitar o número de fragmentos a serem acessados. Analisando a fragmentação com 256 Kbytes como exemplo, verificamos que a consulta Q8 foi a que acessou menos fragmentos (3,99%), seguida das consultas Q5 (6,03%), Q10 (17,18%) e Q3 (55,37%). Esses resultados mostram que a proposta de fragmentação permite reduzir significativamente o volume de dados a ser acessado por consultas que possuam pelo menos um predicado de seleção sobre tabelas cujos identificadores foram mapeados em coordenadas multidimensionais.

A consulta Q6 precisa acessar 100% dos fragmentos. Isso já era esperado, pois ela não possui predicado de seleção sobre as dimensões que participaram da fragmentação. A consulta Q9 também precisa acessar 100% dos fragmentos, pois possui um predicado de seleção sobre a dimensão part mas o campo restringido no predicado de seleção não foi utilizado para ordenação do identificador da tabela part. Nesse caso,

os identificadores que atendem ao critério do predicado de seleção ficaram muito espalhados entre os fragmentos da base.

A tabela 3 lista o tempo médio de execução em segundos para cada consulta variando o tamanho da página dos nós da árvore. A primeira linha da tabela corresponde ao tempo de execução das consultas na base original, sem nenhuma fragmentação. As consultas foram executadas em um ambiente monoprocessado, ou seja, não utilizamos paralelismo. O objetivo desses experimentos é avaliar a aceleração obtida com a restrição do volume de dados a ser acessado pelas consultas.

Frag. \ Consulta	Q3	Q5	Q6	Q8	Q9	Q10
Sem fragmentação	128,068	145,380	1,786	81,118	654,536	51,22
Pág. 128 Kb	287,984	12,964	427,354	3,245	1552,507	111,04
Pág. 256 Kb	309,516	11,406	386,125	3,245	1385,813	119,78
Pág. 512 Kb	224,401	8,839	249,297	2,958	1127,760	37,90

Tabela 3 – Tempo de execução das consultas em variando as configurações de fragmentação (em segundos).

Analisando os resultados apresentados pela tabela 3 e ilustrados pela figura 15, verificamos que obtivemos ganho com as consultas Q5 e Q8 somente com a fragmentação da base. Para tamanho de página de 256 Kbytes, a execução da consulta Q5 foi reduzida em cerca de 92% em relação ao tempo de execução na base não fragmentada. A consulta Q8 obteve redução do tempo de execução de cerca de 96%. A fragmentação física beneficiou as duas consultas pela redução do volume de dados acessado por elas, conforme mostramos na tabela 1.

Observamos também que a fragmentação com árvore com tamanho de página de 512 Kbytes obteve resultados melhores comparada com os outros tamanhos de página. Com esse tamanho de página, obtivemos ganho também para a consulta Q10, com redução de 26% do tempo de execução. A configuração com 128 Kbytes gerou uma quantidade maior de fragmentos (no total 2434), que aumentou a sobrecarga de processamento das consultas Q3, Q6, Q9 e Q10. A consulta Q10 em especial retorna no seu resultado grande quantidade de linhas e colunas, o que gerou um gargalo na composição de resultados. As consultas Q3, Q6 e Q9 são prejudicadas pela quantidade de fragmentos que precisam processar em um único computador, que fica sobrecarregado com o controle da execução das sub-consultas e com o processamento

dos resultados parciais. O processamento distribuído é indicado para esses casos, pois aumentamos o poder de processamento de CPU.

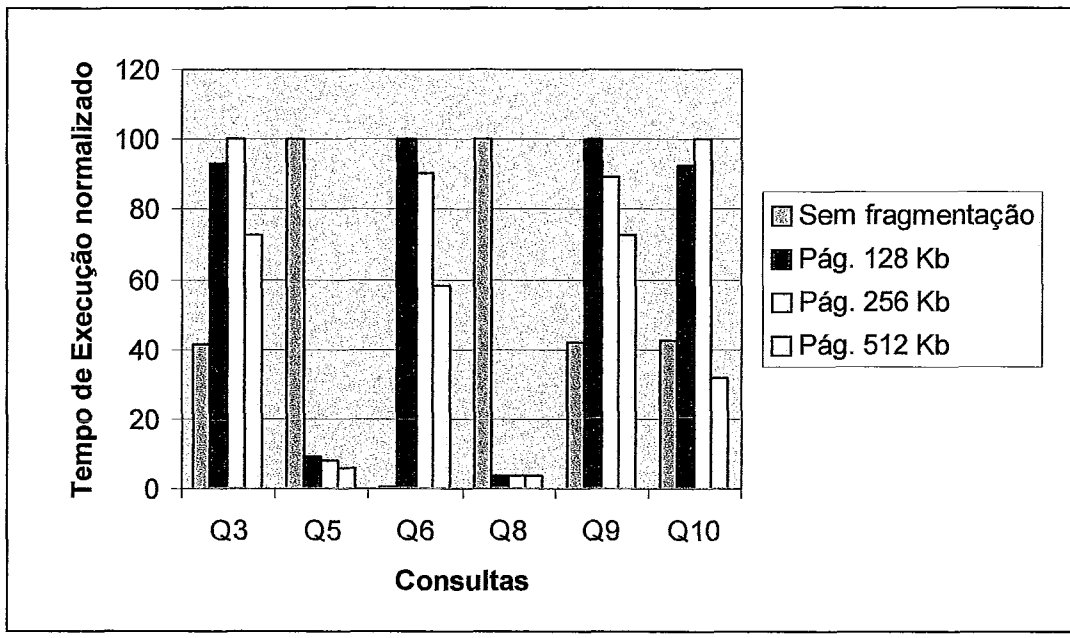


Figura 15 - Tempo de execução das consultas por tipo de fragmentação

Nos experimentos com a base distribuída, selecionamos a fragmentação com configuração de tamanho de página de 256 Kbytes. A tabela 4 apresenta o tempo médio de execução em segundos para cada consulta variando os números de nós do ambiente distribuído. A primeira linha da tabela lista o tempo de execução das consultas na base inteira, sem nenhuma fragmentação. As outras linhas mostram os resultados aumentando o número de nós.

#Nós \ Consulta	Q3	Q5	Q6	Q8	Q9	Q10
1	128,068	145,380	1,786	81,118	654,536	51,224
2	169,031	6,646	87,297	2,739	1083,724	37,474
4	77,917	4,302	1,911	2,141	817,994	43,797
8	45,068	4,896	1,156	2,036	425,734	49,073

Tabela 4 – Tempo de execução das consultas no ambiente distribuído (em segundos).

A figura 16 mostra o tempo de execução normalizado das consultas variando a configuração de nós.

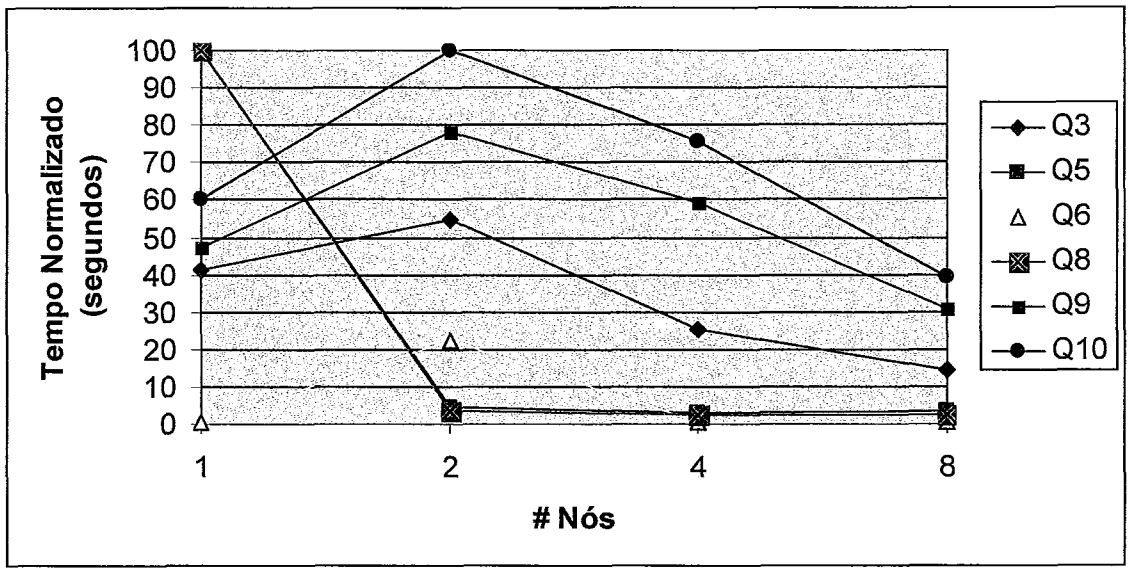


Figura 16 – Tempo de execução normalizado

Com o aumento do número de nós e a paralelização da execução das sub-consultas, o tempo de execução das consultas Q5 e Q8 continuou caindo, mas em uma proporção menor. A consulta Q10 começa a obter ganho de desempenho em relação ao tempo de execução na base inteira já a partir de 2 nós. A consulta Q3 começa a obter ganho a partir de 4 nós. Com 8 nós todas as consultas conseguem reduzir seu tempo de execução comparado ao tempo de execução na base inteira, como era de se esperar em consultas mais demoradas, onde o tempo extra gasto na transferência dos dados ainda é compensado pelo paralelismo.

Nas figuras 17 e 18 podemos comparar os planos de execução gerados pelo SGBD para a consulta Q3 na base original (sem fragmentação) e para uma das sub-consultas de Q3 executada sobre a base fragmentada. Podemos observar que o SGBD gerou um plano de consulta mais eficiente para o fragmento, pois utilizou um laço aninhado indexado (*indexed nested-loop*) para processar a parte da sub-consulta relativa à tabela lineitem, e também não precisou realizar uma ordenação para agrupar o resultado. Vale lembrar que embora o *hashjoin* possa ser mais eficiente em algumas situações, nesse caso, como ele envolve a varredura de duas tabelas grandes (lineitem e orders) ele se torna menos eficiente que um laço aninhado indexado pois o índice percorrido é bem menor que a tabela inteira. O mesmo ocorreu para as consultas Q5, Q8, Q9 e Q10.

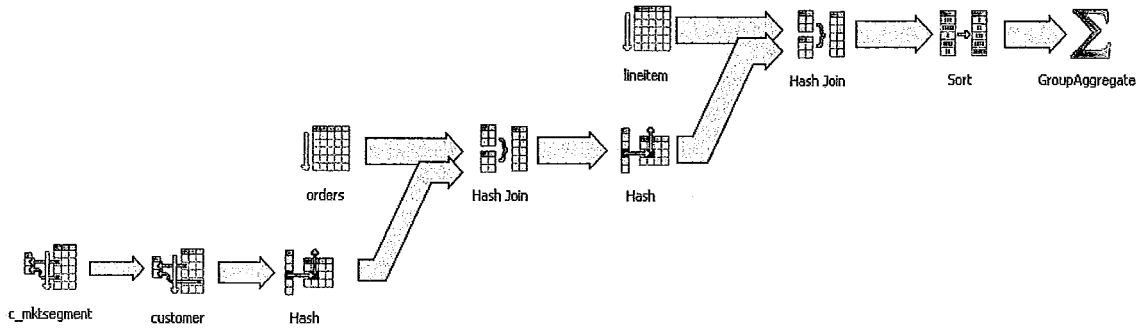


Figura 17 – Plano de Execução da consulta Q3 na base não fragmentada

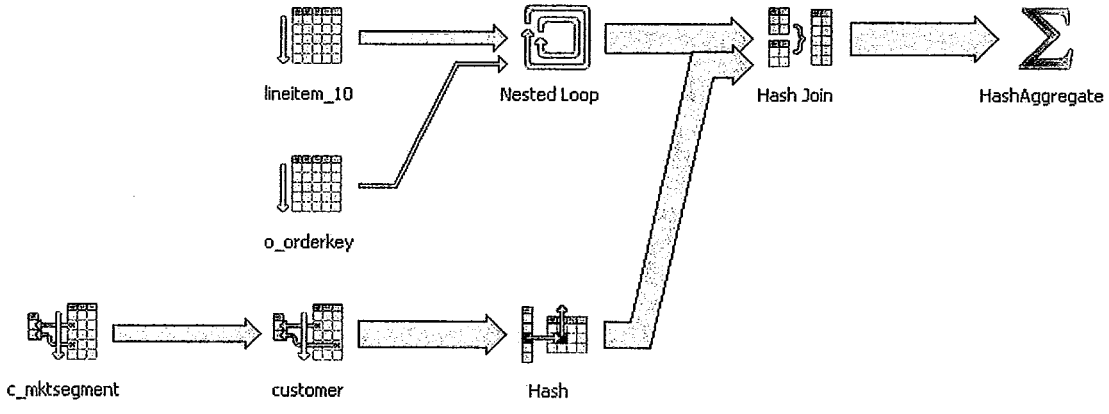


Figura 18 – Plano de Execução da consulta Q3 na base fragmentada

4.5.1. Efeitos do *cache*

Finalmente, mostraremos na tabela 3 o tempo de execução das consultas Q3, Q6 e Q9 com fragmentação com tamanho de página de 256 Kbytes considerando a primeira execução, ou seja, sem *cache* prévio.

	Q3			
	Rodada 1 (a frio)	**Rodada 2**	**Rodada 3**	**Rodada 4**
Base Inteira	99,45	168,24	107,19	108,78
1 nó	319,02	314,59	304,91	309,05
2 nós	128,94	178,09	204,52	124,48
4 nós	88,49	83,56	75,39	74,80
8 nós	127,58	60,94	32,08	42,19
	Q6			
	Rodada 1 (a frio)	**Rodada 2**	**Rodada 3**	**Rodada 4**
Base Inteira	32,45	1,94	1,70	1,72
1 nó	466,56	397,03	385,50	375,84
2 nós	175,66	84,28	86,58	91,03
4 nós	72,41	1,94	1,91	1,89
8 nós	45,66	1,34	1,05	1,08
	Q9			
	Rodada 1 (a frio)	**Rodada 2**	**Rodada 3**	**Rodada 4**
Base Inteira	688,19	559,70	780,88	623,03
1 nó	1637,67	1401,75	1351,30	1404,39
2 nós	1204,84	1149,88	960,91	1140,39
4 nós	841,61	816,81	820,61	816,56
8 nós	603,78	427,27	410,22	439,72

Tabela 5 – Tempo de execução das consultas Q3, Q6 e Q9

Conforme notamos, o tempo extra gasto para processar 1228 consultas não é desprezível, porém não explica o fato de que nas consultas Q3 e Q9 a segunda execução (que deveria se utilizar do *cache*) não foi substancialmente diferente da primeira. No entanto, a consulta Q6 (que acessa apenas a tabela lineitem) teve significativa redução na execução sobre a base não fragmentada. Concluímos que isto ocorreu devido à política de *cache* do PostgreSQL, que privilegia o *cache* de uma tabela inteira em detrimento de várias tabelas pequenas. Na consulta Q6, na base fragmentada, essa redução somente foi significativa quando o número de fragmentos por nó tornou-se pequeno – cerca de 300. Essa conclusão indica que algumas investigações a mais devem ser feitas para se encontrar o número ideal de fragmentos que seja compatível com a política de *cache* do PostgreSQL.

5. Conclusão

A principal contribuição deste trabalho é apontar que o uso de índices multidimensionais baseados em clusterização, dos quais a árvore R é um dos mais populares e com reconhecido desempenho, é uma alternativa de política de fragmentação a ser considerada no projeto de distribuição de dados de *data warehouses.*

Nossa proposta aproveita a característica dimensional da modelagem utilizada em projetos de *data warehouses* para fragmentar fisicamente a tabela fato de um esquema estrela utilizando a árvore R como estrutura de particionamento. As chaves estrangeiras da tabela fato são mapeadas em pontos multidimensionais que por sua vez são indexadas na árvore R. No final, cada nó folha da árvore corresponde a um fragmento físico da tabela fato. Além de definir o particionamento do espaço, a árvore R é também a estrutura utilizada para direcionar as consultas para os fragmentos relevantes a elas.

A clusterização propiciada pela árvore R possibilita limitar o volume de dados a ser acessado por consultas que acessam a tabela fato e restringem seu espaço de busca através de predicados de seleção sobre as dimensões. Aliada à fragmentação física, o processamento distribuído dos fragmentos através da paralelização intra-consulta permite acelerar individualmente a execução das consultas OLAP.

Para avaliar o desempenho da fragmentação física utilizando a árvore R, implementamos um protótipo e realizamos experimentos com um conjunto de consultas de leitura do *benchmark* TPC-H, em um ambiente distribuído de 8 nós, cada um rodando uma instância do SGBD PostgreSQL.

Os resultados dos experimentos mostraram que embora algumas consultas tenham se tornado mais lentas utilizando o esquema proposto de fragmentação quando utilizamos apenas um nó, o ganho obtido nas outras consultas foi realmente encorajador. Identificamos assim que a fragmentação em apenas uma tabela pode não ser o ideal – para beneficiar um número maior de consultas propomos, como trabalhos futuros, investigar o efeito de fragmentações derivadas nas outras tabelas, inclusive nas tabelas dimensão. Outra continuação sugerida para este trabalho é investigar o uso da própria árvore R para o armazenamento dos dados, ou seja, a criação de um tipo de tabela ordenada (como existem em diversos SGBDs), porém onde a ordenação é guiada pela clusterização fornecida pela árvore R.

Outro fator decisivo para o bom desempenho do processamento das consultas sobre a base fragmentada é determinar a granularidade ideal da fragmentação, ou seja, definir o melhor tamanho dos fragmentos. Nos experimentos realizados com diferentes configurações de tamanhos de fragmentos, observamos que uma quantidade excessiva de fragmentos obteve resultados inferiores quando comparada às configurações com fragmentos maiores. Como trabalhos futuros, sugerimos realizar mais experimentos com fragmentos maiores e estudar uma maneira de calcular o tamanho ideal dos fragmentos físicos.

Uma questão não abordada nessa dissertação é em relação à disponibilidade dos dados. Um esquema de distribuição dos fragmentos que considere a replicação dos fragmentos entre os nós da rede além de aumentar a disponibilidade dos dados, possibilita também que seja realizado balanceamento de carga entre os nós da rede. A base de dados que utilizamos nos experimentos não apresenta distorção de dados, por isso propomos também que futuramente sejam realizados experimentos com bases não uniformes.

Vale ressaltar que os experimentos obtidos foram executados somente com consultas de leitura. Nessa dissertação não abordamos o problema da atualização dos dados. Entretanto, a atualização do banco com os novos dados consolidados do DW pode ser feita através da inserção na própria árvore R, que é uma estrutura dinâmica e adaptativa aos dados. Essa inserção pode acarretar a criação de novos fragmentos. Deixamos para trabalhos futuros o tratamento de atualização dos dados utilizando a árvore R. Por fim, sugerimos a incorporação dos algoritmos propostos para fragmentação multidimensional ao ParGRES.

Referências Bibliográficas

AKAL, F., BÖHM, K., SCHEK, H.-J., 2002, "OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism", In: *Proceedings of the 6th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, pp. 218-231, Bratislava, Slovakia, Sep

BECKMANN, N., KRIEGEL, H., SCHNEIDER, R., et al., 1990, "The R* Tree: An Efficient and Robust Access Method for Points and Rectangles", In: *Proceedings of International Conference on Management of Data (ACM SIGMOD)*, pp. 322-331, Atlantic City, New Jersey, USA

BERCHTOLD, S., BÖHM, C., KRIEGEL, H.-P., 1998, "The Pyramid Technique: Towards Breaking the curse of Dimensionality", In: *Proceedings of International Conference on Management of Data (ACM SIGMOD)*, pp. 142-153, Seattle, Washington, USA

BERCHTOLD, S., KEIM, D., KRIEGEL, H.-P., 1996, "The X-tree: An Index Structure for high Dimensional Data", In: *Proceedings of the 22nd VLDB Conference*, pp. 28-38, Bombay, India

BIALEK, B. A. R., 2003, "IBM DB2 Integrated Cluster Environment for Linux", url: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

BRAKATSOULAS, S., PFOSER D., THEODORIDIS Y., 2002, "Revisiting R-Tree Construction Principles", In: *Proceedings of 6th Advances in Databases and Information Systems Conference (ADBIS)*, pp. 149-162, Bratislava, Slovakia

CECCHET, E., MARGUERITE, J., ZWAENEPOEL, W., 2004, "C-JDBC: Flexible Database Clustering Middleware", In: *Proceedings of USENIX Annual Technical Conference*Freenix track

DEWITT, D., GRAY, J., 1992, "Parallel database systems: the future of high performance database systems", In: *Communications of the ACM*, pp. 85-98, June

FURTADO C., 2006, "Physical and Virtual Partitioning for OLAP Query Processing in a Database Cluster", M. Sc. Thesis, Computer Science Department, COPPE, Federal University of Rio de Janeiro.

FURTADO, C., LIMA, A. A. B., PACITTI, E., VALDURIEZ, P., MATTOSO,M., 2006, "Adaptive Hybrid Partitioning for OLAP Query Processing in a Database Cluster", In: *International Journal of High Performance Computing and Networking, v.Prelo*

GARCIA-MOLINA, H., ULLMAN, J., WIDOM D., 2001, "Implementação de Sistemas de Bancos de Dados", In: *Editora Campus*

GUTTMAN, A., 1984, "R-Trees: A Dynamic Index Structure for Spatial Searching", In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 47-57, Boston, Massachusetts, USA

HUANG, Z., 1998, "Extensions to the k-Means algorithm for clustering large datasets with categorical values", In: *Data Mining and Knowledge Discovery*, pp. 283-304,

INMON, W. H., 2005, "Building the Data Warehouse, Fourth Edition", In: *Wiley Computer Publishing*

KIMBALL, R., ROSS, M., 2002, "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2 ed.", In: *Wiley Computer Publishing*

LIMA, A. A. B., 2004, "Intra-query Parallelism in Database Clusters", Ph. D. Thesis, Computer Science Department, COPPE, Federal University of Rio de Janeiro, Brazil, Dec.

LIMA, A. A. B., MATTOSO, M., VALDURIEZ P, 2004, "Adaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster", In: *Proceedings of the 19th Brazilian Symposium on Databases (SBBD)*, pp. 92-105, Brasília, Brazil, Oct.

MACQUEEN, J. B., 1967, "Some methods for classification and analysis of multivariate observations", In: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281-297, Berkeley, California, USA

MATTOSO, M., ZIMBRÃO, G., LIMA, A. A. B., et al., 2005, "ParGRES: Middleware para Processamento Paralelo de Consultas OLAP em Clusters de Banco de Dados", In: *2nd Demo Session - 20th Brazilian Symposium on Databases*, pp. 19-24, Uberlândia, MG, url: http://pargres.nacad.ufrj.br, acessada em dezembro de 2006

MYSQL, 2006, "MySQL DBMS", url: www.mysql.com, consultada em agosto de 2006.

NAVATHE, S., KARLAPALEM, K. R. M., 1995, "A Mixed Fragmentation Methodology for Initial Distributed Database Design", In: *Journal of Computer ad Software Engineering*

OLAP COUNCIL, 1998, "APB-1 OLAP Benchmark Release II", url: www.olapcouncil.org/research/bmarkly.htm, acessada em Janeiro 2007

ORACLE, 2003, "Oracle RAC 10g Release 1 Overview", In: *Oracle White Paper*Nov.

ÖZSU, M., VALDURIEZ, P., 1999, "Principles of Distributed Database Systems, 2 ed.", In: *Prentice-Hall*

POSTGRESQL, 2006, "PostgreSQL DBMS", url: http://www.postgresql.org, acessada em novembro de 2006

RÖHM, U., BÖHM, K., SCHEK H-J., 2000, "OLAP Query Routing and Physical Design in a Database Cluster", In: *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, pp. 254-268, Germany, Mar.

SQLSERVER, 2005, "Microsoft SQLServer 2005 DBMS", url: http://technet.microsoft.com/en-gb/library/ms203721.aspx, acessada em abril 2007

STERLING, T., 2001, "An Introduction to PC Clusters of High Performance Computing", In: *International Journal of High Performance Computing Applications*, pp. 92-101,

STÖR, T., MÄRTENS, H., RAHN, E., 2000, "Multi-dimensional Database Allocation for Parallel Data Warehouses", pp. 273-284, Proceedings oh the 26th International Conference on Very Large Databases (VLDB)

SWAMINATHAN, G., 2005, "Strategies for Partitioning Relational Data Warehouses in Microsoft SQL Server", url: http://www.microsoft.com/technet/prodtechnol/sql/2005/spdw.mspx

TPC, 2005a, "TPC BenchmarkTM H - Revision 2.3.0", url: http://www.tpc.org/tpch, acessada em dezembro de 2005

TPC, 2005b, "TPC BenchmarkTM R - Revision 2.1.0", url: http://www.tpc.org/tpcr, acessada em dezembro de 2005.

TPC, 2005c, "Transaction Processing Performance Council", url: http://www.tpc.org/, acessada em dezembro de 2005.