UM MODELO DE CUSTO PARA O PROCESSAMENTO DE CONSULTAS EM BASES DE OBJETOS DISTRIBUÍDOS

Gabriela Gouveia Guedes Loureiro Ruberg

TESE 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:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Marco Antônio Casanova, Ph.D.

Profª. Maria Luiza Machado Campos, Ph.D.

Prof. Gerson Zaverucha, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MAIO DE 2001

RUBERG, GABRIELA GOUVEIA GUEDES LOUREIRO

Um Modelo de Custo para o Processamento de Consultas em Bases de Objetos Distribuídos [Rio de Janeiro] 2001

XXI, 130 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Bancos de Dados baseados em Objetos
2. Processamento de Consultas
3. Modelos de Custo
4. Distribuição de Objetos

I. COPPE/UFRJ II. Título (série)

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

UM MODELO DE CUSTO PARA O PROCESSAMENTO DE CONSULTAS EM BASES DE OBJETOS DISTRIBUÍDOS

Gabriela Gouveia Guedes Loureiro Ruberg

Maio/2001

Orientador: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Técnicas de otimização são essenciais para o processamento eficiente de consultas em bancos de dados baseados em objetos, principalmente na avaliação de expressões de caminho. Tais expressões representam travessias definidas sobre uma seqüência de relacionamentos entre coleções de uma base de objetos e têm sido o foco de inúmeras pesquisas associadas a tecnologias em destaque, como os Sistemas de Gerência de Bases de Dados Objeto-Relacionais e a linguagem XML. A maioria dos trabalhos encontrados na literatura sobre otimização de consultas considera que métricas de custo devem ser utilizadas para orientar a busca por planos de execução com bom desempenho.

Esta tese propõe um modelo de custo que permite estimar o desempenho das principais estratégias para a avaliação de expressões de caminho em bases de objetos distribuídos. Foram especificadas métricas que representam o custo das operações de entrada/saída (E/S) de dados, de instruções de CPU e de comunicação entre nós. A abrangência do modelo proposto diferencia esta tese dos demais trabalhos, pois é considerada uma ampla gama de aspectos relevantes (i) do modelo de dados OO, como a seletividade de expressões de caminho e a existência de relacionamentos parciais, (ii) do modelo de armazenamento de dados segundo diferentes políticas para agrupamento dos objetos, (iii) do modelo de execução dos algoritmos e (iv) do projeto de distribuição das coleções, levando em conta diferentes técnicas de fragmentação e a alocação dos fragmentos.

O tratamento de aspectos relativos ao impacto da distribuição de objetos no processamento de expressões de caminho é a principal contribuição apresentada nesta dissertação, a qual visa preencher uma importante lacuna, em função da ausência de trabalhos relacionados.

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

A COST MODEL FOR QUERY PROCESSING IN DISTRIBUTED-OBJECT DATABASES

Gabriela Gouveia Guedes Loureiro Ruberg

May/2001

Advisor: Marta Lima de Queirós Mattoso

Department: Computer Science and Systems Engineering

Optimization techniques are essential for the efficient query processing in object-based databases, mainly for the evaluation of path expressions. Such expressions represent a path defined over a sequence of relationships between collections of objects and it has been the focus of numerous researches associated to eminent technologies, such as object-relational databases and the XML language. Most works found in the literature about query optimization considers that cost metrics are used to guide the search of execution plans with good performance.

This thesis proposes a cost model that allows the estimation of the performance of various strategies for the evaluation of path expressions over distributed-object databases. Metrics were specified to represent the cost of data input/output operations (I/O), CPU instructions, and communication between two nodes. The wideness of the proposed model distinguishes this thesis from other works, since it covers a high number of relevant aspects of (i) the OO data model, such as the selectivity of a path expression and partial participation of the collections in the relationships, (ii) the model of data storage according to diverse policies of object clustering, (iii) the execution model of algorithms, and (iv) the collections distribution project, considering different techniques of fragmentation and the fragments allocation.

The modeling of aspects related to the impact of object distribution on the evaluation of path expressions is the major contribution presented in this dissertation, which aims to accomplish a great hiatus left due to the absence of related works.

A Nicolaas Ruberg.

"De alma ao vento e mãos cravadas no timão do barco que eu navego, este marujo de olhos cheios de norte traça-me a única rota possível para quem veleja o coração.
E ainda inspira o registro da minha viagem."

AGRADECIMENTOS

A concretização deste trabalho deve-se à importante colaboração de várias pessoas,
o que fez desse processo um aprendizado além da ciência e das letras.
Eu agradeço a todos que se envolveram nessa jornada.

À profª. Marta Mattoso, pela orientação cuidadosa e sempre presente ao longo deste trabalho, indicando-me com presteza os caminhos adequados a serem seguidos. Dedico-lhe minha profunda admiração pela seriedade e devotamento com que atua na pesquisa científica.

Aos membros da banca examinadora, professores Marco Antônio Casanova, Maria Luiza Campos e Gerson Zaverucha, pela generosidade nas críticas a este trabalho e pelas valiosas sugestões. Foi uma grande honra contar com a avaliação de tão conceituados pesquisadores.

Aos professores do Programa de Engenharia de Sistemas e Computação (PESC) da COPPE/UFRJ, em particular ao prof. Jano Moreira de Souza, pelo incentivo e pelos importantes conhecimentos transmitidos nas disciplinas do mestrado.

A Patrícia Leal, secretária da linha de Banco de Dados do PESC, pela disponibilidade e pelo sorriso em todas as ocasiões. A Solange, Lúcia, Sueli, Cláudia, Mercedes e demais funcionários da Secretaria do PESC, pela atenção e eficiência no atendimento às minhas muitas demandas operacionais.

Aos amigos e companheiros do mestrado, Alê, Robson, Marcelo e Gustavo, pela torcida e pelos momentos de descontração. À querida Catarina Rocha, pelo indispensável ombro amigo e agradáveis reuniões em sua casa.

A Fernanda Baião, com quem dividi a programação da versão centralizada das funções de custo propostas neste trabalho, pela dedicação e inúmeros debates no grupo de estudo sobre o processamento de consultas orientadas a objetos. A Jorge Soares e Flávio Tavares, pela ajuda ao fornecerem os cenários experimentais utilizados na validação do modelo de custo.

À Empresa de Processamento de Dados da Previdência Social – Dataprev, pela licença parcial que permitiu-me cursar grande parte deste mestrado. Em especial, a Joel Ramos, Carlos Murucci e Carlos Vaz, pelo empenho em conceder-me tal oportunidade. Pelo amplo incentivo e total compreensão nas minhas ausências, agradeço aos queridos amigos e colegas da Divisão de Administração de Bancos de Dados: Ana, Cristina, Lia (minha revisora oficial), Luís Fernando, Vinícius, Patrícia, Renato, Beni, Levy, Anete e Adelaide. Foi para mim uma experiência única ter trabalhado com profissionais tão experientes e capazes.

Ao Banco Central do Brasil, pela licença que viabilizou a conclusão deste trabalho, através do Programa de Pós-Graduação do Sistema Banco Central de Educação Permanente. Em especial, a Eduardo Nakao, Ruben Galvão e Carlos Henrique Augusto, pela oportunidade e apoio a mim oferecidos. Espero que este Programa de Pós-Graduação continue fortalecido e contribua para a excelência técnica desta respeitada instituição. Aos meus colegas do DEMAB e, especialmente, da DICEL, cujo incentivo e compreensão foram essenciais nesse processo. Registro também meus agradecimentos à querida Ângela Musiello, pela dedicação, constante orientação e disponibilidade.

A Vicente Fernandes, pela orientação técnica no Banco Central do Brasil.

Aos colegas Leonardo, José Roberto e Luís Antônio, pela força providencial.

Ao prof. José Tadeu Fontes Leite, do Núcleo de Estudos e Tecnologia em Engenharia Biomédica (NETEB) da UFPB, com quem aprendi os primeiros passos rumo à pesquisa e obtive recomendação para meu ingresso no mestrado. A dedicação e o exemplo deste verdadeiro mestre, semeador do pensamento crítico e do espírito criativo, será sempre uma fonte de inspiração para o meu trabalho profissional e acadêmico.

Ao professor Gustavo Motta, do Departamento de Informática da UFPB, pelo incentivo e apoio na continuidade dos meus estudos na pós-graduação.

À amiga Fernanda Lima, pelas inúmeras dicas.

Ao meu marido Nicolaas, por não somente ter compreendido as minhas muitas e prolongadas ausências, mas também por ter participado ativamente em todas as etapas desta conquista. Pelas inúmeras sugestões, revisões e correções desta dissertação. Pelo apoio moral, emocional, afetivo, técnico e logístico durante todo o mestrado. Enfim, por sua dedicação excepcional e companheirismo apaixonante.

Aos meus amados pais, Adelma e Edson Loureiro, por participarem tanto e tão positivamente em minha vida. Sua orientação e seus esforços sempre facilitaram a trilha de minhas ambições, ainda mais tendo como modelo a conduta e as muitas qualidades destas pessoas maravilhosas.

Aos meus irmãos, Danielly, Adelminha e Laerte, por compartilharem comigo a vontade de concluir este trabalho. Aos meus avós Elizete e Macário Loureiro, por acreditarem na minha capacidade e pelo exemplo de fortaleza. À vovó Sylvia (*in memoriam*), por ter dedicado a mim muitos momentos de sua vida.

A Angela, Adriana e Claudia Ruberg, pelo incentivo e pelas preces. Aos meus "novos" avós Izolina e Arcídio Sabbanelli, pela acolhida aconchegante que tantas vezes permitiu-me recarregar as forças. A Gerry Ruberg, pelo astral e confiança.

Aos meus queridos amigos de João Pessoa: Micheline e Antônio Silveira, Kalina e Ricardo Abrahão, Micheline Barroso, Márcio Cornélio e Valmir Vaz. Pelas intermináveis horas ao telefone, transfusões de ânimo e motivação.

ÍNDICE DO TEXTO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SÍMBOLOS[1]

- ❖ C_i – Coleção i de uma expressão de caminho ($1 \leq i \leq \ell$)
- ❖ $cl_{i,i+1}$ – Indica se a coleção C_i está agrupada por referência com C_{i+1}
- ❖ Cl_i – Partição física contendo somente objetos da coleção C_i
- ❖ Cl_i^* – Partição física contendo somente objetos da coleção C_i que não participam de relacionamento com outra coleção
- ❖ Cl_i^{**} – Partição física contendo somente objetos da coleção C_i que são apontados por objetos de C_{i-1} em um agrupamento por referência $Cl_{i-1 \rightarrow i}$, mas que não couberam na página-raiz deste agrupamento
- ❖ $Cl_{i-1 \rightarrow i}$ – Partição física que agrupa objetos das coleções C_{i-1} e C_i, tal que C_{i-1} é a raiz do agrupamento
- ❖ $comp$ – Tempo para comparar dois valores na memória
- ❖ $custoCPU\_p_i$ – Tempo de CPU gasto para validar o predicado aninhado p_i na coleção C_i
- ❖ $custoCPU\_p_j^i$ – Tempo de CPU gasto para validar o predicado aninhado p_i no fragmento F_j^i
- ❖ $D_{i,i+1}$ – Número total de ponteiros distintos a partir de objetos da coleção C_i para objetos de C_{i+1}

[1] Tempo é medido em milisegundos (ms).

- $D_{i,j}^{i+1}$ – Número total de ponteiros distintos a partir de objetos da coleção C_i para o fragmento F_j^{i+1}
- $D_{j,i+1}^{i}$ – Número total de ponteiros distintos a partir de objetos do fragmento F_j^i para a coleção C_{i+1}
- *dot* – Tempo para acessar um atributo de um objeto carregado na memória
- E_i – Conjunto de todos os fragmentos da coleção C_i que foram eliminados da consulta na avaliação de uma expressão de caminho
- $Eliminados_i$ – Conjunto de fragmentos da coleção C_i que foram eliminados da consulta devido a SEL_i na etapa de redução da expressão de caminho
- $Eliminados'_i$ – Conjunto de fragmentos horizontais derivados da coleção C_i que foram eliminados da consulta na avaliação da expressão de caminho
- $fan_{i,i+1}$ – Número médio de ponteiros a partir de um objeto da coleção C_i para objetos de C_{i+1}, incluindo as réplicas
- $fan_{i,j}^{i+1}$ – Número médio de ponteiros a partir de um objeto da coleção C_i para objetos do fragmento F_j^{i+1}, incluindo as réplicas
- $fan_{j,i+1}^{i}$ – Número médio de ponteiros a partir de um objeto do fragmento F_j^i para objetos da coleção C_{i+1}, incluindo as réplicas
- *fetch* – Tempo para carregar na memória uma página de dados em disco

- f_i – Número de fragmentos da coleção C_i
- F_j^i – Fragmento j da coleção C_i $(1 \leq j \leq f_i)$
- $frags_i^n$ – Conjunto de fragmentos da coleção C_i, que não foram eliminados da expressão de caminho, alocados no nó n
- $hash$ – Tempo para decodificar um identificador de objeto (IDO)
- ℓ – Comprimento de uma expressão de caminho
- m – Tamanho da memória principal disponível (em páginas)
- M_i – Variável utilizada para verificar a disponibilidade de memória durante a execução dos algoritmos de consulta
- $move$ – Tempo para inserir um par de IDOs em uma tabela de suporte
- $O_{i-1,i}$ – Número médio de objetos de C_i que serão armazenados fora de sua página-raiz em cada unidade do agrupamento $Cl_{i-1 \rightarrow i}$
- $O_{i-1,i}^*$ – Número médio de objetos de C_i que restam, se $\left(O_{i-1,i} \times S_{C_i}\right) > S_P$, após a alocação de parte dos $O_{i-1,i}$ objetos em quantas páginas inteiras forem possíveis
- $part_j^i(fator)$ – Função que retorna a participação do fragmento F_j^i nos objetos selecionados por $fator$ em C_i
- REF_i – Número de objetos distintos recuperados a partir da coleção C_i na avaliação de uma expressão de caminho
- $REF_{i,k}$ – Número de objetos distintos recuperados a partir da partição física k de C_i na avaliação de uma expressão de caminho

- ref_j^i – Número de objetos distintos recuperados a partir do fragmento F_j^i na avaliação de uma expressão de caminho
- REF_v_i – Número total de objetos recuperados a partir de todos os fragmentos verticais da coleção C_i durante a avaliação de uma expressão de caminho
- $R_{i,i+1}$ – Número total de ponteiros a partir de objetos da coleção C_i para objetos de C_{i+1}, incluindo as réplicas
- $R_{i,j}^{i+1}$ – Número total de ponteiros a partir de objetos da coleção C_i para objetos do fragmento F_j^{i+1}, incluindo as réplicas
- $R_{j,i+1}^i$ – Número total de ponteiros a partir de objetos do fragmento F_j^i para objetos da coleção C_{i+1}, incluindo as réplicas
- S_{C_i} – Tamanho médio de um objeto (em *bytes*) da coleção C_i
- $S_{Cl_{i-1\to i}}$ – Tamanho médio de um objeto (em *bytes*) do agrupamento $Cl_{i-1\to i}$
- $selE_i$ – Somatório da seletividade dos fragmentos pertencentes a E_i
- $selEliminados_i$ – Somatório da seletividade dos fragmentos pertencentes a $Eliminados_i$
- $selEliminados'_i$ – Somatório da seletividade dos fragmentos pertencentes a $Eliminados'_i$
- $selFrags_i^n$ – Soma dos fatores sel_j^i dos fragmentos F_j^i que pertencem a $frags_i^n$
- SEL_i – Seletividade do predicado aninhado p_i sobre a coleção C_i

- sel_j^i – Seletividade do fragmento F_j^i em relação à cardinalidade de C_i
- SEL_j^i – Seletividade do predicado aninhado p_i sobre o fragmento F_j^i
- $selReqs_i^n$ – Participação percentual do nó n no volume de páginas requisitadas pela coleção de partida durante a avaliação de uma expressão de caminho, de acordo com a participação dos fragmentos F_j^i que pertencem a $frags_i^n$
- SEL'_i – Seletividade da expressão de caminho até a coleção C_i
- SEL'^i_j – Seletividade da expressão de caminho até o fragmento F_j^i
- $send$ – Tempo para enviar/receber uma página de dados na rede
- $S_{F_j^i}$ – Tamanho médio de um objeto (em *bytes*) do fragmento F_j^i
- $share_{i,i+1}$ – Número médio de ponteiros a partir de objetos da coleção C_i que apontam para um mesmo objeto da coleção C_{i+1}, incluindo as réplicas
- $share_{i,j}^{i+1}$ – Número médio de ponteiros a partir de objetos da coleção C_i que apontam para um mesmo objeto do fragmento F_j^{i+1}, incluindo as réplicas
- $share_{j,i+1}^i$ – Número médio de ponteiros a partir de objetos do fragmento F_j^i que apontam para um mesmo objeto da coleção C_{i+1}, incluindo as réplicas
- S_P – Tamanho da página do banco de dados (em *bytes*)

- ❖ *store* – Tempo para inserir uma entrada em uma tabela de suporte baseada em uma função de dispersão (*hashing*)
- ❖ T_{MSG} – Tempo para inicializar/receber uma mensagem
- ❖ $T_{E/S}$ – Tempo para carregar uma página de dados em memória (uma unidade de E/S)
- ❖ T_R – Tempo para transmitir um *byte* (constante)
- ❖ $X_{i,i+1}$ – Número de objetos da coleção C_i cujos ponteiros para objetos de C_{i+1} são todos nulos
- ❖ $X_{i,j}^{i+1}$ – Número de objetos da coleção C_i cujos ponteiros para objetos do fragmento F_j^{i+1} são todos nulos
- ❖ $X_{j,i+1}^{i}$ – Número de objetos do fragmento F_j^i cujos ponteiros para objetos da coleção C_{i+1} são todos nulos
- ❖ $Z_{i,i+1}$ – Número médio de ponteiros distintos a partir de um objeto da coleção C_i para objetos de C_{i+1}, considerando os objetos de C_i que possuem pelo menos uma referência não nula
- ❖ $Z_{i,j}^{i+1}$ – Número médio de ponteiros distintos a partir de um objeto da coleção C_i para objetos do fragmento F_j^{i+1}, considerando os objetos de C_i que possuem pelo menos uma referência não nula
- ❖ $Z_{j,i+1}^{i}$ – Número médio de ponteiros distintos a partir de um objeto do fragmento F_j^i para objetos da coleção C_{i+1}, considerando os objetos de F_j^i que possuem pelo menos uma referência não nula

- $\#E_i$ – Cardinalidade do conjunto E_i
- $\#frags_i^n$ – Cardinalidade do conjunto $frags_i^n$
- *#nós* – Número de nós da rede
- $\#pageHits_i$ – Número de páginas acessadas na coleção C_i para recuperar os objetos de C_i selecionados pela expressão de caminho
- $\#pageHits_j^i$ – Número de páginas acessadas no fragmento F_j^i para recuperar os objetos de F_j^i selecionados pela expressão de caminho
- *#pageReqs* – Número total de páginas requisitadas nas coleções que são apontadas na avaliação de uma expressão de caminho
- $\#pageReqs_i$ – Número total de páginas requisitadas na coleção C_i durante a avaliação de uma expressão de caminho
- $\#p_i$ – Número de predicados simples do predicado aninhado p_i
- $\hat{}REF_i$ – Número total de ponteiros para a coleção C_i selecionados na avaliação de uma expressão de caminho, incluindo as réplicas
- $\hat{}REF\_v_i$ – Número total de ponteiros (incluindo as réplicas) para a coleção C_i fragmentada verticalmente, considerando o acesso a todos os fragmentos, selecionados na avaliação de uma expressão de caminho
- $\hat{}ref_j^i$ – Número total de ponteiros para o fragmento F_j^i selecionados na avaliação de uma expressão de caminho, incluindo as réplicas

- α_n – Fator de acessibilidade local do nó n às páginas solicitadas nas coleções apontadas durante a avaliação das subexpressões de caminho que começam nos fragmentos alocados em n
- α_i^n – Fator de acessibilidade local do nó n às páginas solicitadas na coleção C_i durante a avaliação das subexpressões de caminho que começam nos fragmentos alocados em n
- $\Delta_{i-1,i}$ – Grau de distribuição física dos objetos de C_i no disco, em relação à alocação física dos objetos de C_{i-1}
- π_i – Número de partições físicas da coleção C_i
- π_j^i – Número de partições físicas do fragmento F_j^i
- $\|C_i\|$ – Cardinalidade da coleção C_i
- $\|C_i\|_k$ – Cardinalidade da partição k de C_i $(1 \le k \le \pi_i)$
- $|C_i|$ – Número total de páginas ocupadas pela coleção C_i
- $|C_i|_k$ – Número de páginas ocupadas pela partição k de C_i $(1 \le k \le \pi_i)$
- $\|F_j^i\|$ – Cardinalidade do fragmento F_j^i
- $|F_j^i|$ – Número de páginas ocupadas pelo fragmento F_j^i
- $|F_j^i|_k$ – Número total de páginas ocupadas pela partição k de F_j^i

Capítulo 1

INTRODUÇÃO

O contexto no qual este trabalho está inserido, os fatos que o motivaram e os principais objetivos desta tese são apresentados neste capítulo, que também inclui um breve resumo sobre os capítulos restantes e a organização do texto.

1.1 MOTIVAÇÃO

Os Sistemas de Gerência de Bases de Dados Orientados a Objetos (SGBDOOs) e Objeto-Relacionais (SGBDORs) surgiram para atender a demanda crescente de aplicações não convencionais, as quais apresentam estruturas de dados complexas e geralmente envolvem o tratamento de volumes consideráveis de informações. Entre estas aplicações, podemos citar os sistemas multimídia, os sistemas para automação de escritório, as ferramentas CAD (*Computer-Aided Design*) e os Sistemas de Informações Geográficas (SIGs) (CATTELL, 1994). Todavia, o poder semântico da orientação a objetos beneficia qualquer aplicação, independente de sua complexidade. Recentemente, recursos provenientes do modelo de dados orientado a objetos têm sido bastante explorados em bancos de dados devido à disseminação dos SGBDORs e do processamento de consultas em XML (RAMASAMY *et al.*, 2000).

Uma das funcionalidades básicas e essenciais de um Sistema de Gerência de Bases de Dados (SGBD) consiste em oferecer uma linguagem declarativa de consulta aos dados armazenados em disco. Através de tal linguagem é possível definir quais dados serão retornados por uma consulta, sem especificar detalhes sobre como esses dados serão recuperados, requerendo assim pouco conhecimento dos usuários sobre as estruturas internas do banco de dados. Todavia, cabe ao SGBD a execução eficiente das consultas e transações submetidas pelos usuários, sendo o módulo responsável por essa tarefa conhecido como processador de consultas.

Cumprir requisitos de eficiência no processamento de consultas em bancos de dados baseados em objetos não é um problema trivial, pois a abstração inerente do modelo de dados orientado a objetos (OO) dificulta sobremaneira o processo de otimização de consultas (MAIER *et al*., 1993). Este problema ocorre principalmente na avaliação de expressões de caminho, que representam um percurso definido sobre uma seqüência de relacionamentos entre coleções de objetos. Expressões de caminho podem ser encontradas, por exemplo, na OQL (*Object Query Language*) (CATTELL *et al*., 2000), na SQL:1999 (*Structured Query Language*) (EISENBERG e MELTON, 1999) e na XML (*eXtended Markup Language*) (DEUTSCH *et al*., 1999). Um vasta pesquisa tem sido realizada sobre técnicas de otimização para o processamento de consultas em bases de objetos (BRAUMANDL *et al*., 2000, SU *et al.*, 2000, ROTH *et al.*, 1999, CHO

et al., 1996, FEGARAS e MAIER, 1995, CLUET e DELOBEL, 1993). A maioria destes trabalhos considera que o processador de consultas utiliza uma função de custo para identificar os melhores planos de execução de cada consulta.

Um modelo de custo permite estimar o desempenho dos planos de execução de uma consulta e consiste em um conjunto de funções baseadas em parâmetros e estatísticas sobre o banco de dados (GRUSER, 1996). Inúmeros modelos de custo para o processamento de consultas foram propostos na literatura, abordando principalmente o cálculo de fatores de seletividade e o processamento de junções em bancos de dados relacionais (CLAUSSEN *et al*., 2000, HAAS *et al*., 1993, MISHRA e EICH, 1992, PIATETSKY e CONNELL, 1989, YAO, 1979). Atualmente, esforços estão sendo focalizados sobre a estimativa de desempenho da avaliação de expressões de caminho em bases de objetos (BRAUMANDL *et al*., 2000), porém a pesquisa realizada até então é fortemente influenciada pelos resultados advindos do contexto relacional. Em geral, os modelos de custo não tratam a participação parcial das coleções nos relacionamentos (OZKAN *et al*., 1996, GRUSER, 1996), restringem-se a apenas uma estratégia de avaliação de expressões de caminho (GARDARIN *et al*., 1995) e assumem políticas simplificadas de armazenamento dos objetos (BRAUMANDL *et al*., 2000, GARDARIN *et al*., 1996), o que dificilmente acontece na prática.

Embora há mais de uma década tem-se abordado o desempenho do processamento de expressões de caminho, muitos problemas nesta área ainda estão em aberto (ÖZSU e VALDURIEZ, 1999). É notória a ausência de modelos de custo para o processamento de consultas em bases de objetos distribuídos. Os poucos trabalhos existentes realizam uma análise superficial e restrita deste tema, ignorando as diferentes técnicas de fragmentação das coleções, o custo de comunicação entre os nós, a política de armazenamento dos objetos e os diversos algoritmos para execução de consultas (EZEIFE e ZHENG, 1999, BELLATRECHE *et al*., 1998a, 1998b, FUNG *et al.*, 1997).

Um considerável número de novas aplicações baseia-se na tecnologia de bancos de dados distribuídos, com destaque para os sistemas de trabalho cooperativo, de gerência de fluxo de documentos e de processos, de teleconferência e de comércio eletrônico (KOSSMANN, 2000). Essa tendência é motivada por fatores como a drástica redução do custos dos computadores pessoais (PCs – *Personal Computers*) em relação aos supercomputadores e a facilidade de expansão gradual, além da possibilidade de

integração de diversos recursos e de sistemas legados. Entretanto, garantir a execução eficiente de consultas em um ambiente de bancos de dados distribuídos é um problema complexo. Neste sentido, é imperativa a necessidade de métricas que auxiliem a escolha das melhores alternativas para o processamento distribuído de uma consulta.

1.2 OBJETIVOS DESTA TESE

O principal objetivo desta tese é propor um modelo de custo para estimar o desempenho das principais estratégias de avaliação de expressões de caminho em bases de objetos distribuídos. Entretanto, alguns objetivos complementares merecem destaque, como a caracterização e a representação de expressões de caminho, o estudo da concepção de modelos de custo para o processamento de consultas, o tratamento de relacionamentos parciais entre coleções de objetos, a análise de técnicas de agrupamento aplicadas a coleções fragmentadas e o levantamento dos algoritmos típicos para a avaliação de expressões de caminho.

O modelo de custo desenvolvido nesta dissertação constitui-se de métricas que representam o custo em tempo total e em tempo de resposta das operações de entrada/saída (E/S) de dados, de CPU e de comunicação entre nós, referentes à avaliação de uma expressão de caminho. Tais indicadores são baseados na representação dos seguintes aspectos, cuja abrangência diferencia esta tese dos demais trabalhos apresentados na literatura: (i) o modelo de dados OO, onde é considerada a existência de atributos multivalorados e é analisada a seletividade de expressões de caminho considerando a existência de relacionamentos parciais; (ii) o modelo de armazenamento de dados, o qual leva em conta diferentes políticas para agrupamento dos objetos; (iii) o modelo de execução, referente aos algoritmos típicos que implementam os operadores de um plano de execução; e (iv) o projeto de distribuição das coleções, que trata do efeito de diferentes técnicas de fragmentação no cálculo da seletividade de expressões de caminho, no modelo de armazenamento de objetos e na execução dos algoritmos.

As funções de custo do modelo proposto são baseadas nas idéias sobre cálculo de seletividade de CHO *et al*. (1996), onde é apresentado um método para calcular fatores de seletividade de predicados que considera a participação parcial das coleções

nos relacionamentos. Tal característica é ignorada pela maioria dos trabalhos encontrados na literatura que avaliam o desempenho do processamento de consultas em bases de objetos (BRAUMANDL *et al*., 2000, GARDARIN *et al*., 1996, OZKAN *et al*., 1996) e influencia a estimativa do fator de seletividade de expressões de caminho, para a qual nós apresentamos um método de fácil cômputo e com pequenas margens de erro. O método tradicional equivalente possui um alto custo computacional e desvios consideráveis quando os fatores de seletividade são baixos (GARDARIN *et al*., 1995, 1996, BELLATRECHE *et al*., 1998a).

Nós estendemos as funções de custo apresentadas em GARDARIN *et al*. (1995), que tratam o efeito do agrupamento de coleções na avaliação de expressões de caminho, ignorado por trabalhos como BRAUMANDL *et al*. (2000), BELLATRECHE *et al*. (1998a), GARDARIN *et al*. (1996) e OZKAN *et al*. (1996). No contexto centralizado, nós incluímos uma análise sobre a partição física decorrente de agrupamentos cujo tamanho seja maior que uma página do banco de dados. Além disso, nós acrescentamos à proposta de GARDARIN *et al.* (1995) um exame sobre a aplicação de técnicas de agrupamento em diferentes tipos de fragmentação das coleções.

As propostas apresentadas na literatura sobre modelos de custo para o processamento de consultas em bases de objetos distribuídos restringem-se a apenas uma técnica de fragmentação das coleções e ignoram a alocação dos fragmentos. Tais propostas consideram apenas o custo de E/S, não avaliam o compartilhamento de objetos na seletividade da expressão de caminho e não estimam o efeito desta seletividade sobre a participação dos fragmentos na consulta (BELLATRECHE *et al.*,1998a, 1998b, FUNG *et al.*, 1997). Além disso, elas assumem uma política simplificada de armazenamento dos objetos e só examinam o desempenho do operador n-ário na estratégia de avaliação descendente de expressões de caminho, o que limita a aplicação das funções de custo correspondentes.

A proposta de modelo de custo desta tese apresenta inovações à medida que possui uma ampla abrangência, pois estima o impacto de diferentes técnicas de fragmentação de objetos e da alocação dos fragmentos na avaliação de expressões de caminho, baseada no custo das operações de E/S, de CPU e de comunicação entre nós de uma rede. O modelo trata a fragmentação vertical, a fragmentação horizontal primária e a horizontal derivada. São levados em conta aspectos relevantes quanto ao

modelo de dados OO, à política de armazenamento dos objetos e aos algoritmos típicos de execução de consultas.

Existem na literatura vários trabalhos sobre a avaliação de desempenho do processamento de consultas OO baseada em experimentos práticos (MATTOSO, 1993, LIMA e MATTOSO, 1996, DEWITT *et al*., 1996, MEYER, 1997, AILAMAKI *et al*., 1999, DARMONT e SCHNEIDER, 1999, TAVARES *et al*., 2000, RAMASAMY *et al*., 2000, OSTHOFF *et al*., 2000, SOARES, 2000, OGASAWARA, 2000, LIMA, 2000, e MANEGOLD *et al*., 2000). Estes trabalhos, devido ao custo de implementação próprio da análise experimental (NICOLA e JARKE, 2000), abrangem um conjunto limitado de casos estudados. Através de um modelo de custo é possível observar o comportamento do sistema através de uma variação exaustiva dos aspectos relevantes no processamento de consultas, complementando assim a avaliação de desempenho experimental. O modelo de custo proposto nesta tese foi motivado pelo acúmulo de expertise em análises experimentais sobre o processamento de consultas em bases de objetos, realizadas pela linha de banco de dados da COPPE/UFRJ, aliado às descobertas sobre o projeto de distribuição de classes de objetos realizadas em BAIÃO *et al.* (1998).

A fidedignidade dos resultados gerados pelo modelo de custo foi examinada através de comparações com resultados experimentais encontrados em TAVARES (1999). Além de uma validação experimental, realizamos simulações sobre configurações sintéticas do *benchmark* OO7, explorando a variação de características relevantes para a avaliação de expressões de caminho em bases de objetos distribuídos.

1.3 APLICAÇÕES DO MODELO DE CUSTO PROPOSTO

O modelo de custo proposto nesta tese pode ser utilizado como parte de um otimizador de consultas OO, para estimar o custo de planos de execução de consultas. São exemplos de otimizadores de consultas o OPT++ (KABRA e DEWITT, 1996) e o otimizador do *lambda-DB* (FEGARAS, 1999). Uma outra aplicação interessante do modelo de custo proposto consiste na inclusão de um otimizador de consultas gerado pela ferramenta OPTGEN (FEGARAS, 1998) em um SGBD baseado em objetos, como o GOA++ (MAURO *et al*., 1997) e o PARGOA (TAVARES, 1999), cujo processo de otimização seja orientado pelas métricas de custo desta tese. Além disso, este modelo

pode atuar como uma ferramenta de auxílio ao projeto de distribuição de objetos, como em BAIÃO *et al*. (2000), permitindo que as melhores configurações de esquema de um projeto sejam identificadas objetivamente através de uma análise quantitativa.

Vale ressaltar que modelo de custo desta tese não visa gerar um indicador único de custo, mas uma variedade de parâmetros que podem ser utilizados nas mais diversas aplicações. Dentre estas aplicações, nós destacamos a estimativa de volume ocupado pelas coleções na etapa de projeto físico de um banco de dados, a configuração do *cache* de dados de um SGBD, a análise de impacto de técnicas para agrupamento físico de objetos e a análise comparativa dos algoritmos típicos para processamento de expressões de caminho. Enfim, os parâmetros de custo apresentados propiciam uma vasta gama de possibilidades de uso e favorecem a compreensão do processamento de expressões de caminho em bases de objetos.

1.4 ORGANIZAÇÃO DO TEXTO

Os estudos e análises realizados no desenvolvimento desta tese foram organizados em cinco capítulos adicionais, como se segue.

O Capítulo 2 apresenta uma revisão bibliográfica sobre as características e problemas do processamento de consultas orientadas a objetos, incluindo uma notação compacta e uniforme para representar expressões de caminho. São examinadas as possíveis extensões de uma expressão de caminho e as implicações da flexibilidade do modelo de dados OO nos relacionamentos opcionais entre as classes. Além disso, são caracterizadas as árvores de processamento de consultas e as estratégias básicas para a avaliação de expressões de caminho. Por fim, são descritas as principais técnicas para distribuição de bases de objetos.

O Capítulo 3 apresenta fundamentos sobre modelos de custo para o processamento de consultas em SGBDs, considerando o modelo de dados OO e a execução distribuída dos algoritmos. Este capítulo aborda aspectos envolvidos na concepção de um modelo de custo, como a definição de métricas de desempenho, a origem dos parâmetros de entrada e a adoção de suposições e restrições. Também são discutidos os custos das operações básicas do processamento de consultas e a importância da validação experimental. Ao final, é exposto um estudo sobre o estado da

arte dos modelos de custo no processamento de consultas OO, onde são destacadas as contribuições e limitações dos principais trabalhos apresentados na literatura.

A proposta deste trabalho, um modelo de custo para o processamento de consultas em bases de objetos distribuídos, é apresentada no Capítulo 4. Inicialmente, posicionamos o modelo de custo proposto face aos trabalhos existentes na literatura, realçando os pontos comuns e as vantagens da nossa solução. O resto do capítulo detalha a estimativa dos indicadores de custo que compõem o modelo, como a seletividade de uma expressão de caminho, o volume de páginas ocupadas pelas coleções e os custos referentes às operações de E/S, de CPU e de comunicação entre os nós de processamento na avaliação distribuída de expressões de caminho.

No Capítulo 5 é verificada a acurácia do modelo de custo proposto através da validação experimental e análise de resultados gerados por uma ferramenta que implementa as funções de custo deste modelo. A validação experimental é baseada em comparações com resultados práticos obtidos na literatura, referentes à execução centralizada e paralela de consultas contendo expressões de caminho. Além disso, o comportamento do modelo de custo proposto é examinado através de simulações sobre configurações sintéticas do *benchmark* OO7, variando características que influenciam o desempenho dos operadores típicos para a avaliação de expressões de caminho em bases de objetos distribuídos.

O epílogo deste trabalho é apresentado no Capítulo 6, onde são reportadas as conclusões gerais e as contribuições desta tese, bem como algumas indicações para trabalhos futuros.

Capítulo 2

PROCESSAMENTO DE CONSULTAS ORIENTADAS A OBJETOS

As principais características e problemas do processamento de consultas orientadas a objetos são abordados neste capítulo, incluindo uma notação formal para representar expressões de caminho. Além disso, são apresentados aspectos sobre árvores de processamento de consultas e técnicas para distribuição de coleções.

2.1 INTRODUÇÃO

Oferecer recursos para a recuperação e a manipulação dos dados armazenados em disco constitui um requisito básico e fundamental de um Sistema de Gerência de Bases de Dados (SGBD). O processador de consultas é o módulo do SGBD responsável por oferecer esse serviço na maneira mais eficiente possível (ELMASRI e NAVATHE, 1994). Para isso, as solicitações dos usuários, geralmente descritas em linguagem declarativa, são analisadas e refinadas sucessivamente pelo processador de consultas, que gera planos de execução otimizados e aplica os algoritmos adequados sobre as coleções.

Há cerca de uma década atrás, já era um consenso na comunidade científica que o sucesso da tecnologia de bancos de dados baseados em objetos dependeria, principalmente, da eficiência apresentada na implementação de suas funcionalidades (MAIER *et al*., 1993). Entretanto, cumprir os requisitos de desempenho exigidos durante o processamento de consultas nos bancos de dados baseados em objetos não é um problema trivial (ÖZSU e BLAKELEY, 1995). Visando atender a este objetivo, uma vasta pesquisa tem sido realizada sobre técnicas de otimização de consultas orientadas a objetos (BRAUMANDL *et al*., 2000, SU *et al.*, 2000, ROTH *et al.*, 1999, CHO *et al.*, 1996, FEGARAS e MAIER, 1995, CLUET e DELOBEL, 1993).

Os SGBDs baseados em objetos oferecem a riqueza semântica inerente do modelo de dados orientado a objetos (OO) e proporcionam um ambiente de desenvolvimento mais integrado. Por outro lado, implementar a abstração inerente do modelo de dados OO dificulta o processamento de consultas, principalmente das expressões de caminho. Uma expressão de caminho representa uma travessia entre objetos definida por uma seqüência de relacionamentos entre algumas coleções de uma base de dados e podem ser encontradas, por exemplo, na OQL (CATTELL *et al*., 2000), na SQL:1999 (EISENBERG e MELTON, 1999) e na XML (DEUTSCH *et al.*, 1999).

2.2 ASPECTOS DO MODELO OO

Um objeto é a abstração de uma entidade do mundo real representada por um identificador único, um conjunto de valores de atributos que define o estado do objeto e

uma interface implementada através de métodos que determinam o comportamento do mesmo (ELMASRI e NAVATHE, 1994). O identificador de objeto (IDO) permanece imutável durante todo o ciclo de vida do mesmo e é utilizado pelo banco de dados para manipular cada objeto da base respeitando sua identidade. Além disso, os objetos armazenados em uma base de dados podem ser compartilhados através de referências aos seus respectivos IDOs.

As linguagens de consulta relacionais possuem um conjunto bem definido de operadores da álgebra relacional e manipulam tipos simples de dados, previamente definidos na base. Isso não ocorre nas linguagens de consulta OO, que manipulam coleções de tipos complexos (possivelmente distintos entre si) e onde o resultado de uma consulta pode ser de um tipo diferente dos definidos para as coleções envolvidas (FEGARAS e MAIER, 1995). Aliás, uma consulta OO pode gerar objetos de um tipo ainda não definido na base de dados.

Tais características, dentre outras, definem os requisitos do processamento de consultas orientadas a objetos, conforme veremos a seguir.

2.2.1 MÉTODOS E ENCAPSULAMENTO

O encapsulamento define que todo acesso às propriedades de um objeto deve ser feito através de uma interface bem definida por métodos. Entretanto, os métodos de classe são geralmente escritos em linguagens de programação de propósito geral e sua otimização é considerada uma tarefa extremamente difícil (ÖZSU e BLAKELEY, 1995). Além disso, o encapsulamento das propriedades físicas dos objetos dificulta sobremaneira a predição de custos no processamento de consultas. Por essa razão, a maioria dos SGBDs baseados em objetos considera que o otimizador de consultas é uma aplicação especial, capaz de "quebrar" o encapsulamento (ÖZSU e VALDURIEZ, 1999, BERTINO e FOSCOLI, 1997). Alguns trabalhos consideram que os objetos "revelam" dados sobre seus custos como parte de sua interface. Esta última abordagem ainda é pouco utilizada.

2.2.2 MODELO DE ARMAZENAMENTO

O modo como os objetos são representados e armazenados no banco de dados é determinante no desempenho do processamento de consultas (GRUSER, 1996). Na maioria dos SGBDs baseados em objetos, as definições sobre as classes e os esquemas são armazenadas separadamente dos objetos. Além disso, cada objeto pode ser armazenado de maneira independente da classe a qual pertença em uma seqüência contígua de bytes (GARDARIN *et al*., 1995).

A abstração proporcionada pelo modelo de dados OO requer técnicas sofisticadas para o armazenamento e o acesso eficiente à base de objetos. Detalharemos algumas destas técnicas nas seções seguintes.

2.2.2.1 COLEÇÕES E EXTENSÕES

Em bancos de dados baseados em objetos, os usuários podem criar tipos complexos de dados através da especificação de classes de objetos. A definição de uma classe descreve o estado e o comportamento, em um nível abstrato, de um tipo de objeto (CATTELL *et al*., 2000). Por outro lado, a extensão de uma classe é formada pelo conjunto de todos os objetos pertencentes a esta classe (BERTINO e FOSCOLI, 1997). Uma coleção representa um subconjunto (possivelmente igual) da extensão de uma classe, pois reúne os objetos pertencentes a esta classe, a um atributo de referência/coleção (lista, vetor, conjunto, etc.) ou ao retorno de uma consulta (CATTELL, 1994).

Em geral, é possível determinar na definição de uma classe qual coleção representará a extensão padrão dos objetos correspondentes. Todavia, uma classe pode não possuir uma coleção associada e ainda assim ter uma extensão, isto é, ter objetos na base. Este é o caso dos objetos armazenados através de um relacionamento de agregação (por exemplo, uma classe `Capítulo` pode possuir objetos armazenados através do relacionamento com objetos da classe `Livro`). Além da coleção representante da extensão padrão, mantida automaticamente pelo SGBD, uma classe pode possuir coleções criadas e mantidas manualmente pelos usuários.

É importante ressaltar que as consultas e transações de um SGBD baseado em objetos não são expressas sobre classes, mas sobre extensões de classes ou sobre determinadas coleções.

2.2.2.2 TIPOS DE IDENTIFICADORES DE OBJETOS

Os identificadores de objetos (IDOs) permitem ao SGBD identificar unicamente cada objeto armazenado na base (ELMASRI e NAVATHE, 1994). Este conceito é fundamental para a implementação de referências entre objetos e influencia diretamente o custo de processamento das consultas.

Existem dois tipos básicos de identificadores de objetos (CATTELL, 1994): físicos e lógicos. Na primeira abordagem, o banco de dados utiliza a composição de endereços físicos (segmento, página, posição no vetor, etc.) para permitir um acesso direto ao objeto no disco, o que agiliza o armazenamento e a recuperação dos mesmos. Porém, a manutenção de IDOs físicos é problemática quando os objetos armazenados precisam ser movidos entre as páginas do banco de dados, por exemplo, em conseqüência de uma atualização.

Na abordagem de IDOs lógicos, o banco de dados gera um valor a partir de uma função de dispersão (*hashing*), de uma estrutura de índice ou de uma outra técnica de mapeamento que corresponda à localização física do objeto. O desempenho da manipulação dos IDOs lógicos é degradado devido à execução freqüente da função de mapeamento. Porém, é solucionado o problema da mobilidade interna dos objetos no banco de dados.

A geração de IDOs lógicos pode ser realizada segundo três técnicas básicas (BRAUMANDL *et al.*, 2000):

- ⇨ Por Função de Dispersão (*hashing*) – onde os IDOs gerados pelo SGBD podem ser totalmente independentes da localização física dos objetos;
- ⇨ Por uma Árvore B – a qual possui a vantagem de ser um recurso disponível na maioria dos SGBDs, mas requer mecanismos para balanceamento de sua estrutura, devido à degradação sofrida pela geração seqüencial dos IDOs; e

- ⇨ Por Mapeamento Direto – esta alternativa consiste em uma técnica semelhante aos ponteiros de desvio utilizados quando objetos com IDOs físicos são movidos no banco de dados e é baseada em apontadores (*handles*) para os endereços físicos dos objetos. A diferença para os ponteiros de desvio é que, por mapeamento direto, o SGBD aloca um *handle* para cada objeto no momento em que os objetos são criados. Se um objeto é movido, seu *handle* é atualizado. Já os ponteiros de desvio são criados apenas quando os objetos são movidos.

Muitos algoritmos para avaliação de expressões de caminho ordenam os seus resultados intermediários pelos IDOs físicos ou pelos endereços físicos dos IDOs lógicos, considerando que assim os objetos estarão armazenados em regiões próximas no disco e será necessário um número menor de operações de E/S (BRAUMANDL *et al.*, 2000).

2.2.2.3 AGRUPAMENTO DE OBJETOS

O acesso aos dados em um SGBD baseado em objetos é muito complexo devido à rica variedade de construtores de tipos disponíveis. Além das técnicas comuns para varredura da base, originárias dos sistemas relacionais, bancos de dados baseados em objetos também oferecem um acesso "navegacional" pelos relacionamentos entre os objetos (BENZAKEN, 1990). Considerando que, na prática, geralmente não é possível carregar o banco de dados inteiro na memória principal, realizar o percurso destas referências em disco pode ser uma operação de alto custo.

Uma estratégia de agrupamento físico de objetos visa minimizar as operações de entrada e saída (E/S) de dados. Para isso, os componentes de um objeto são alocados fisicamente em uma mesma unidade de armazenamento. No melhor caso, quando um agrupamento cabe inteiramente em uma página do SGBD, a taxa de objetos solicitados não encontrados na memória (*object faults*) será mínima (BENZAKEN, 1990). O trabalho apresentado por DEWITT *et al.* (1990) verifica que a existência de agrupamentos entre coleções beneficia a arquitetura de SGBD "servidor de páginas"[2].

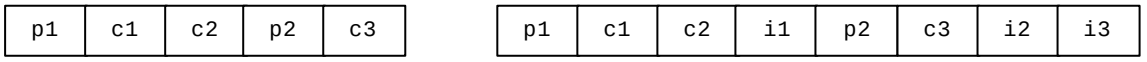
[2] Um "servidor de páginas" é aquele onde a unidade básica para acesso à base é uma página de dados.

Um agrupamento representa o armazenamento de objetos em áreas contíguas no disco, segundo um determinado critério, de modo a minimizar o número de páginas ocupadas e aumentar o desempenho das operações que acessam a base (SHRUFI, 1994). O tipo mais comum de agrupamento reúne objetos de uma mesma classe, também conhecido como o agrupamento por extensão da classe.

Objetos pertencentes a diferentes classes podem ser armazenados juntos, ou seja, em áreas físicas próximas. Assim, é possível reduzir o número de acessos à páginas em disco na recuperação de um objeto com referências para outros objetos. Isto provoca um considerável ganho no desempenho do banco de dados. Existem vários tipos possíveis de agrupamento de objetos. O agrupamento por referência simples armazena juntos os objetos pertencentes a duas classes relacionadas entre si. Caso os objetos de uma classe estejam armazenados próximo a objetos de outras duas ou mais classes, de acordo com relacionamentos de agregação ou associação da primeira para as demais, teremos um agrupamento conjuntivo. A Figura 1 ilustra os dois tipos de agrupamento citados.

```
class Pessoa
 {
   List(Veiculo) carros;
   List<Imovel> imoveis;
   int idade;
   string nome;
   ...
};

p1 = new Pessoa ( {c1,c2},{i1},     40, ”Jorge Silva” )
p2 = new Pessoa ( {c3},   {i2,i3}, 37, ”João Barros” )
```



(a) Agrupamento por referência da classe `Pessoa` com `Veiculo`

(b) Agrupamento conjuntivo da Classe `Pessoa` com `Veiculo` e `Imovel`

Figura 1. Exemplos de agrupamento por referência

O problema de definir qual a melhor estratégia de agrupamento físico dos objetos em um SGBD é semelhante ao problema de fragmentação de classes, pois ambos necessitam identificar grupos de objetos com padrões de acesso semelhantes.

2.2.2.4 ÍNDICES

Índices são uma técnica vastamente explorada no ambiente relacional e representam um mecanismo crucial em sistemas de bancos de dados baseados em

objetos (OGASAWARA e MATTOSO, 1999). Estas estruturas permitem a execução eficiente de consultas que retornam um pequeno subconjunto a partir de um grande volume de dados.

Um índice OO pode ser classificado como (BERTINO e FOSCOLI, 1995): estrutural, quando é baseado nos atributos das classes; ou comportamental, caso vise melhorar o desempenho de consultas contendo chamadas de métodos. Existem diversas estruturas de índices para bases de objetos propostas na literatura. Um índice hierárquico representa um índice único para toda uma hierarquia de classes. Multi-índices são formados pela combinação de um índice simples para cada classe percorrida em uma expressão de caminho. Já índices aninhados provêem uma ligação direta entre os objetos da classe inicial e as instâncias finais de uma expressão de caminho. Por fim, índices sobre caminhos armazenam todas as instanciações da extensão completa à direita de uma expressão de caminho (OGASAWARA e MATTOSO, 1999).

2.2.3 LINGUAGEM DE CONSULTA OO

Uma linguagem de consulta é uma das principais funcionalidades oferecidas por um SGBD. Em vista da ausência de uma linguagem de consulta orientada a objetos amplamente aceita, esforços foram realizados por membros do ODMG (*Object Data Management Group*) no sentido de definir e especificar um padrão formal a ser adotado pelos trabalhos da área (CATTELL *et al.*, 2000).

A linguagem OQL (*Object Query Language*) é uma linguagem similar à SQL que permite a manipulação de objetos segundo o modelo de objetos do ODMG. Esta especificação visa oferecer uma linguagem de consulta declarativa, de fácil compreensão, com maior liberdade em relação à tradicional cláusula *select-from-where* e que não privilegie as operações com conjuntos. A OQL não é uma linguagem computacional completa, mas suporta objetos complexos, tratamento de identidade de objetos, expressões de caminho, agregação, polimorfismo, chamada de métodos, dentre outras funcionalidades necessárias à manipulação de dados em uma base de objetos. O ponto de entrada de uma consulta OQL consiste em um objeto nomeado ou em uma coleção de objetos nomeada, por exemplo a coleção que representa a extensão de uma classe.

A linguagem de consulta OQL suporta tanto objetos, os quais possuem um IDO associado, como literais (objetos simples), cuja identidade é representada por seus valores. Uma expressão OQL pode retornar um literal, um objeto, ou ainda uma coleção de literais ou de objetos. Para criar um novo objeto, deve ser utilizado um construtor de classe disponível no SGBD, pois não existe na OQL uma sintaxe equivalente aos comandos *insert*, *delete* e *update* do SQL. Isto ocorre porque a linguagem OQL considera que a propriedade de encapsulamento deve ser respeitada, onde todas as alterações sobre os objetos de uma classe devem ser realizadas através de sua interface (métodos).

Em uma consulta escrita em OQL, pode-se percorrer ponteiros (IDOs) atinentes aos relacionamentos entre os objetos, "navegando" entre estes relacionamentos. Isto implica em uma maneira diferente de processar a consulta, com acesso direto aos objetos através dos relacionamentos. As expressões de consulta correspondentes a essa navegação entre ponteiros são chamadas de expressões de caminho. Além da navegação por expressões de caminho, coleções que não estão relacionadas explicitamente entre si podem ser declaradas para a realização de junções. Neste caso, o processamento é realizado de maneira análoga ao processamento do SQL padrão.

2.2.4 OTIMIZAÇÃO DE CONSULTAS OO

As consultas submetidas a um SGBD são tipicamente expressas em uma linguagem declarativa que permite definir "quais dados" serão retornados, porém sem especificar "como" o resultado será obtido. Desta maneira, cabe ao processador de consultas identificar aspectos internos da base, como a estrutura de armazenamento dos objetos e a existência de índices de acesso. Com estas informações, o processador de consultas é responsável por decidir sobre a utilização eficiente das estratégias de processamento existentes no SGBD. Em geral, o processo de otimização de uma consulta é realizado segundo as etapas da Figura 2.

Na primeira etapa, a consulta submetida em uma linguagem declarativa é validada léxica e sintaticamente, sendo então normalizada para eliminar as redundâncias. A expressão normalizada é convertida para uma representação interna inicial, equivalente na álgebra de objetos. Existem várias formas para representar uma consulta na álgebra, sendo a árvore de operadores a mais utilizada na literatura. Em

seguida, a expressão algébrica equivalente é avaliada quanto à correção de tipos. Estes passos iniciais são considerados como etapas de preparação da consulta submetida.

Figura 2. Metodologia de otimização de consultas (ÖZSU e BLAKELEY, 1995)

As etapas seguintes são responsáveis pela otimização efetiva da consulta. Na etapa de otimização algébrica, a expressão algébrica válida é refinada sucessivamente através da aplicação de regras de transformação baseadas em heurísticas para melhorar o desempenho da consulta. A saída deste passo é uma expressão algébrica equivalente com desempenho superior. Na etapa final, são gerados planos de execução equivalentes para a consulta, a partir da expressão algébrica otimizada. A escolha dos algoritmos nesta fase é orientada por aspectos físicos da base de objetos. Finalmente, o processador de consultas utiliza uma função de custo para escolher o melhor plano de execução para a consulta.

Em ambientes distribuídos ou paralelos, o otimizador de consultas precisa reconhecer certas características de otimização próprias da distribuição de processamento, como a extração de paralelismo entre os operadores e a redução da comunicação entre os nós.

2.3 EXPRESSÕES DE CAMINHO

Expressões de caminho são um recurso simples e elegante para representar consultas baseadas nos relacionamentos entre os objetos de uma base. Esta Seção apresenta os conceitos relativos a expressões de caminho e adota a notação formal proposta em RUBERG *et al.* (2001a) para este tipo de consulta, a qual será utilizada no restante deste trabalho.

A notação proposta possui algumas vantagens em relação às representações existentes na literatura. Primeiramente, ela consiste em uma representação compacta e

uniforme de um caminho de navegação entre quaisquer coleções relacionadas na base de dados. Cada percurso entre as coleções é definido precisamente por um atributo de referência, evitando assim ambigüidades quanto aos relacionamentos considerados na expressão de caminho. Esta notação é compatível com a OQL, pois é definida sobre coleções, ao invés de classes – como é utilizado em BERTINO e FOSCOLI (1997). Além disso, nossa notação diferencia claramente os atributos de referência e os cursores utilizados na consulta, o que não ocorre na notação apresentada por GARDARIN *et al.* (1996). Tal diferenciação também permite evidenciar as expressões de caminho multivaloradas.

2.3.1 REPRESENTAÇÃO DE UMA EXPRESSÃO DE CAMINHO

Seja $\{T_1, T_2, \cdots, T_\ell\}$ um conjunto de classes onde cada T_i $(1 \le i < \ell)$ possui um atributo A_i, o qual é o atributo de referência para um ou mais objetos de T_{i+1} que será seguido pela expressão de caminho. $\{C_1, C_2, \cdots, C_\ell\}$ é um conjunto de coleções de objetos das classes $T_1, T_2, \cdots, T_\ell$, respectivamente, e C'_i é a coleção de objetos de T_i que são referenciados por objetos de C_{i-1} através do atributo A_{i-1}, tal que $C'_i \subseteq C_i$ $(1 \le i \le \ell)$. Assumimos que $C'_1 = C_1$.

Uma expressão de caminho EC é representada por:

$$EC : x_1[(p_1)].A_1[-x_2][(p_2)].A_2[-x_3][(p_3)]\cdots.A_{\ell-1}[-x_\ell][(p_\ell)] \ ,$$

onde x_i é uma variável, definida na consulta, que representa cada objeto de C_i. Quando A_{i-1} é um atributo multivalorado, x_i representa uma variável do tipo cursor, o qual irá acessar cada objeto de C_i que está sendo referenciado através de A_{i-1}. Neste caso, "$-x_i$" deverá aparecer explicitamente na expressão de caminho. Caso A_{i-1} seja um atributo monovalorado, a variável x_i poderá ser omitida. Os colchetes ([ ]) indicam que o termo é opcional.

O comprimento de uma expressão de caminho EC é definido pelo número de coleções envolvidas (ℓ). C_1, ou C_ℓ, é dita coleção de partida, de acordo com a

estratégia de avaliação da expressão de caminho (se descendente ou ascendente, respectivamente).

Observe as expressões de caminho contidas na consulta da Figura 4, escrita em OQL sobre um esquema simplificado do *benchmark* OO7 (CAREY *et al.*, 1993) mostrado na Figura 3, referentes aos diferentes relacionamentos da coleção `CompositeParts`.

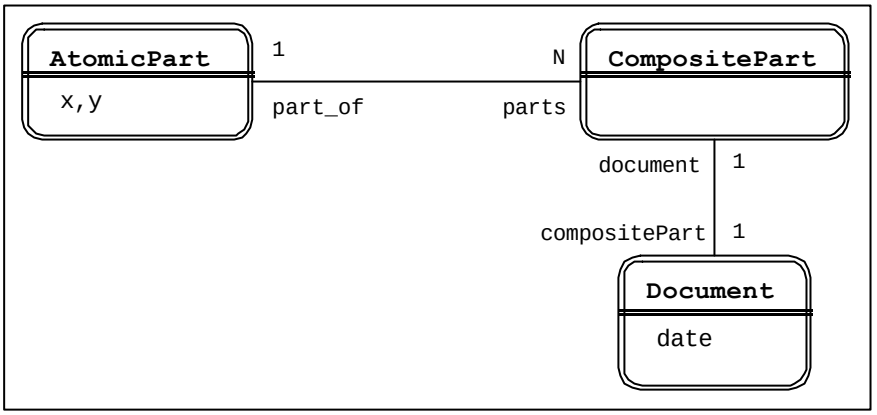


Figura 3. Esquema simplificado do *benchmark* OO7

```
select c
from c in CompositeParts, a in c.parts
where a.x < 900000 and c.document.date < 10/01/1940
```

```
P1: c.parts-a(x<900000)
P2: c.document(date<10/01/1940)
```

Figura 4. Exemplos de expressões de caminho

A cada coleção C_i $(1 \leq i \leq \ell)$ de uma expressão de caminho está associado um predicado p_i, chamado de predicado aninhado. Cada predicado p_i é opcional e estando vazio qualificará todos os objetos de C_i. Os predicados aninhados podem ser formados por uma combinação lógica de expressões matemáticas sobre os atributos das classes pertencentes à expressão de caminho. Cada expressão matemática é da forma $\langle Ai \rangle [\langle operador \rangle \langle valor \rangle]$ (por exemplo, "`x < 900000`"). Em particular, uma expressão da forma $\langle Ai \rangle$ representa a projeção deste atributo no resultado da consulta.

Um predicado aninhado pode conter atributos simples, complexos ou uma combinação de atributos simples e complexos. A existência de um atributo complexo em um predicado aninhado indica apenas a recuperação dos IDOs envolvidos, sem que haja acesso aos objetos correspondentes. Por exemplo, os ponteiros contidos nos atributos complexos (os quais representam IDOs de objetos relacionados) podem ser recuperados para participar de comparações.

Quando todos os atributos A_i $(1 \leq i < \ell)$ de uma expressão de caminho são atributos de referência monovalorados, temos uma expressão de caminho monovalorada. Caso exista pelo menos um atributo de coleção na expressão de caminho, esta é dita multivalorada. Consideramos que os atributos multivalorados possuem domínio restrito a apenas uma classe.

Uma expressão de caminho pode ser decomposta em duas ou mais subexpressões, definidas de modo recursivo. Na avaliação de uma expressão de caminho, diferentes algoritmos podem ser aplicados a cada subexpressão da mesma. Isto implica em um acréscimo considerável de complexidade no processo de otimização deste tipo de consulta.

2.3.2 EXTENSÃO DE UMA EXPRESSÃO DE CAMINHO

Uma expressão de caminho representa um conjunto de tuplas da forma $\{O_1.O_2.\cdots.O_\ell\}$, onde cada tupla contém objetos pertencentes ao caminho que satisfazem os predicados aninhados da expressão (GARDARIN *et al*., 1996). O objetivo de uma expressão de caminho é recuperar apenas os objetos que satisfazem todo o percurso especificado. Entretanto, a existência de relacionamentos opcionais na base de objetos permite que caminhos incompletos sejam varridos durante a materialização da expressão (CHO *et al.*, 1996).

Instâncias parciais de uma expressão de caminho ocorrem quando existem relacionamentos opcionais entre as classes da base de objetos. Ou seja, quando é possível a ocorrência de nulos nos atributos que compõem a expressão de caminho. Podemos identificar os seguintes tipos de extensão em uma expressão de caminho (BERTINO e FOSCOLI, 1997, KEMPER e MOERKOTTE, 1995):

- ⇨ Extensão canônica, que contém apenas as tuplas que satisfazem o expressão completa, isto é, sempre originadas em x_1 e sempre terminadas em x_ℓ;
- ⇨ Extensão completa à esquerda, que contém todos os caminhos originados em x_1 que não necessariamente terminam em x_ℓ;
- ⇨ Extensão completa à direita, definida analogamente à anterior, contém os caminhos terminados em x_ℓ, mas possivelmente originados em algum objeto de x_i que não é referenciado por qualquer objeto de x_{i-1} através de A_{i-1}; e
- ⇨ Extensão completa, que contém todos os caminhos parciais, mesmo que não sejam originados em x_1 ou terminem em um valor nulo.

O padrão ODMG 3.0 (CATTELL *et al.*, 2000) determina que, se existirem relacionamentos opcionais na base, o acesso a uma propriedade (atributo) de um objeto nulo retornará o valor especial *UNDEFINED*. O relacionamento opcional possui impacto em alguns fatores importantes da base de objetos, cujas implicações ainda não foram devidamente exploradas na literatura. Dentre eles, destacamos:

- Na definição da fragmentação horizontal derivada de uma classe, deve existir um fragmento cuja cláusula de seleção seja definida pelo operador especial *ELSE*, onde serão armazenados todos os objetos que não estão relacionados à classe primária;
- Caso haja agrupamento físico por referência entre as coleções, existirão diferentes tipos de partições físicas para armazenar os objetos correspondentes. Estas partições possuem densidades distintas, em número de objetos por página; e
- O fator de seletividade de uma expressão de caminho deverá refletir a participação parcial das coleções correspondentes, visto que a cardinalidade de C'_i poderá ser inferior à cardinalidade de C_i.

2.4 ÁRVORES DE PROCESSAMENTO DE CONSULTAS

As árvores de processamento de consultas constituem uma representação comum para planos de execução de consultas (GARDARIN *et al.*, 1996). Os nós de uma árvore

de processamento correspondem aos operadores de execução da consulta. A ordem em que os operadores serão executados é definida pela topologia da árvore. Anotações sobre os nós e sobre os *links* da árvore tornam mais rica essa representação, indicando aspectos adicionais, como o algoritmo utilizado em cada operador.

Uma árvore de processamento (AP) de uma consulta é uma árvore n-ária rotulada onde os nós-folhas representam as coleções de entrada da consulta (CASANOVA e MOURA, 1985). Além disso, cada nó interno de uma AP é um operador associado a um determinado algoritmo, cujo resultado é uma coleção intermediária. As coleções de entrada da consulta são operandos dos nós internos da AP. Operadores correspondentes a junções possuem sempre dois operandos, os quais podem ser coleções intermediárias. Já os operadores n-ários baseados na navegação por ponteiros podem ter dois ou mais operandos, os quais são nós-folhas da árvore de processamento.

Nós consideramos a existência de três tipos de operadores em uma AP: (i) operadores unários, os quais representam operadores básicos como o de seleção; (ii) operadores binários, relacionados às operações de junção, por exemplo; e (iii) operadores n-ários, como o operador DFF de GARDARIN *et al*. (1996).

É interessante representar com setas direcionadas os *links* entre dois operadores que possuam um canal de comunicação do tipo "*pipeline*" (OZSÜ e VALDURIEZ, 1999). Isto é, quando o segundo operador pode iniciar sua execução antes do término do primeiro. Este tipo de execução não requer que resultados intermediários sejam armazenados. Operadores baseados em ponteiros são exemplos de operadores cuja saída pode ser direcionada para o próximo nó. Observe que esse canal só irá existir entre dois nós se estes forem capazes de gerar e consumir, respectivamente, unidades dos resultados parciais. Um operador de um junção geralmente pode receber apenas uma de suas entradas (o operando externo) através de um canal. As anotações sobre a existência de canais entre os operadores são úteis na determinação das fases de um plano de execução paralelo. Uma fase corresponde a uma subárvore na qual todos os operadores podem ser executados de maneira independente (isto é, em paralelo).

Em um SGBD distribuído ou paralelo, o conjunto de unidades de processamento onde uma coleção está armazenada é chamado de origem da coleção. De maneira

análoga, a origem de um operador corresponde ao conjunto de unidades de processamento onde o mesmo é executado. Quando a origem de um operador é a mesma de seus operandos, não há custo de transferência de dados durante a execução do algoritmo correspondente. Tal característica é mais provável em operadores unários e binários, pois na prática raramente um operador n-ário terá a mesma origem de todos os seus operandos.

Iremos adotar uma notação baseada em árvores de processamento de consultas para representar os planos de execução de uma expressão de caminho. Uma árvore linear é aquela onde cada nó operador possui pelo menos um operando representando uma coleção de entrada da expressão de caminho. Caso pelo menos um operador da árvore de processamento possua todos os operandos representando coleções intermediárias, esta é dita genericamente como árvore densa. Nós consideramos quatro estratégias para a construção de uma AP, conforme pode ser observado na Figura 5, onde as duas primeiras são árvores lineares:

- Árvore Profunda à Esquerda;
- Árvore Profunda à Direita;
- Árvore Zig-zag; e
- Árvore Densa.

A estratégia de avaliação de uma expressão de caminho pode ser definida de acordo com o sentido em que a expressão é resolvida, isto é, em relação à ordem na qual as coleções aparecem na expressão. As duas alternativas básicas são: avaliação descendente, se o percurso sobre coleções é realizado no mesmo sentido em que estas aparecem na expressão de caminho; ou avaliação ascendente, caso contrário (MATTOSO, 1993). A avaliação descendente é representada pela árvore profunda à esquerda, enquanto a avaliação descendente corresponde à árvore profunda à direita.

A árvore de processamento de uma expressão de caminho pode conter subárvores que representam estratégias de avaliação distintas entre si. Neste caso, a estratégia de avaliação da expressão de caminho será híbrida e poderá ser representada ou por uma árvore zig-zag ou por uma árvore densa ("*bushy tree*"). Esta última apresenta características favoráveis ao paralelismo independente entre os operadores, o que pode reduzir bastante o tempo de resposta de um plano de execução.

A representação inicial de uma expressão de caminho resulta geralmente em uma árvore profunda à esquerda ou profunda à direita. O otimizador de consultas pode transformar a árvore de processamento privilegiando o balanceamento. Ou seja, transformações baseadas em heurísticas podem ser aplicadas à árvore inicial de maneira a diminuir a sua altura.

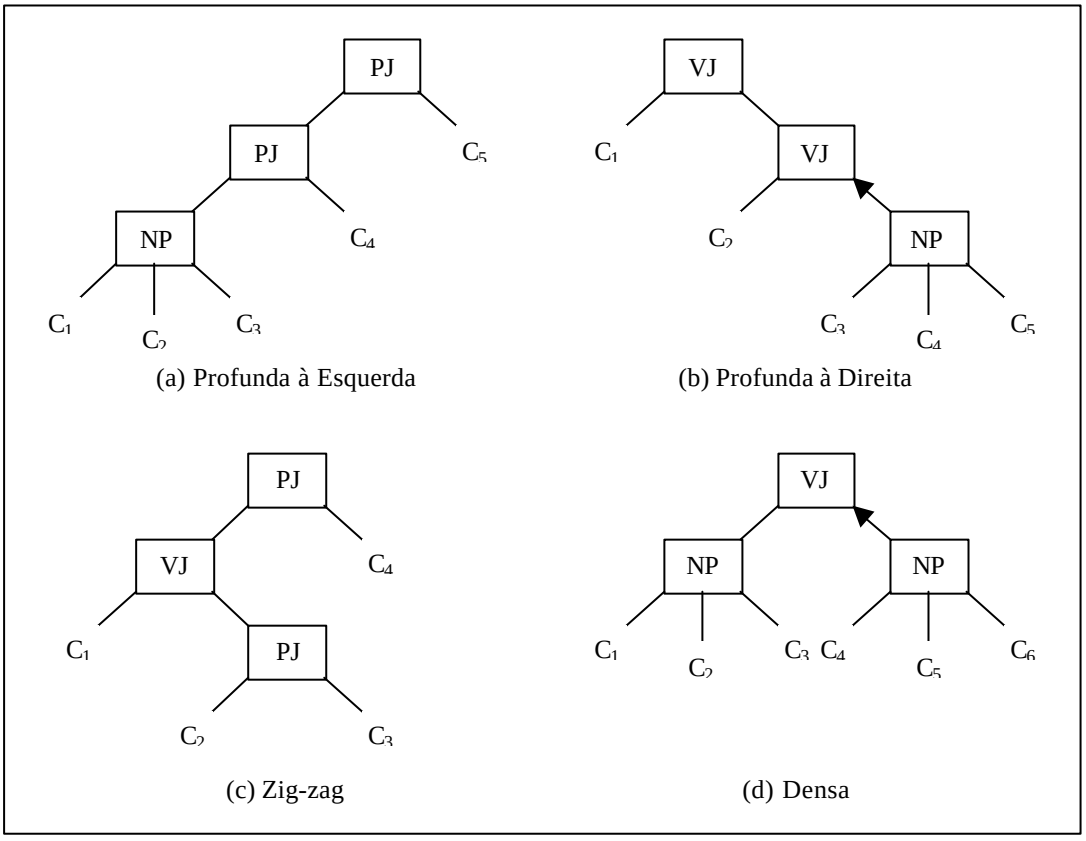


Figura 5. Tipos de árvores de processamento[3]

2.4.1 PONTO DE QUEBRA

Em estratégias híbridas para avaliação de expressões de caminho, um ponto de quebra é o nó onde os resultados de estratégias diferentes se encontram. Quando todos os operandos de um ponto de quebra são coleções intermediárias, não é possível realizar a navegação por ponteiros. Isso ocorre porque os resultados intermediários são armazenados em tabelas de suporte. Neste caso, deve ser utilizado um operador de junção baseada em valor para obter o resultado final.

[3] Os algoritmos utilizados nestes exemplos são *Naive Pointer-chasing* (NP), *Pointer-based Join* (PJ) e *Value-Based Join* (VJ), os quais são descritos na Seção 2.4.3.

2.4.2 NÍVEIS DE PARALELISMO

É possível identificar fases em uma árvore de processamento de uma expressão de caminho, onde determinados operadores desta árvore poderão ser executados em paralelo. Existem três níveis de paralelismo que podem ser explorados no processamento de consultas (MEYER, 1997):

- Paralelismo entre consultas (*Interquery*) – referente à execução concorrente de múltiplas consultas distintas;
- Paralelismo dentro da consulta (*Intraquery*, *Interoperator* ou *Pipelined*) – o qual consiste na execução paralela de diferentes operadores dentro de uma mesma consulta; e
- Paralelismo dentro de um operador (*Intraoperator* ou *Partitioned*) – obtido através da distribuição dos dados de entrada de um operador pelos diversos nós de processamento e da replicação da operação que será executada simultaneamente sobre os fragmentos.

Os benefícios obtidos pelo paralelismo *Pipelined* são limitados por três fatores: (i) o encadeamento entre as operações, (ii) a ausência de suporte a *pipeline* por alguns operadores e (iii) a dificuldade de balancear adequadamente o processamento, visto que o custo de execução de um operador pode ser muito maior que o dos demais. O trabalho apresentado em MENDES e SAMPAIO (1998) apresenta heurísticas para a extração de paralelismo dentro de uma consulta, de acordo com as dependências envolvidas.

O paralelismo dentro de um operador (*Intraoperator*) baseia-se no paralelismo de dados através da composição de algoritmos globais e algoritmos locais. Um algoritmo global, executado na unidade de controle, decompõe as entradas do operador para a posterior execução dos vários algoritmos locais e constrói o resultado final. Esta técnica é utilizada em SGBDs paralelos em MEYER (1997).

2.4.3 GERAÇÃO DO PLANO DE EXECUÇÃO

No contexto de expressões de caminho, as alternativas para a geração de planos de execução podem ser classificadas de acordo com duas dimensões básicas: o sentido da estratégia de avaliação e o tipo dos operadores utilizados no processamento dos relacionamentos.

O sentido da estratégia de avaliação define a ordem na qual as coleções da expressão de caminho serão processadas, podendo ser descendente ou ascendente. A árvore de processamento correspondente será profunda à esquerda ou profunda à direita, respectivamente. Por outro lado, os tipos de operadores básicos para a avaliação de expressões de caminho são os operadores binário e n-ário.

Os algoritmos de junção constituem a implementação típica do operador binário, os quais podem ser divididos em dois grupos: (i) as junções baseadas em valor (VJ – *Value-based Join*), advindas de técnicas vastamente exploradas em SGBDs relacionais (MISHRA e EICH, 1992); e (ii) as junções baseadas em ponteiros (PJ – *Pointer-based Join*), no caso IDOs, que representam um dos principais focos de pesquisa sobre bancos de dados baseados em objetos (BRAUMANDL *et al.*, 2000). A avaliação de uma expressão de caminho por operadores binários consiste na aplicação de junções sucessivas sobre as coleções envolvidas.

O algoritmo clássico para implementar o operador n-ário é o chamado de "busca por ponteiros ingênua" (NP – *Naive Pointer-chasing*). Este algoritmo consiste em percorrer recursivamente, para cada objeto da coleção de partida, todos os ponteiros referenciados na expressão de caminho (de modo "*Depth-First-Fetch*", conforme GARDARIN *et al*., 1996). Uma de suas principais vantagens é a ausência de geração/manutenção de resultados intermediários, que também proporciona um baixo custo de CPU.

Vale ressaltar que os algoritmos baseados em ponteiros, sejam de operadores binários ou n-ários, só poderão ser aplicados na estratégia ascendente quando existirem relacionamentos inversos aos utilizados na expressão de caminho. Isso nem sempre ocorre porque tais relacionamentos não são obrigatórios no padrão ODMG.

A Tabela 1 apresenta a classificação de alguns algoritmos para a avaliação de expressões de caminho encontrados na literatura, segundo as dimensões de direção da estratégia e tipo de operador. Detalhes adicionais sobre as características e contribuições destes trabalhos poderão ser encontrados em RUBERG *et al*. (2001a).

Tabela 1. Dimensões da avaliação de expressões de caminho

TIPO DE OPERADOR / ESTRATÉGIA	OPERADOR N-ÁRIO (*Pointer Chasing*)	OPERADOR BINÁRIO (Junção)	
		Baseado em Ponteiros	*Baseado em Valores*
DESCENDENTE	Gardarin *et al.* (1996), Tavares *et al.* (2000), Keller e Maier (1991)	Shekita e Carey (1990), Braumandl *et al.* (1998), DeWitt *et al.* (1993)	Shekita e Carey (1990)
ASCENDENTE	---	---	Tavares *et al.* (2000), Gardarin *et al.* (1996), DeWitt *et al.* (1993)

2.5 DISTRIBUIÇÃO DE BASES DE OBJETOS

O resultado do projeto de distribuição de bases de objetos pode aumentar o desempenho do processamento de consultas, pois proporciona a execução paralela das operações, reduz o volume de dados irrelevantes acessados e permite que haja otimização na utilização da memória local (LIMA e MATTOSO, 1996). Por outro lado, a distribuição incorre em um significativo aumento de complexidade no controle semântico das coleções e pode impactar negativamente no desempenho das operações que acessem várias partições de dados espalhadas pela rede (BAIÃO, 1997).

Existem muitos trabalhos na literatura que abordam aspectos do projeto de distribuição de bases de objetos, entre eles BAIÃO (1997), BAIÃO *et al.* (1998, 2000), EZEIFE e BARKER (1994), EZEIFE e ZHENG (1999) e BELLATRECHE *et al.* (1998a., 1998b). Entretanto, apenas BAIÃO (1997) e BAIÃO *et al*. (1998) apresentam uma abordagem unificada de projeto que prevê as diversas possibilidades de fragmentação das classes.

Um projeto de distribuição tem por objetivo associar partições de uma base de dados aos diversos nós de um SGBD distribuído ou paralelo. Na abordagem de projeto descendente, esse processo é dividido em duas fases principais (OZSÜ e VALDURIEZ, 1999): Fragmentação e Alocação.

2.5.1 FRAGMENTAÇÃO DE CLASSES

A etapa de fragmentação de uma base de objetos é responsável por agrupar, em fragmentos da base, as informações acessadas por determinados conjuntos de consultas ou transações. Tais fragmentos serão distribuídos pelos nós do sistema na etapa de alocação.

Os tipos básicos de fragmentação de uma classe são: horizontal e vertical (BAIÃO *et al*., 1999). A fragmentação vertical divide os atributos e os métodos de uma classe, distribuindo-os pelos fragmentos. Tais fragmentos verticais possuem um mesmo número de objetos. Um objeto fragmentado verticalmente pode ser representado por um conjunto de objetos distintos, cada um possuindo uma identidade própria, ou por um único IDO, o qual é replicado nos fragmentos verticais do objeto. No caso da fragmentação horizontal, uma classe é representada por várias coleções de objetos com uma mesma estrutura, mas conteúdos diferentes. O que determina o conteúdo de cada coleção é a função de fragmentação utilizada, onde cada fragmento é um subconjunto próprio da extensão original e possui um critério de "seleção" associado.

A fragmentação horizontal pode ser primária, quando os critérios de seleção são definidos sobre a classe que está sendo fragmentada, e derivada, se os critérios de fragmentação dependem de uma classe relacionada. BAIÃO *et al*. (2000) apresentam a definição formal da relação entre a fragmentação horizontal primária e a fragmentação horizontal derivada em bancos de dados baseados em objetos, bem como a definição de quais classes são candidatas à fragmentação horizontal derivada. O algoritmo de projeto proposto neste trabalho é validado com resultados experimentais. Vale ressaltar que essa relação entre a fragmentação horizontal primária e a derivada é bastante diferente do que ocorre no contexto relacional.

As principais técnicas utilizadas para a fragmentação horizontal primária são:

- **Fragmentação por Faixa de Valores** – onde os fragmentos são obtidos em função dos valores dos atributos de cada objeto da base;
- **Fragmentação por Função de Dispersão** (*Hashing*) – a qual se baseia na aplicação de uma função de dispersão sobre os valores de um atributo da classe;

- **Fragmentação Circular** – que consiste em associar seqüencialmente, de modo circular, cada objeto da extensão global a um fragmento, sem considerar o estado de seus atributos.

Em muitos casos, a fragmentação horizontal ou a vertical de uma base de objetos não é suficiente para satisfazer os requisitos das aplicações correspondentes. Assim, a fragmentação vertical pode ser combinada à horizontal, ou vice-versa, produzindo uma fragmentação híbrida da classe e explorando as vantagens advindas das duas técnicas.

2.5.2 ALOCAÇÃO DE CLASSES

Na etapa de alocação, os fragmentos gerados a partir de extensão de classe são distribuídos ou replicados pelos diversos nós do sistema. Um dos principais objetivos deste processo é maximizar o processamento local, alocando o dado o mais próximo possível da aplicação que acessará o mesmo (BARROSO, 1998).

Capítulo 3

MODELOS DE CUSTO PARA O PROCESSAMENTO DE CONSULTAS

Este capítulo apresenta fundamentos sobre modelos de custo para o processamento de consultas em Sistemas de Gerência de Bases de Dados, considerando características do modelo de dados orientado a objetos e da execução distribuída dos algoritmos. Ao final, é realizada uma análise dos principais trabalhos apresentados na literatura.

3.1 INTRODUÇÃO

A avaliação de desempenho no processamento de consultas em bancos de dados pode ser realizada através de experimentos práticos ou por um modelo teórico. Entretanto, avaliações experimentais possuem um alto custo de implementação, além de representarem apenas um determinado conjunto de casos estudados sob condições específicas. Por outro lado, um modelo analítico precisa ter seu comportamento validado através de comparações com um mínimo representativo de casos práticos. Quando devidamente elaborados e testados, modelos teóricos constituem uma ferramenta importante para a predição de desempenho (NICOLA e JARKE, 2000).

Um modelo de custo consiste em um conjunto de fórmulas que permite estimar os custos de processamento dos possíveis planos de execução para cada consulta submetida ao banco de dados. A maioria das pesquisas realizadas sobre técnicas de otimização de consultas orientadas a objetos presume a utilização de um modelo de custo para selecionar os melhores planos de execução (GARDARIN et al., 1995). Desta maneira, funções de custo são tipicamente utilizadas pelo processador de consultas para estimar o custo de execução de uma consulta, o qual engloba os custos de processamento (ou utilização da CPU), de transferência de dados entre a memória secundária e a memória principal (entrada/saída – E/S) e de comunicação, no caso de sistemas distribuídos (ELMASRI e NAVATHE, 1994). Uma atribuição importante das funções de custo é reduzir o espaço de busca durante a otimização de consultas através do descarte de planos de execução "reconhecidamente" ineficientes (ÖZSU e VALDURIEZ, 1999). Além disso, algumas heurísticas aplicadas à consulta ainda na etapa de otimização algébrica dependem da aplicação de funções de custo (GRUSER, 1996).

As metodologias para projeto de distribuição de dados geralmente são baseadas na utilização de heurísticas sobre o processamento de consultas (BAIÃO *et al.*, 1998, 1999, 2000). Um dos principais objetivos do projeto de distribuição é determinar qual configuração de esquema fragmentado permitirá o melhor desempenho para a execução das consultas relevantes. Um modelo de custo pode ser utilizado para uma avaliação quantitativa de cada configuração possível no projeto, permitindo assim que as melhores opções sejam identificadas objetivamente.

Modelos de custo também são utilizados para avaliar: o desempenho de novos algoritmos de execução de consultas (BRAUMANDL *et al*., 2000, SU *et al.*, 2000), políticas para armazenamento físico de objetos (BENZAKEN, 1990), novas estruturas de índices (BERTINO e FOSCOLI, 1995, LEE e LEE, 1998), entre outros. Em REIS e CAMPOS (2000) é proposto um modelo de custo para auxiliar a seleção dos agregados a serem materializados em um *Data Warehouse*, respeitando requisitos de desempenho e minimizando o espaço alocado para o armazenamento. Tal modelo é baseado no número de tuplas envolvidas na geração de cada agregado.

No contexto da avaliação de expressões de caminho, as funções de custo devem considerar a complexidade introduzida pela orientação a objetos. Dentre os aspectos que implicam nesta complexidade, nós destacamos a existência de (i) relacionamentos parciais entre as coleções, de (ii) atributos multivalorados e de (iii) atributos de referência (ponteiros). Estes últimos são a base do conceito de "navegação" entre as coleções e permitem que seja realizado acesso direto aos objetos, o que requer a implementação de algoritmos adequados (SHEKITA e CAREY, 1990). Além disso, para compensar a ausência de normalização das coleções, bancos de dados baseados em objetos permitem definir políticas sofisticadas para o armazenamento de objetos (GARDARIN *et al*., 1995). Estes problemas foram apresentados em detalhes no Capítulo 2.

Por outro lado, a avaliação de expressões de caminho em um ambiente distribuído provoca um aumento considerável na complexidade do modelo de custo (ÖZSU e VALDURIEZ, 1999). Um grande número de parâmetros sobre o sistema e sobre as coleções envolvidas deve ser mantido. Essa complexidade também se estende ao cálculo de parâmetros tradicionais em modelos de custo. Por exemplo, o número de páginas acessadas em uma coleção fragmentada deve considerar apenas as páginas acessadas nos fragmentos que participam da consulta (BELLATRECHE *et al*, 1998a). A concepção de um modelo de custo que considere essas características no processamento de consultas orientadas a objetos não é um problema trivial.

3.2 CRITÉRIOS PARA ANÁLISE DE CUSTOS

Um modelo de custo é especificado de acordo com um conjunto de decisões sobre quais aspectos serão abordados e qual será o nível de detalhamento desta abordagem. Essas decisões dizem respeito às configurações e algoritmos considerados, aos métodos matemáticos empregados, etc., definindo a fronteira de aplicação do modelo de custo. Além disso, para que os resultados obtidos por um modelo de custo possuam significado, é necessário definir quais métricas de desempenho serão adotadas. As métricas mais comuns para avaliar o processamento de consultas são (NICOLA e JARKE, 2000):

- ⇨ Tempo Total – correspondente à soma do tempo gasto em todas as operações realizadas durante a consulta; e
- ⇨ Tempo de Resposta – que é o tempo registrado entre o início e o encerramento da consulta.

Estes critérios estabelecem uma relação de compromisso entre o nível de utilização do sistema e o tempo de resposta da consulta. Em um ambiente centralizado, o tempo total da consulta é muito próximo de seu tempo de resposta. Por outro lado, a execução paralela ou distribuída de uma consulta permite minimizar o tempo de resposta correspondente, embora o tempo total de execução possa aumentar neste caso.

Métricas perceptíveis pelos usuários, como o tempo de resposta de uma consulta, são consideradas como critérios externos de desempenho (NICOLA e JARKE, 2000). Já os critérios internos de desempenho constituem a base da maioria dos modelos de custo, dentre os quais podemos citar o número de páginas acessadas durante uma junção (GARDARIN *et al*., 1995) e o número de objetos recuperados na avaliação de uma expressão de caminho (BERTINO e FOSCOLI, 1997).

3.2.1 TEMPO TOTAL

Em um banco de dados centralizado, o tempo total referente ao processamento local de um plano de execução corresponde ao somatório do custo das operações de entrada e saída de dados (E/S) e do custo de utilização de CPU. Já em um ambiente

distribuído, deve ser considerado também o custo da comunicação entre os nós (OZSU e VALDURIEZ, 1999), de modo que:

$$Tempo\_Total = T_{CPU} \times \#insts + T_{E/S} \times \#E/Ss + T_{MSG} \times \#msgs + T_{TR} \times \#bytes \ , \qquad (1)$$

onde:

- $T_{E/S}$: custo para carregar uma página em memória (uma unidade de E/S);
- T_{CPU} : custo de execução de uma instrução na máquina;
- T_{MSG} : tempo fixo necessário para inicializar/receber uma mensagem;
- T_R : tempo gasto para transmitir um *byte* (constante);
- $\#insts$: número de instruções executadas pelo processador;
- $\#E/Ss$: número de operações de E/S;
- $\#msgs$: número de mensagens trocadas entre os nós; e
- $\#bytes$: número de *bytes* transmitidos durante a execução da consulta.

No caso de bancos de dados distribuídos em redes locais, prevalecem os custos de E/S e de comunicação. Já em redes de longa distância, o custo de comunicação possui impacto decisivo.

3.2.2 TEMPO DE RESPOSTA

A avaliação baseada em tempo de resposta facilita o processo de validação do modelo de custo através de comparações com resultados experimentais equivalentes. Em ambientes distribuídos, essa comparação não é realizada diretamente quando as funções de custo retornam o tempo total dos planos de execução. O tempo de resposta de uma consulta executada em um ambiente distribuído é geralmente inferior ao tempo total gasto pela mesma em uma execução seqüencial. Isso ocorre porque determinados conjuntos de operações da consulta (fases) podem ser executados em paralelo. Assim, temos que (OZSÜ e VALDURIEZ, 1999):

$$\begin{aligned} Tempo\_Resposta = {} & T_{CPU} \times seq\_\#insts + T_{E/S} \times seq\_\#E/Ss \\ & + T_{MSG} \times seq\_\#msgs + T_{TR} \times seq\_\#bytes \end{aligned} \qquad (2)$$

onde $seq\_\#ints$, $seq\_\#E/Ss$, $seq\_\#msgs$ e $seq\_\#bytes$ correspondem respectivamente ao número máximo de instruções, operações de E/S, mensagens ou *bytes* transmitidos que devem ser realizados seqüencialmente no processador "mais lento" para a execução da consulta.

3.2.3 UTILIZAÇÃO DE MEMÓRIA

Além da avaliação baseada em tempo gasto, a estimativa de custos de um plano de execução pode contabilizar também o volume de recursos alocados no sistema. Um importante recurso a ser considerado é a memória principal do SGBD, também conhecida como *cache*. Esta memória geralmente possui tamanho fixo, determinado na configuração do SGBD, e é dividida em três partes principais: (i) o *cache* do catálogo, onde ficam carregados os metadados do SGBD; (ii) o *cache* auxiliar, utilizado para a manutenção de tabelas de suporte e demais estruturas auxiliares aos algoritmos de consulta; e (iii) o *cache* de dados, onde são mantidas as páginas de objetos recuperadas do disco.

O *cache* do catálogo é relativamente pequeno, se comparado aos demais, e é o mesmo para todas as consultas. Por outro lado, é freqüente não haver espaço suficiente no *cache* de dados para carregar todas as páginas solicitadas por um algoritmo. Por isso, uma política para substituição das páginas de objetos é parte fundamental da manutenção deste *cache* (JORDAN, 1998).

É importante observar que o cache auxiliar deve ser suficiente para manter inteiramente as estruturas utilizadas pelos algoritmos de consulta, a fim de evitar operações de entrada e saída de dados desnecessárias e custosas. Assim, o otimizador de consultas pode assumir que certos planos de execução, cujo consumo de memória no *cache* auxiliar seja maior que o disponível no sistema, devem ser descartados. Basicamente, a utilização do *cache* auxiliar é avaliada pelo tamanho das tabelas de suporte que armazenam os resultados intermediários.

Uma tabela de suporte é uma estrutura auxiliar ao processamento de um algoritmo, podendo ser entendida como uma coleção de tuplas de IDOs qualificados e, opcionalmente, atributos relacionados (KEMPER e MOERKOTTE, 1995). Na avaliação de expressões de caminho, esta estrutura é também referenciada como um tipo

de índice de acesso (FIEBIG e MOERKOTTE, 2000, BERTINO e KIM, 1989). Estas tabelas são preferencialmente mantidas na memória principal e podem estar associadas a uma função de dispersão (*hashing*). A aplicação típica de tabelas de suporte é no armazenamento de resultados intermediários provenientes de algoritmos de junção. Além disso, tabelas de suporte podem ser utilizadas como entrada de um algoritmo de consulta (GARDARIN *et al*., 1996).

3.2.4 SUPOSIÇÕES E RESTRIÇÕES

A adoção de suposições e restrições que simplificam o universo analisado é comum a todos os modelos de custo apresentados na literatura (NICOLA e JARKE, 2000). Esta prática é aceitável e justificada porque um modelo de custo visa proporcionar uma análise específica, permitindo que determinados aspectos do contexto sejam estudados isoladamente. Porém, é muito importante que as suposições adotadas no modelo não representem simplificações exageradas da realidade, as quais possam comprometer os resultados obtidos na análise.

Em geral, as suposições e restrições de um modelo de custo visam eliminar do contexto os casos especiais, como o tratamento de objetos longos e o desbalanceamento excessivo de carga. Uma suposição muito utilizada é a que considera o otimizador de consultas capaz de "quebrar" o encapsulamento dos objetos e acessar diretamente informações sobre propriedades físicas (ÖZSU e BLAKELEY, 1995, BERTINO e FOSCOLI, 1997). Essas restrições estão presentes na maioria dos bancos de dados baseados em objetos e, portanto, não limitam o poder de expressividade de um modelo de custo. Uma outra suposição comum em modelos de custo é a distribuição uniforme dos valores de atributo pelas instâncias de uma classe (GARDARIN *et al*., 1995, CHO *et al*., 1996, BELLATRECHE *et al*., 1998a).

3.3 ESTATÍSTICAS E PARÂMETROS DE CUSTO

Um modelo de custo utiliza como parâmetros de entrada um conjunto de estatísticas e propriedades do SGBD, das coleções armazenadas e das aplicações executadas sobre o banco de dados. Existem várias maneiras de prover estatísticas em banco de dados baseados em objetos, da mesma forma que ocorre em sistemas

relacionais. As principais estratégias são baseadas na aquisição automática pelo SGBD, ao comando do administrador do banco de dados, através de amostragem ou da inspeção de todos os dados armazenados (CHO *et al.*, 1996). O custo de manutenção das estatísticas é proporcional à precisão com a qual estas são adquiridas ou calculadas (PIATETSKY e CONNELL, 1984).

São exemplos de estatísticas típicas de bancos de dados baseados em objetos:

- Cardinalidade das coleções;
- Número de fragmentos de cada coleção, quando for o caso;
- Tamanho médio (em *bytes*) do objeto de cada coleção;
- Tipo de armazenamento físico (se existe agrupamento, qual tipo, etc.);
- Existência de índices de acesso;
- Tamanho da página de memória do SGBD; e
- Memória principal disponível, em número de páginas.

Por outro lado, alguns parâmetros de um modelo de custo são derivados de funções matemáticas. Estes parâmetros formam um conjunto de dados essenciais ao processo de otimização. Exemplos destes dados são:

- Número de páginas ocupadas pelas coleções;
- Número de referências entre os objetos relacionados;
- Seletividade de um predicado sobre uma coleção; e
- Seletividade de uma expressão de caminho.

Quanto maior o número de parâmetros utilizados por um modelo de custo, maior será sua complexidade e seu próprio custo computacional. Assim, é importante não considerar um número alto de estatísticas a serem armazenadas e mantidas pelo SGBD. Vale ressaltar também que o emprego extensivo de fórmulas na derivação dos parâmetros de um modelo de custo pode gerar uma propagação de aproximações cuja margem final de erro representa uma alteração significativa nos valores esperados.

3.4 CUSTO DAS OPERAÇÕES BÁSICAS

Conforme vimos no Capítulo 2, um plano de execução de uma consulta consiste em uma determinada combinação de algoritmos que operam sobre as coleções

envolvidas. Cada algoritmo de um plano de execução pode ser decomposto ainda em operações básicas mais elementares, tal que o custo de processamento referente a um dado algoritmo será equivalente ao somatório dos custos de todas estas operações, respeitadas as respectivas freqüências (GRUSER, 1996). No contexto seqüencial, estas operações básicas são:

- *hash* – identifica e retorna o endereço físico correspondente a um ponteiro de objeto (IDO);
- *fetch* – dado um endereço físico de objeto, carrega o mesmo na memória principal, a partir do disco ou a partir da rede, caso este já não esteja no *cache* do SGBD;
- *dot* – retorna o valor de um atributo de um método do objeto;
- *comp* – avalia um predicado simples sobre um atributo e retorna VERDADEIRO ou FALSO;
- *move* – copia um par de IDOs para uma tabela de suporte, no caso da geração de resultados intermediários; e
- *store* – insere uma entrada em uma tabela de suporte através de uma função de dispersão (*hashing*).

Além destas operações, em um banco de dados distribuído também temos a operação *send*, que transmite um pacote de dados (página) entre nós de processamento. Assumimos que o custo desta operação é constante e obtido pela função linear (ÖZSU e VALDURIEZ, 1999):

$$send = T_{MSG} + T_R \times S_P \ , \qquad (3)$$

onde T_{MSG} é o tempo necessário para inicializar/receber uma mensagem, T_R é o tempo para transmitir um *byte* e S_P é o tamanho de uma página do banco de dados (em *bytes*).

Os custos de processamento de cada operação básica geralmente dizem respeito a apenas um tipo de recurso utilizado (E/S de dados, CPU ou comunicação). Em um ambiente distribuído, as operações seqüenciais podem implicar na execução de

operações *send*, cujos custos adicionais devem ser considerados. A Tabela 2 apresenta uma breve análise dos custos envolvidos em cada operação básica.

Tabela 2. Análise de custos das operações básicas

OPERAÇÃO BÁSICA	CUSTOS
hash	$custoIO(hash) = \begin{cases} 0 & \text{se os IDOs são físicos,} \\ constante & \text{caso contrário.} \end{cases}$ $custoCPU(hash) = constante.$
fetch	$custoIO(fetch) = \begin{cases} 0 & \text{se o objeto está na memória,} \\ constante & \text{se precisa carregar a página.} \end{cases}$ $custoCPU(fetch) = 0$.
dot	$custoIO(dot) = 0$. $custoCPU(dot) = constante$
comp	$custoIO(comp) = 0$. $custoCPU(comp) = \begin{cases} 0 & \text{se não houver predicado,} \\ constante & \text{caso contrário.} \end{cases}$ [4]
move	$custoIO(move) = 0$. $custoCPU(move) = constante$.
store	$custoIO(store) = 0$. $custoCPU(store) = constante$.
send	$custoIO(send) = 0$. $custoCPU(send) = 0$. $custoCOM(send) = constante$.

Os fatores que determinam os custos das operações básicas são: (i) o desempenho dos recursos físicos do sistema, isto é, a capacidade de processamento da máquina onde o banco de dados está hospedado, dos dispositivos de armazenamento, das unidades de memória principal e da rede de comunicação; e (ii) o desempenho das

[4] Caso o predicado não seja especificado, ele é definido como VERDADEIRO.

funcionalidades do SGBD, como o tipo de implementação de IDOs, de tabelas de suporte e das funções de dispersão utilizadas.

3.4.1 CUSTOS DE UMA OPERAÇÃO DE E/S

A operação básica *fetch* representa genericamente uma operação de E/S de dados, cujo custo pode ser detalhado de acordo com os passos realizados durante o acesso ao disco (BRAUMANDL *et al*., 2000). Existem três principais etapas em uma operação de E/S onde é consumido tempo de processamento (DARMONT e SCHNEIDER, 1999, HAAS *et al*., 1993):

- ⇨ procura (*seek*) – tempo para mover o cabeçote do disco para o cilindro pesquisado, quando necessário;
- ⇨ latência – durante a qual o disco gira até que o endereço procurado no cilindro seja posicionado sob o cabeçote, quando for o caso; e
- ⇨ transferência – referente ao tempo gasto durante a transferência de uma ou mais páginas de dados entre o disco e a memória principal.

No caso da leitura seqüencial de uma coleção, ocorrerá apenas uma etapa de procura e uma etapa de latência, no início do processo. Basicamente, o tempo gasto em leituras seqüenciais é referente à transferência de dados propriamente dita. Ou seja, o custo para ler seqüencialmente b páginas de dados é calculado por (OZKAN *et al.*, 1996):

$$custoIO(b, Scan) = seek + r + transfer \times b \quad (4)$$

onde $seek$ é o tempo de procura, r é o tempo de latência rotacional média e $transfer$ é o tempo de transferência de uma página.

Para métodos de acesso aleatório, como é o caso de algoritmos baseados em ponteiro, em cada acesso ocorrerão todas as etapas de uma operação de E/S. Deste modo (OZKAN *et al.*, 1996):

$$custoIO(b, Random) = b \times (seek + r + transfer) \quad (5)$$

Quanto menos agrupados estiverem os objetos, pior será o desempenho do acesso aleatório. Observe, entretanto, que o acesso aleatório via ponteiros é interessante quando o volume de páginas a serem acessadas é bem menor que o total de páginas da coleção.

3.5 VALIDAÇÃO EXPERIMENTAL

A acurácia dos resultados gerados por um modelo de custo pode ser avaliada em relação ao comportamento "esperado" dos fatores estudados. Entretanto, nem sempre é possível ou prático determinar este comportamento sem o auxílio de resultados experimentais correspondentes. Além disso, a validação experimental permite que seja avaliado o erro produzido pelas aproximações matemáticas.

A análise experimental constitui uma alternativa interessante para a avaliação de desempenho quando a complexidade matemática dos aspectos que estão sendo analisados é muito alta (NICOLA e JARKE, 2000). Neste caso, a elaboração de um modelo analítico pode ser inexeqüível. Em geral, a avaliação experimental é realizada sobre uma configuração que represente as principais características de uma determinada classe de aplicações, conhecida como *benchmark* (CAREY *et al*., 1993).

São exemplos de avaliações de desempenho baseadas em experimentos práticos os trabalhos apresentados em MATTOSO (1993), LIMA e MATTOSO (1996), MEYER (1997), AILAMAKI *et al*. (1999), DARMONT e SCHNEIDER (1999), TAVARES (1999), RAMASAMY *et al*. (2000), OSTHOFF *et al*. (2000), SOARES (2000), OGASAWARA (2000), LIMA (2000), DEWITT *et al*. (1996) e MANEGOLD *et al*. (2000). Estes trabalhos, devido ao custo de implementação próprio da análise experimental (NICOLA e JARKE, 2000), abrangem um conjunto limitado de casos estudados. Através de um modelo de custo é possível observar o comportamento do sistema através de uma variação exaustiva dos aspectos relevantes no processamento de consultas, complementando assim a avaliação de desempenho experimental.

3.6 O ESTADO DA ARTE

Inúmeros modelos de custo para o processamento de consultas foram propostos na literatura. Grande parte destes trabalhos está concentrada no cálculo de fatores de seletividade e no processamento de junções em bancos de dados relacionais. Atualmente, estes esforços estão sendo focalizados sobre a estimativa de desempenho do processamento de expressões de caminho definidas sobre bases de objetos (BRAUMANDL *et al*., 2000). Naturalmente, esta pesquisa é fortemente influenciada pelos resultados advindos do contexto relacional. Entre os predecessores de modelos de custo para bases de objetos, nós ressaltamos os seguintes trabalhos:

- MISHRA e EICH (1992) apresentam um completo levantamento sobre o processamento de junções em bancos de dados relacionais. Neste trabalho são descritas as características gerais e a aplicação dos diferentes tipos de operadores de junção (junção natural, semi-junção, equi-junção, junção "outer", etc.) e dos principais algoritmos (laços aninhados, "sort-merge", junção baseada em funções de dispersão e em estruturas especiais). O desempenho de cada algoritmo é analisado em linhas gerais no texto, sem a composição de um modelo de custo formal. Aspectos de distribuição e paralelismo também são abordados nas análises;

- HAAS *et al*. (1993) propõem um modelo de custo baseado apenas em operações de E/S. O principal enfoque é o processamento de junções relacionais em consultas *ad hoc* e o detalhamento do custo de E/S, que envolve o custo de busca no disco (*seek*) e da latência rotacional. Os algoritmos de junção baseados em blocos aninhados, "*sort-merge*", em função de dispersão simples, "*Grace*" e "*hybrid-hash*" têm seu comportamento analisado através de simulações do modelo de custo, o qual é validado contra experimentos práticos. Também é considerado o efeito da memória principal sobre os algoritmos;

- Em CHAKRAVARTHY *et al*. (1994), o projeto de fragmentação vertical é orientado por uma função de custo. Uma matriz de utilização de atributos é utilizada para estimar o custo de prováveis esquemas de fragmentação, visando maximizar a acessibilidade local dos nós de processamento. Um módulo é responsável por gerar diversos esquemas, enumerados exaustivamente, e aplicar

a função de custo. Esta função é chamada de Avaliador de Particionamento (PE – *Partition Evaluator*). A métrica de desempenho utilizada é o custo em tempo total e o modelo assume que não há redundância de dados; e

- ➔ HARRIS e RAMAMOHANARAO (1996) revisam os custo dos algoritmos mais comuns de junção, acrescentando a avaliação de CPU. Além disso, em cada algoritmo são identificados os fatores críticos para a redução do custo de processamento, como limites de memória e nível de agrupamento físico. A validação do modelo é realizada experimentalmente.

Embora o desempenho do processamento de expressões de caminho tenha sido o foco de muitos trabalhos há mais de uma década, muitos problemas nesta área ainda estão em aberto (ÖZSU e VALDURIEZ, 1999). Além disso, pouca atenção tem sido dedicada à análise de técnicas de fragmentação de classes através de uma função de custo objetiva (EZEIFE e ZHENG, 1999). A seguir, apresentamos uma coletânea histórica de alguns trabalhos importantes sobre modelos de custo para a avaliação de expressões de caminhos:

- ➔ O exposto em SHEKITA e CAREY (1990) é um trabalho clássico sobre a análise de junções baseadas em ponteiros. O custo é estimado em tempo total das operações de E/S e de CPU. Entretanto, como as junções são analisadas somente sobre duas coleções, os efeitos da profundidade de uma expressão de caminho são ignorados. Esta ausência de contexto em relação ao percurso completo da expressão de caminho também é percebida pela falta de variação na estratégia de avaliação. São analisados os algoritmos de junção baseada em laços aninhados, "*sort-merge*" e "*hybrid-hash*". É considerada uma política simplificada de armazenamento físico dos objetos e não são tratados atributos multivalorados;

- ➔ O modelo de custo de DEWITT *et al*. (1993) consiste na versão paralela do trabalho apresentado em SHEKITA CAREY (1990), onde apenas o custo de E/S é estimado. As simulações são realizadas comparando o desempenho das consultas na presença de agrupamento físico entre os objetos. Este trabalho propõe um novo algoritmo de junção baseada em ponteiros (*Probe-child*) para o processamento de consultas sobre coleções sem extensão explícita (embutidas em outras coleções);

- GARDARIN *et al*. (1995) apresentam um modelo de custo em tempo total (E/S e CPU) para a avaliação de expressões de caminho. Uma contribuição relevante deste trabalho é uma função para estimar o número de páginas ocupadas por coleções fisicamente agrupadas por referência. Além disso, é feita uma extensão à fórmula de Yao (YAO, 1977) para o cálculo de páginas acessadas aleatoriamente em uma coleção. Essa extensão, chamada Yao', visa tratar o efeito de diferentes políticas de armazenamento na avaliação de expressões de caminho. Todavia, a estimativa de páginas ocupadas e a análise de páginas acessadas assumem uma aproximação que ignora a existência de certas partições de objetos, a qual pode gerar erros significativos. Além disso, é verificado apenas o desempenho do operador n-ário na avaliação de expressões de caminho. Por fim, a seletividade de uma expressão de caminho é estimada segundo um método baseado em probabilidade, o qual possui um alto custo computacional e apresenta uma margem de erro considerável;

- CHO *et al*. (1996) propõem um método para calcular o fator de seletividade de predicados de seleção e de expressões de caminho. Os resultados são fornecidos em número de objetos selecionados, incluindo as réplicas, e consideram a participação parcial das coleções no relacionamentos. Em acréscimo, são apresentadas alternativas para a obtenção das estatísticas necessárias para refletir os relacionamentos parciais;

- O modelo de custo apresentado em GARDARIN *et al*. (1996) é uma extensão do proposto em GARDARIN *et al*. (1995), a qual considera junções baseadas em ponteiros e em valor. Como contribuições, este trabalho propõe uma notação para representar expressões de caminho, analisa diferentes estratégias de avaliação de expressões e especifica heurísticas para a aplicação eficiente do operador n-ário de navegação por ponteiros. Embora esta notação seja uma das primeiras a evidenciar o acesso às coleções (ao invés das classes), ela não apresenta uma diferenciação clara entre os atributos de referência e os cursores utilizados na consulta. A análise dos algoritmos abrange a geração de tabelas de suporte para o armazenamento de resultados intermediários, mas é considerada uma política simplificada de armazenamento dos objetos. Por fim, são apresentados resultados experimentais. Entretanto, não é realizada uma validação explícita das funções de custo;

- OZKAN *et al*. (1996) apresentam um modelo que detalha o custo das operações de E/S em relação ao desempenho de leituras seqüenciais ou aleatórias. Este modelo assume que a memória é suficiente para armazenar todos as páginas acessadas durante a navegação, o que geralmente não ocorre. Além disso, não são analisadas as junções baseadas em ponteiro;

- BERTINO e FOSCOLI (1997) abordam o cálculo da seletividade de expressões de caminho considerando principalmente a ocorrência de nulos (participação parcial) e a existência de herança. A notação apresentada não diferencia classes e coleções, nem representa os cursores utilizados na navegação. Além disso, as funções matemáticas são bastante complexas e o custo computacional correspondente restringe a aplicação das mesmas;

- FUNG *et al*. (1997) propõem uma função de custo para avaliar a fragmentação vertical de objetos, baseada apenas no custo de E/S. O enfoque deste trabalho é a execução de métodos de classe. Apenas o operador n-ário de navegação é avaliado e não é considerada a participação parcial das coleções nos relacionamentos. Além disso, não é considerado o grau de compartilhamento dos objetos;

- BELLATRECHE *et al*. (1998a) elaboram um modelo de custo baseado em E/S para analisar o efeito da fragmentação horizontal primária, baseado no modelo proposto em GARDARIN *et al.* (1995). Como as estatísticas são mantidas sobre as coleções e não sobre os fragmentos, este modelo de custo não apresenta um bom comportamento quando a função de fragmentação não possui uma distribuição uniforme. O mesmo modelo de custo é utilizado em BELLATRECHE *et al*. (1998b) para analisar a fragmentação horizontal derivada. Entretanto, não é estimado o efeito da seletividade da expressão de caminho sobre a participação dos fragmentos de uma coleção na consulta;

- BRAUMANDL *et al*. (1998) analisam o impacto dos diferentes tipos de identificadores de objetos (IDOs) na avaliação de expressões de caminho. O modelo de custo utilizado é aplicado a algoritmos de junção baseada em ponteiros e em valor. Este trabalho propõe um novo algoritmo, chamado P(PM)*M, o qual é baseado na ordenação e na fragmentação dos ponteiros que serão utilizados para acessar, através de IDOs, os objetos de uma coleção. Uma

plataforma experimental é utilizada para comparar os resultados do algoritmo proposto. O modelo de custo, baseado em HAAS *et al*. (1993) é também validado com os resultados experimentais;

- ➔ A função de custo apresentada em EZEIFE e ZHENG (1999) utiliza a idéia de CHAKRAVARTHY *et al.* (1994). Este trabalho propõe uma metodologia para projeto de fragmentação horizontal de objetos baseada em uma função de custo (OHPE – *Object Horizontal Partition Evaluator*) que fornece o número de objetos acessados em cada coleção de uma consulta. Ou seja, não são examinadas implicações referentes à utilização de recursos físicos, como por exemplo gerência de *cache* e acesso ao disco;
- ➔ BOULOS e ONO (1999) definem uma técnica baseada em histogramas para estimar automaticamente o custo da execução de métodos de classe definidos pelos usuários. Os custos dos métodos são classificados como constantes, monótonos (cuja variação é bem previsível) ou não-monótonos (de execução complexa e difícil estimativa). Experimentos validam a técnica proposta.

Modelos de custo também exercem um importante papel na otimização global de consultas em bancos de dados heterogêneos (ROTH *et al.*, 1999). Entretanto, neste caso, nem sempre é possível estimar o custo das consultas com um bom nível de detalhamento. Isso ocorre porque muitas informações locais podem não estar disponíveis. Uma alternativa para viabilizar a estimativa de custos neste ambiente é a identificação de grupos de consultas com comportamentos similares (ZHU e LARSON, 1998). Assim, o desempenho de algumas consultas é monitorado (como uma caixa-preta) e associado aos grupos correspondentes. A partir daí, um modelo de regressão é utilizado para definir os custos de cada membro destes grupos.

Conforme visto nos trabalhos apresentados, não existe um modelo de custo único para o processamento de consultas em bancos de dados baseados em objetos. O que há na literatura é uma grande diversidade de modelos, cada um analisando um determinado conjunto de características e assumindo hipóteses próprias. Entretanto, a abordagem destes trabalhos é fortemente influenciada pela pesquisa advinda dos SGBDs relacionais e muitos aspectos importantes do modelo de dados OO ainda não foram devidamente tratados (por exemplo, a existência de relacionamento parciais). Além disso, é notória a ausência de modelos de custo para o processamento de consultas

em bases de objetos distribuídos. Os poucos trabalhos existentes realizam uma análise muito superficial e restrita deste tema, ignorando as diferentes técnicas de fragmentação das coleções, a política de armazenamento dos objetos e os diversos algoritmos para execução de consultas (EZEIFE e ZHENG, 1999, BELLATRECHE *et al*., 1998a, 1998b, FUNG *et al.*, 1997).

No próximo Capítulo, será proposto um modelo de custo para estimar o impacto das diferentes técnicas de fragmentação no processamento de expressões de caminho, baseado no custo das operações de E/S, de CPU e de comunicação. Neste modelo, são considerados aspectos como a seletividade da expressão de caminho, diferentes políticas de armazenamento dos objetos e os algoritmos típicos para a avaliação de expressão de caminho.

Capítulo 4

APRESENTAÇÃO DO MODELO DE CUSTO PARA BASES DE OBJETOS DISTRIBUÍDOS

Este capítulo apresenta a proposta deste trabalho: um modelo de custo para o processamento de consultas em bases de objetos distribuídos, o qual permite estimar o desempenho de consultas considerando aspectos relevantes do modelo de dados OO, da política de armazenamento dos objetos, da execução dos algoritmos mais representativos e das técnicas de fragmentação de dados.

4.1 INTRODUÇÃO

A concepção de um modelo de custo é baseada na análise de aspectos referentes ao modelo de dados, de armazenamento e de execução do SGBD, o que tende a tornar as funções de custo muito específicas. Cada novo algoritmo proposto ou novo recurso do banco de dados é geralmente validado através de um modelo de custo próprio (BRAUMANDL *et al*., 2000, SU *et al*., 2000).

Entretanto, estes modelos não permitem em geral realizar uma análise comparativa de diferentes trabalhos sobre um mesmo tema. Por outro lado, quanto mais genérico o modelo de custo, mais distante ele estará de fornecer resultados realísticos. Em um otimizador de consultas, cujo objetivo é saber "instantaneamente" o desempenho de uma consulta no banco de dados, um modelo de custo muito genérico possui uma aplicação muito restrita. Já na avaliação de novas metodologias, técnicas e decisões de projeto de banco de dados, é necessário uma ferramenta que permita avaliar e simular o desempenho futuro do sistema em várias situações, sem privilegiar recursos de um banco de dados específico. Ainda neste caso, a definição de um modelo de custo deve atentar para as principais implementações dos fatores que apresentam maior influência sobre o sistema analisado.

As expressões de caminho representam o percurso por uma determinada seqüência de relacionamentos entre coleções de uma base de objetos e constituem um recurso valioso em SGBDOOs e SGBDORs. O desempenho da avaliação de uma expressão de caminho pode variar muito, de acordo com a estratégia da árvore de processamento da consulta e com os algoritmos utilizados no plano de execução correspondente (TAVARES *et al*., 2000). Outros fatores também têm um impacto importante sobre este desempenho, como: a existência de agrupamento físico entre as coleções armazenadas (GARDARIN *et al*., 1995); a participação parcial das classes nos relacionamentos da expressão de caminho (CHO *et al*., 1996); o tipo de fragmentação das coleções envolvidas na consulta (BELLATRECHE *et al*., 1998a); e o grau de compartilhamento dos objetos referenciados no percurso.

O modelo de custo proposto neste trabalho permite estimar o desempenho das principais estratégias de avaliação de expressões de caminho, baseado na representação

típica de cada aspecto analisado. As métricas utilizadas computam o custo em tempo total e em tempo de resposta das operações de entrada/saída de dados, de CPU e de comunicação entre nós. Nosso modelo de custos abrange os seguintes aspectos:

- ⇨ O modelo de dados OO, onde é considerada a existência de atributos multivalorados e é analisada a seletividade de expressões de caminho segundo a participação parcial das classes nos relacionamentos;

- ⇨ O modelo de armazenamento de objetos, o qual determina os vários tipos de partições físicas de uma coleção, de acordo com diferentes políticas para agrupamento dos objetos;

- ⇨ O modelo de execução de consultas OO, que inclui a análise dos algoritmos mais representativos para operadores de consultas e das principais estratégias na avaliação de expressões de caminho; e

- ⇨ O projeto de distribuição das classes, onde é abordado o efeito de diferentes técnicas de fragmentação no cálculo da seletividade de expressões de caminho, no modelo de armazenamento de objetos e na execução dos algoritmos.

Para evitar um nível de complexidade muito alto, o qual implicaria em um modelo de custo com pouca utilidade prática, nós assumimos no modelo de custo proposto algumas simplificações sobre o banco de dados e o modelo de objetos. Nós consideramos que: (i) o otimizador de consultas é capaz de violar o encapsulamento dos objetos; (ii) os objetos armazenados possuem tamanho menor ou igual a uma página de dados do SGBD; e (iii) os atributos multivalorados possuem uma única classe como domínio. A restrição (ii) exclui o tratamento de objetos longos do contexto do nosso trabalho, sendo necessária devido às fórmulas utilizadas na estimativa de volume das coleções em disco e de páginas acessadas aleatoriamente (pela fórmula de Yao, conforme Anexo I). Entretanto, nós propomos um método para calcular o volume de páginas ocupadas por agrupamentos físicos maiores do que uma página do SGBD, cujo princípio pode ser aplicado na estimativa de volume de coleções com objetos longos. Observe que estas restrições estão presentes na maioria dos SGBDOOs e dos SGBDORs, bem como em suas aplicações típicas (ÖZSU e BLAKELEY, 1995, GARDARIN, 1995, BERTINO e FOSCOLI, 1997). Portanto, estas suposições não limitam o poder de expressividade do nosso modelo de custo. Além disso, não

consideramos a existência de índices de acesso sobre as coleções e assumimos que a distribuição de valores dos atributos é uniforme.

4.1.1 TRABALHOS RELACIONADOS

Conforme dito, o modelo de custo proposto abrange os algoritmos de execução de consulta mais representativos e as principais estratégias para a avaliação de expressões de caminho. As funções de custo do nosso trabalho são baseadas nas idéias sobre cálculo de seletividade apresentadas em CHO *et al*. (1996), onde é abordada a participação parcial das classes nos relacionamentos. Outros modelos de custo não consideram a existência de relacionamentos parciais (OZKAN *et al*., 1996, GARDARIN *et al*., 1996). Esse aspecto influencia diretamente o cálculo do fator de seletividade das expressões de caminho, cujo método típico para estimativa apresenta um alto custo computacional e gera desvios consideráveis dos resultados esperados quando os fatores de seletividade são baixos (GARDARIN *et al*., 1995, 1996, BELLATRECHE *et al*., 1998a). Para tratar este problema, nós propomos um método de fácil cômputo e com margens de erro irrisórias.

Técnicas para agrupamento físico de objetos não são consideradas na maioria dos modelos de custo para o processamento de consultas OO, como em BRAUMANDL *et al*. (2000), GARDARIN *et al*. (1996) e OZKAN *et al*. (1996). GARDARIN *et al*. (1995) apresentam um modelo de custo sobre o efeito do agrupamento de coleções na avaliação de expressões de caminho, porém a análise realizada limita-se à estratégia de avaliação baseada no operador n-ário. No contexto centralizado, nós estendemos o trabalho de GARDARIN *et al.* (1995) para incluir uma análise sobre a partição física decorrente de agrupamentos cujo tamanho seja maior que uma página do banco de dados. Além disso, nós acrescentamos à proposta de GARDARIN *et al.* (1995) um exame sobre o efeito de diferentes tipos de fragmentação das coleções.

Nos algoritmos baseados em ponteiros, caso a memória não seja suficiente para conter todos os objetos solicitados, podem ser necessários vários acessos ao disco para a recuperação de um único objeto. Os modelos apresentados em OZKAN *et al.* (1996) e GARDARIN *et al.* (1996) realizam a predição deste custo adicional de E/S baseados no grau de saída das coleções (*fan-out*). Entretanto, nós consideramos que esta estimativa depende do grau de compartilhamento das coleções (*share*), pois os objetos de uma

coleção serão recarregados na memória de acordo com o número médio de referências para cada objeto.

Existem poucos modelos de custo para o processamento de consultas em bases de objetos distribuídos na literatura. As propostas apresentadas em BELLATRECHE *et al.* (1998a, 1998b) abordam métricas para orientar o projeto de fragmentação horizontal de classes em SGBDOOs. De maneira análoga, FUNG *et al.* (1997) propõem métricas para a fragmentação vertical das classes baseadas no custo da execução de métodos. Entretanto, tais trabalhos consideram apenas o custo de E/S, não avaliam o compartilhamento de objetos na seletividade da expressão de caminho e não estimam quantitativamente o efeito desta seletividade sobre a participação dos fragmentos na consulta. Além disso, eles consideram uma política simplificada de armazenamento dos objetos e só avaliam o desempenho do operador n-ário na estratégia descendente, o que limita a aplicação das funções de custo correspondentes. Nossa proposta consiste em um modelo de custo que estima o impacto das diferentes técnicas de fragmentação no processamento de expressões de caminho, baseado no custo das operações de E/S, de CPU e de comunicação entre nós de uma rede. Nosso modelo de custo considera todos os aspectos mencionados nesta Seção referentes ao modelo de dados OO, à política de armazenamento dos objetos e aos algoritmos de execução de consultas.

4.2 PARÂMETROS E ESTATÍSTICAS NECESSÁRIAS

A análise dos custos envolvidos na avaliação de uma expressão de caminho é baseada em um conjunto de parâmetros e estatísticas sobre a base de objetos. Alguns destes parâmetros são derivados de funções matemáticas que constituem uma parte fundamental do modelo de custo, como é o caso do fator de seletividade de uma expressão de caminho. A Lista de Símbolos apresentada no início desta tese reúne a definição de todos os parâmetros utilizados no modelo de custo proposto. Basicamente, estes parâmetros podem ser classificados em:

(i) Parâmetros do Banco de Dados – referentes às propriedades específicas do SGBD, como o tamanho de uma página de dados (S_P), número de páginas do *cache* (m) e o tempo de CPU necessário para decodificar um IDO (*hash*);

(ii) Parâmetros das Coleções – referentes à definição e ao armazenamento das coleções da base de objetos, como a cardinalidade de cada coleção ($\|C_i\|$), o número de páginas ocupadas ($|C_i|$) e o grau de saída de seus respectivos relacionamentos ($share_{i,i+1}$); e

(iii) Parâmetros da Consulta – referentes às características da expressão de caminho, como o comprimento da mesma (ℓ), o fator de seletividade de um predicado aninhado p_i (SEL_i) e o número total de objetos da coleção C_i recuperados durante a avaliação da expressão de caminho (REF_i).

Os parâmetros $fan_{i,i+1}$ (*fan-out*) e $share_{i,i+1}$ refletem o grau de saída e o grau de compartilhamento, respectivamente, de um determinado relacionamento entre as coleções C_i e C_{i+1}. Ou seja, $fan_{i,i+1}$ é o número médio de ponteiros presentes em um atributo de referência A_i de um objeto da coleção C_i, os quais apontam para objetos de C_{i+1}, incluindo as réplicas. Por outro lado, $share_{i,i+1}$ é o número médio de ponteiros presentes no atributo A_i da coleção C_i que apontam para um mesmo objeto da coleção C_{i+1}, incluindo as réplicas.

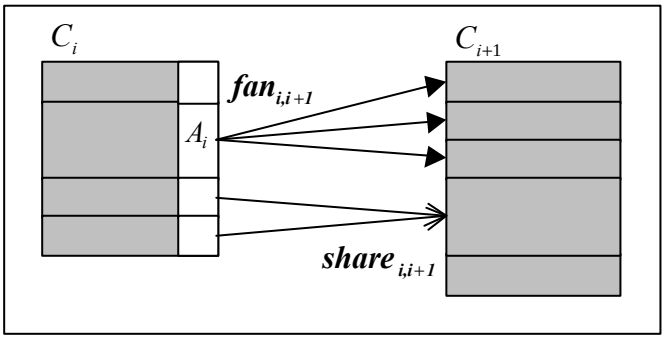


Figura 6. Grau de saída (*fan-out*) e de compartilhamento (*share*) entre coleções

Observe que $fan_{i,i+1}$ é o grau de saída dos ponteiros de objetos da coleção C_i para C_{i+1} e que $share_{i,i+1}$ é o grau de compartilhamento dos objetos da coleção C_{i+1} em relação à C_i, considerando um determinado relacionamento entre estas coleções. Caso existam vários relacionamentos entre as coleções C_i e C_{i+1}, cada relacionamento terá

um grau de saída e um grau de compartilhamento próprios, de acordo com o número de ponteiros existentes no respectivo atributo de referência.

Os parâmetros $fan_{i,i+1}$ e $share_{i,i+1}$ geralmente são calculados sem considerar a participação parcial das classes no relacionamento (BELLATRECHE *et al*., 1998a, GARDARIN *et al.*, 1996). A fórmula apresentada em CHO *et al*. (1996), a qual é considerada no nosso modelo de custo, oferece maior precisão para este cálculo. Estes parâmetros são obtidos por:

$$fan_{i,i+1} = \frac{R_{i,i+1}}{\|C_i\|} \tag{6}$$

e $$share_{i,i+1} = \frac{R_{i,i+1}}{D_{i,i+1}} \ . \tag{7}$$

Além destes parâmetros, é necessário estimar o número médio de ponteiros distintos a partir de um objeto da coleção C_i para objetos de C_{i+1}, levando em conta somente os objetos de C_i que possuem pelo menos uma referência não nula para C_{i+1}. Este parâmetro é dado por:

$$Z_{i,i+1} = \frac{D_{i,i+1}}{\|C_i\| - X_{i,i+1}} \ . \tag{8}$$

Por fim, consideramos que as consultas são submetidas sobre coleções e, por causa desta suposição, algumas estatísticas são mantidas sobre as coleções e não sobre as classes. Do mesmo modo, consideramos que são mantidas estatísticas sobre os fragmentos de uma coleção.

4.3 SELETIVIDADE DA EXPRESSÃO DE CAMINHO

O principal sustentáculo das técnicas de otimização no processamento de consultas baseadas em custo é o cálculo do fator de seletividade dos predicados de seleção e das junções (CHO *et al.*, 1996). O fator de seletividade de uma condição

representa a porção de objetos que a satisfazem e determina o tamanho do resultado intermediário correspondente, quando for o caso. A flexibilidade e o alto nível de abstração do modelo de dados OO tornam bastante complexa a estimativa da seletividade de uma expressão de caminho (BERTINO e FOSCOLI, 1997).

Uma expressão de caminho constitui uma seleção de objetos através de uma seqüência de junções implícitas e predicados aninhados. As fórmulas tradicionais para o cálculo de seletividade (GRUSER, 1996, ÖZSU e VALDURIEZ, 1999) não são aplicáveis às consultas OO, pois as coleções envolvidas podem participar parcialmente nos relacionamentos da expressão de caminho (CHO *et al*., 1996). Aliás, estes relacionamentos parciais também influenciam a estimativa da seletividade dos predicados aninhados.

O fator de seletividade de uma expressão de caminho corresponde à seletividade resultante dos predicados aninhados e da participação de cada coleção nos relacionamentos do caminho, de maneira acumulativa. Quando são utilizados algoritmos baseados em ponteiros, este fator permite determinar o número de objetos que serão acessados em cada coleção durante a avaliação da expressão de caminho. Além disso, no caso dos algoritmos de junção (baseada em ponteiros ou em valor), a seletividade de uma expressão de caminho representa a cardinalidade dos resultados intermediários gerados. O cálculo deste fator em cada coleção C_i $(1 \le i \le \ell)$ também depende de onde a expressão de caminho começa a ser avaliada, ou seja, da direção da estratégia (descendente ou ascendente).

Seja a avaliação descendente de uma expressão de caminho de tamanho $\ell = 3$ definida por $EC : x_1(p_1).A_1 - x_2(p_2).A_2 - x_3(p_3)$, onde as variáveis x_1, x_2 e x_3 representam objetos das coleções C_1, C_2 e C_3, respectivamente, como mostrado na Figura 7. Na Figura 7.(a), note que as coleções C_1, C_2 e C_3 podem participar parcialmente nos relacionamentos da expressão de caminho. Na Figura 7.(b), verificamos que não há nenhum relacionamento apontando para a coleção de partida (neste caso, C_1) e a seletividade da expressão de caminho até esta coleção é total, ou seja, todos os objetos de C_1 são recuperados e submetidos ao predicado aninhado p_1.

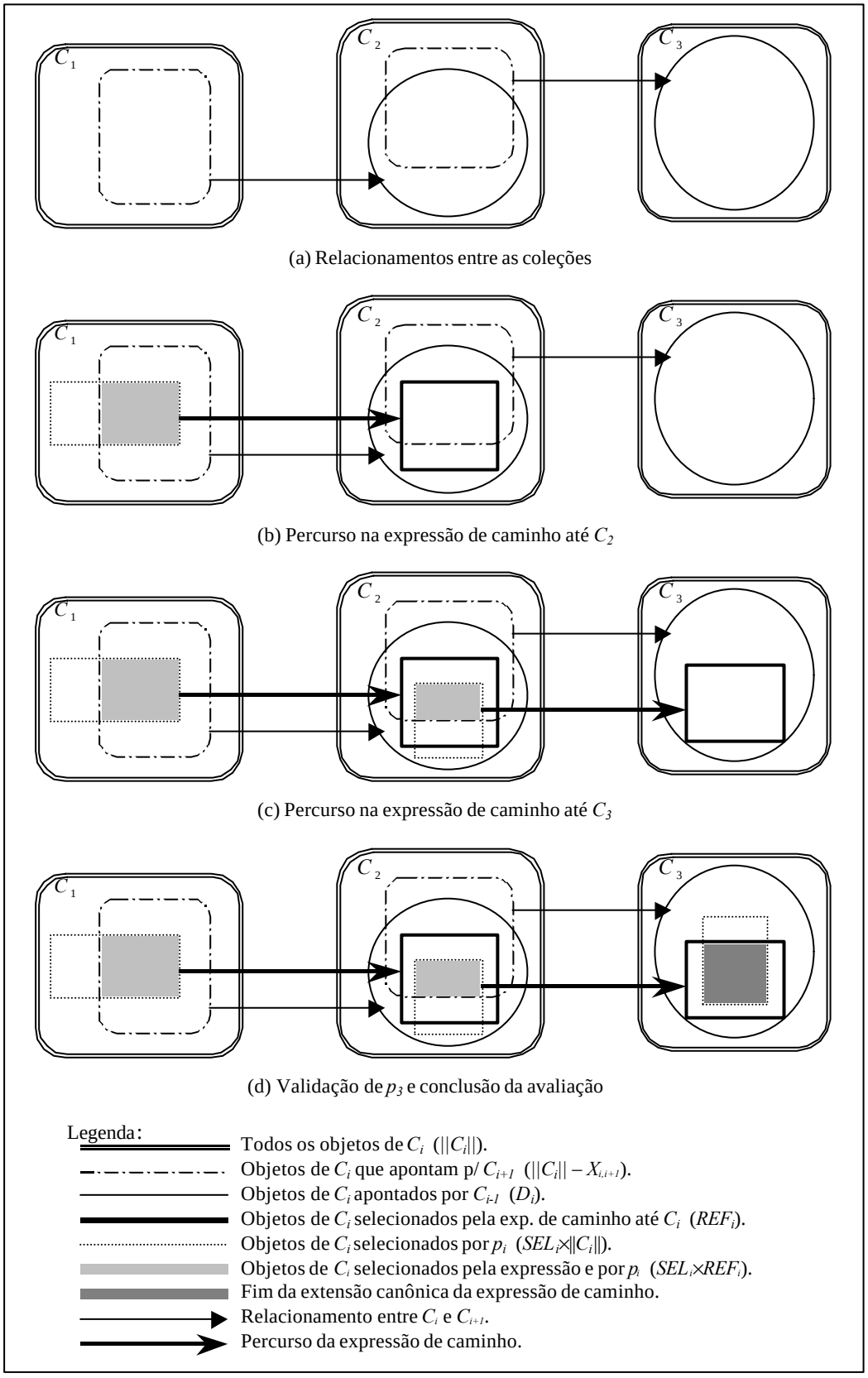


Figura 7. Seletividade de uma Expressão de Caminho

A seletividade da expressão de caminho até a coleção C_2 refere-se aos objetos que são apontados pelos objetos recuperados em C_1 e validados por p_1. Isso é aplicado recursivamente a C_3 (Figura 7.(c)) , onde o volume de objetos recuperados em uma coleção depende do grau de saída e do grau de compartilhamento no relacionamento desta com a coleção anterior na estratégia de avaliação. Por fim, é aplicado o predicado aninhado p_3 sobre os objetos recuperados em C_3 e temos resultado final, isto é, temos a extensão canônica da expressão de caminho (Figura 7.(d)). Para obter esta extensão canônica foi necessário percorrer alguns caminhos incompletos. Por exemplo, temos os caminhos que terminaram em objetos de C_2 porque não foram qualificados por p_2. Como foi realizada uma avaliação segundo a estratégia descendente, estes caminhos são referentes à extensão completa à esquerda da expressão de caminho (vide Seção 2.3.2).

4.3.1 COLEÇÕES SEM FRAGMENTAÇÃO

Em bases de objetos centralizadas, a estimativa do número de objetos distintos recuperados em uma coleção C_i $(1 \le i \le \ell)$ durante a avaliação de uma expressão de caminho é dada por:

$$REF_i = \lceil SEL'_i \times \|C_i\| \rceil , \tag{9}$$

onde SEL'_i é a seletividade da expressão de caminho até C_i, sendo calculada de acordo com a direção da estratégia de avaliação:

$$\Rightarrow \quad SEL'_1 = 1 \quad \text{e} \quad SEL'_i = \frac{SEL'_{i-1} \times SEL_{i-1} \times D_{i-1,i}}{\|C_i\|} \qquad \text{– Descendente ;} \tag{10}$$

$$\Rightarrow \quad SEL'_\ell = 1 \quad \text{e} \quad SEL'_i = \frac{SEL'_{i+1} \times SEL_{i+1} \times \left(\|C_i\| - X_{i,i+1}\right)}{\|C_i\|} \qquad \text{– Ascendente.} \tag{11}$$

Temos que i varia de 2 até ℓ, na avaliação descendente, e i varia de $\ell - 1$ até 1, na ascendente. Observe que todos os objetos da coleção de partida (C_1 ou C_ℓ, de acordo com a direção da estratégia) serão recuperados porque, como o nome diz (partida), não há filtro de relacionamento apontando para esta coleção na expressão de caminho.

É importante ressaltar que o número de objetos selecionados por uma expressão de caminho em cada coleção envolvida é geralmente calculado por um método probabilístico. Este método resulta em desvios expressivos dos valores esperados quando as expressões de caminho envolvem coleções volumosas e baixos fatores de seletividade. Por exemplo, seja a expressão de caminho contida na consulta da Figura 8. Considere a descrição padrão do *benchmark* OO7 (CAREY *et al*., 1993), na configuração média (*CompositeParts*, *AtomicParts* e *Connections* com respectivamente 500, 100000 e 300000 objetos), onde a participação das coleções nos relacionamentos da expressão de caminho abaixo é total.

```
select t
from c in CompositeParts, a in c.parts, t in a.to
```

```
EC: c.parts-a.to-t
```

Figura 8. Exemplo de consulta contendo uma expressão de caminho EC

De acordo com BELLATRECHE *et al*. (1998a, 1998b) e GARDARIN *et al*. (1995, 1996), que usam uma estimativa baseada em probabilidade, o número de objetos recuperados na coleção *Connections* durante a avaliação descendente desta expressão de caminho é calculado por:

$$REF_3 = \left(1-\left(1-\frac{1}{\|C_3\|}\right)^{\|C_2\|\times SEL_2\times fan_{2,3}}\right)\times\|C_3\| = \left(1-\left(1-\frac{1}{300000}\right)^{100000\times 1\times 3}\right)\times 300000 \cong 189637 \ .$$

Como não há predicados aninhados e estamos considerando que a participação das coleções nos relacionamentos da expressão de caminho é total, o valor esperado é 300000 objetos. Utilizando nosso modelo de custo, o número de objetos da coleção *Connections* recuperados na estratégia descendente é obtido pelas fórmulas (9) e (10). Ou seja:

$$SEL_3 = \frac{1\times 1\times 300000}{300000} = 1 \qquad \text{e} \qquad REF_3 = \lceil 1\times 300000 \rceil = 300000 \ .$$

Esta diferença, de aproximadamente 37%, se refletiria no número de páginas acessadas na coleção e nos demais custos derivados deste parâmetro.

4.3.2 EFEITO DA FRAGMENTAÇÃO

As consultas e transações sobre uma base de objetos distribuídos manipulam uma unidade lógica de dados mais adequada, de acordo com as informações acessadas por essas operações, que permite reduzir o volume de objetos irrelevantes acessados durante a execução dos algoritmos (KOSSMANN, 2000). Por exemplo, a execução de uma junção entre as coleções A e B, estando A fragmentada horizontalmente e $A = A_1 \cup A_2$, pode ser realizada das seguintes maneiras:

$$(A_1 \cup A_2) \infty B \qquad \text{ou} \qquad (A_1 \infty B) \cup (A_2 \infty B).$$

Em certos casos, a função de fragmentação de A está definida de tal forma que é possível, pelo predicado de seleção da consulta submetida, identificar algum termo $(A_i \infty B)$ cujo resultado seja um conjunto vazio. A eliminação de fragmentos irrelevantes de uma coleção ocorre durante a etapa de localização de dados no processamento de consultas em bases distribuídas (OZSÜ e VALDURIEZ, 1999), através da aplicação de técnicas de redução. Nesta etapa, uma consulta global, expressa sobre uma visão centralizada da base de objetos, é transformada em uma consulta sobre fragmentos físicos e reduzida a uma versão otimizada, livre de operações cujo resultado seja vazio.

4.3.2.1 ELIMINAÇÃO DE FRAGMENTOS NA EXPRESSÃO DE CAMINHO

Seja a coleção C_i $(1 \le i \le \ell)$ de uma dada expressão de caminho. A seletividade do predicado aninhado p_i em F_j^i é nula se este fragmento for eliminado na redução da expressão de caminho. Ou seja:

$$SEL_j^i = 0, \quad \text{caso:} \tag{12}$$

(i) o fragmento F_j^i seja horizontal e a seletividade de p_i em F_j^i seja nula; ou

(ii) o fragmento F_j^i seja vertical e nenhum de seus atributos é utilizado na navegação, nem está presente no predicado aninhado p_i.

Assim, quando C_i estiver fragmentada, serão acessados somente os fragmentos desta coleção onde a $SEL_j^i \neq 0$. Além disso, caso a função de fragmentação de C_i seja horizontal derivada, alguns de seus fragmentos poderão ser eliminados pela seletividade da expressão de caminho até C_i, conforme veremos a seguir.

Basicamente, caso alguma coleção C_i da expressão de caminho esteja fragmentada, o conjunto (possivelmente vazio) de todos os fragmentos de C_i que <u>não</u> serão acessados durante o processamento da consulta é definido como:

$$E_i = Eliminados_i \cup Eliminados'_i \ , \tag{13}$$

onde o fator de seletividade relativo aos objetos de C_i pertencentes aos fragmentos horizontais eliminados, quando for o caso, é:

$$selE_i = \sum_{F_j^i \in E_i} sel_j^i \ . \tag{14}$$

O subconjunto $Eliminados_i$ reúne os fragmentos de C_i eliminados por SEL_j^i, tal que:

$$Eliminados_i = \left\{F_j^i, 1 \leq j \leq f_i \mid SEL_j^i = 0\right\} . \tag{15}$$

Por outro lado, o subconjunto $Eliminados'_i$ é referente aos fragmentos horizontais de C_i, <u>cuja fragmentação é do tipo derivada</u> (quando for o caso), que foram eliminados pela seletividade da expressão de caminho. Note que a definição deste subconjunto depende da direção da estratégia de avaliação, tal que:

- $Eliminados'_i = \left\{F_j^i, 1 \leq j \leq f_i \mid \left(F_j^{i-1} \mapsto F_j^i\right) \wedge \left(F_j^{i-1} \in E_{i-1}\right)\right\}$ - Descendente; (16)

- $Eliminados'_i = \left\{F_j^i, 1 \leq j \leq f_i \mid \left(F_j^{i+1} \mapsto F_j^i\right) \wedge \left(F_j^{i+1} \in E_{i+1}\right)\right\}$ - Ascendente. (17)

O termo $F_j^y \mapsto F_j^i$ significa que o fragmento F_j^y determina o fragmento derivado F_j^i na função de fragmentação de C_i.

4.3.2.2 SELETIVIDADE DA EXP. DE CAMINHO NA FRAGMENTAÇÃO HORIZONTAL

Durante a avaliação de uma expressão de caminho de tamanho ℓ, o número de objetos distintos recuperados em uma coleção C_i fragmentada horizontalmente ($1 \le i \le \ell$) é dado por:

$$REF_i = \sum_{j=1}^{f_i} ref_j^i \ , \tag{18}$$

$$\text{onde} \quad ref_j^i = \left| SEL''_j \times \left\| F_j^i \right\| \right| . \tag{19}$$

Entretanto, quando um fragmento é eliminado da avaliação, temos que:

$$ref_j^i = 0 , \qquad \text{se } F_j^i \in E_i \ . \tag{20}$$

Por outro lado, de maneira análoga ao cálculo de SEL'_i, temos que SEL'^i_j é a seletividade da expressão de caminho sobre o fragmento F_j^i e sua estimativa depende da direção da estratégia de avaliação. Deste modo, de acordo com o tipo da fragmentação horizontal de C_i, teremos:

<u>FRAGMENTAÇÃO HORIZONTAL PRIMÁRIA</u>

$$\Rightarrow SEL'^1_j = 1 \quad \text{e} \quad SEL''^i_j = \frac{SEL'_{i-1} \times SEL_{i-1} \times D^i_{i-1,j}}{\left\| F_j^i \right\|} \qquad \text{- Desc.;} \tag{21}$$

$$\Rightarrow SEL'^\ell_j = 1 \quad \text{e} \quad SEL''^i_j = \frac{SEL'_{i+1} \times SEL_{i+1} \times \left(\left\| F_j^i \right\| - X^i_{j,i+1} \right)}{\left\| F_j^i \right\|} \qquad \text{- Asc. .} \tag{22}$$

<u>FRAGMENTAÇÃO HORIZONTAL DERIVADA</u>

$$\Rightarrow SEL'^1_j = 1 \quad \text{e} \quad SEL''^i_j = \frac{part^i_j \left(SEL'_{i-1} \times SEL_{i-1} \right) \times D^i_{i-1,j}}{\left\| F_j^i \right\|} \qquad \text{- Desc.;} \tag{23}$$

$$\Rightarrow\ SEL'^{\ell}_{j} = 1 \quad \text{e} \quad SEL'^{i}_{j} = \frac{part^{i}_{j}\left(SEL'_{i+1} \times SEL_{i+1}\right) \times \left(\left\|F^{i}_{j}\right\| - X^{i}_{j,i+1}\right)}{\left\|F^{i}_{j}\right\|} \text{ - Asc. .} \tag{24}$$

A função $part^{i}_{j}(fator)$ retorna a participação do fragmento F^{i}_{j} nos objetos selecionados por $fator$ em C_i. Este tratamento é necessário na fragmentação horizontal derivada de C_i quando algum fragmento F^{i}_{j} for eliminado pela seletividade da expressão de caminho até C_i. Isto ocorre porque, neste caso, os fragmentos de C_i não eliminados na avaliação terão uma participação maior sobre os objetos de C_i para que seja satisfeita a seletividade de $fator$. Esta função é definida da seguinte maneira:

$$part^{i}_{j}(fator) = \begin{cases} 1 & \text{, se } fator = 1, \\ fator + \dfrac{selEliminados'_{i} \times fator}{(1 - selEliminados'_{i})} & \text{, caso contrário.} \end{cases} \tag{25}$$

O percentual de objetos referentes aos fragmentos de C_i eliminados pela seletividade da expressão de caminho, $selEliminados'_{i}$, é obtido analogamente à equação (14) (Seção 4.3.2.1). Por fim, a seletividade da expressão de caminho e a seletividade do predicado aninhado p_i sobre uma coleção C_i fragmentada horizontalmente correspondem, respectivamente, a:

$$\Rightarrow\ SEL'_{1} = 1 \text{ p/ Desc., ou } SEL'_{\ell} = 1 \text{ para Asc., e } SEL'_{i} = \sum_{j=1}^{f_i} \left(sel^{i}_{j} \times SEL'^{i}_{j}\right); \tag{26}$$

$$\Rightarrow\ SEL_{i} = \sum_{j=1}^{f_i} \left(sel^{i}_{j} \times SEL^{i}_{j}\right). \tag{27}$$

As fórmulas (26) e (27) consideram todos os fragmentos, eliminados ou não.

4.3.2.3 SELETIVIDADE DA EXP. DE CAMINHO NA FRAGMENTAÇÃO VERTICAL

Um objeto da visão global de uma coleção fragmentada verticalmente é representado por vários objetos (com mesma identidade ou não), distribuídos nos respectivos fragmentos verticais. Quando a coleção C_i $(1 \le i < \ell)$ participa da avaliação de uma expressão de caminho, apenas um fragmento vertical de C_i conterá o atributo

de referência A_i envolvido no percurso. Os demais fragmentos serão acessados somente se forem necessários para satisfazer o predicado aninhado p_i.

Por outro lado, os fragmentos verticais de uma coleção C_i possuem a mesma cardinalidade, a qual corresponde à cardinalidade de C_i considerando os objetos distintos. Além disso, todos os fragmentos verticais de C_i que não forem eliminados pela condição (12) (Seção 4.3.2.1) sofrerão uma mesma seletividade da expressão de caminho até C_i. Portanto, o número total de objetos distintos recuperados a partir da coleção C_i, fragmentada verticalmente, durante a avaliação de uma expressão de caminho é calculado por:

$$REF_i = ref_*^i \ , \tag{28}$$

onde o termo ref_*^i representa o número de objetos distintos de C_i recuperados nos fragmentos verticais durante a avaliação da expressão de caminho, tal que:

$$ref_*^i = \left\lceil SEL'_i \times \| C_i \| \right\rceil . \tag{29}$$

Observe que SEL'_i é a seletividade da expressão de caminho até C_i, sendo calculada de acordo com as fórmulas (10) e (11). Contudo, um objeto global de uma coleção fragmentada verticalmente é representado por vários objetos, com mesmo IDO ou não. Logo, o total de objetos envolvidos na avaliação de uma expressão de caminho em todos os fragmentos verticais não eliminados é:

$$REF\_v_i = \left(f_i - \#E_i \right) \times ref_*^i \ . \tag{30}$$

Quanto ao predicado aninhado p_i, a sua validação poderá surtir fatores de seletividade distintos nos fragmentos verticais de C_i. Assim, teremos que:

$$SEL_i = \min_{F_j^i \notin E_i} \left(SEL_j^i \right) \ . \tag{31}$$

4.4 MÚLTIPLAS REFERÊNCIAS IDÊNTICAS

De acordo com o grau de compartilhamento da coleção C_i em uma expressão de caminho, poderão existir referências replicadas para os objetos de C_i. Estas réplicas irão impactar no número de páginas de C_i que serão recarregadas quando são utilizados algoritmos baseados em ponteiros e a memória disponível não é suficiente para conter todos os objetos de C_i recuperados na avaliação da expressão de caminho.

4.4.1 COLEÇÕES SEM FRAGMENTAÇÃO

A estimativa do número total de referências para os objetos de uma coleção C_i na avaliação de uma expressão de caminho, incluindo as réplicas, depende da direção da estratégia de avaliação, sendo obtida pelas seguintes fórmulas:

$$\Rightarrow \wedge REF_1 = \|C_1\| \quad \text{e} \wedge REF_i = \begin{cases} \|C_i\| \text{ , se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Desc. ;} \\ \quad \lceil REF_{i-1} \times SEL_{i-1} \times fan_{i-1,i} \rceil . \end{cases} \tag{32}$$

$$\Rightarrow \wedge REF_\ell = \|C_\ell\| \quad \text{e} \wedge REF_i = \begin{cases} \|C_\ell\| \text{ , se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Asc. .} \\ \quad \lceil REF_{i+1} \times SEL_{i+1} \times fan_{i+1,i} \rceil . \end{cases} \tag{33}$$

Note que, caso não existam atributos de relacionamento inverso definidos nas coleções da expressão de caminho, a estratégia ascendente só poderá ser efetuada através de junções baseadas em valor.

4.4.2 EFEITO DA FRAGMENTAÇÃO HORIZONTAL

De maneira análoga, no caso de coleções fragmentadas horizontalmente, teremos para cada fragmento não eliminado:

4.4.2.1 FRAGMENTAÇÃO HORIZONTAL PRIMÁRIA

$$\Rightarrow \wedge ref_j^1 = \left\| F_j^1 \right\| \quad \text{e} \wedge ref_j^i = \begin{cases} \left\| F_j^i \right\| \text{, se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Desc. ;} \\ \left\lceil REF_{i-1} \times SEL_{i-1} \times fan_{i-1,j}^i \times sel_j^i \right\rceil . \end{cases} \tag{34}$$

$$\Rightarrow \wedge ref_j^\ell = \left\| F_j^\ell \right\| \quad \text{e} \wedge ref_j^i = \begin{cases} \left\| F_j^i \right\| \text{, se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Asc. .} \\ \left\lceil REF_{i+1} \times SEL_{i+1} \times fan_{i+1,j}^i \times sel_j^i \right\rceil . \end{cases} \tag{35}$$

4.4.2.2 FRAGMENTAÇÃO HORIZONTAL DERIVADA

$$\Rightarrow \wedge ref_j^1 = \left\| F_j^1 \right\| \text{ e} \wedge ref_j^i = \begin{cases} \left\| F_j^i \right\| \text{, se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Desc. ;} \\ \left\lceil REF_{i-1} \times part_j^i (SEL_{i-1}) \times fan_{i-1,j}^i \times sel_j^i \right\rceil \end{cases} \tag{36}$$

$$\Rightarrow \wedge ref_j^\ell = \left\| F_j^\ell \right\| \text{ e} \wedge ref_j^i = \begin{cases} \left\| F_j^i \right\| \text{, se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Asc. .} \\ \left\lceil REF_{i+1} \times part_j^i (SEL_{i+1}) \times fan_{i+1,j}^i \times sel_j^i \right\rceil \end{cases} \tag{37}$$

Deste modo, o número total de referências para objetos da coleção C_i, incluindo as réplicas, será dado por:

$$\wedge REF_i = \sum_{j=1}^{f_i} \wedge ref_j^i \ . \tag{38}$$

4.4.3 EFEITO DA FRAGMENTAÇÃO VERTICAL

Neste caso, o número total de referências para objetos distintos da coleção C_i é obtido por:

$$\wedge REF_i = \wedge ref_*^i \ , \tag{39}$$

tal que $\wedge ref_*^i$ é calculado pelas seguintes fórmulas:

$$\Rightarrow \wedge ref_*^1 = \|C_1\| \quad \text{e} \wedge ref_j^i = \begin{cases} \|C_i\| \text{ , se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Desc. ;} \\ \lceil REF_{i-1} \times SEL_{i-1} \times fan_{i-1,i} \rceil \end{cases} \tag{40}$$

$$\Rightarrow \wedge ref_*^\ell = \|C_\ell\| \quad \text{e} \wedge ref_j^i = \begin{cases} \|C_i\| \text{, se é usada junção por valor.} \\ \text{Caso contrário,} \qquad \text{- Asc. .} \\ \lceil REF_{i+1} \times SEL_{i+1} \times fan_{i+1,i} \rceil \end{cases} \tag{41}$$

Entretanto, um objeto global de uma coleção fragmentada verticalmente é representado por vários objetos, conforme as considerações da Seção 4.3.2.3. Logo, o total de ponteiros envolvidos na avaliação de uma expressão de caminho considerando todos os fragmentos verticais acessados é:

$$\wedge REF\_v_i = (f_i - \#E_i) \times \wedge ref_*^i \ . \tag{42}$$

4.5 PÁGINAS OCUPADAS PELAS COLEÇÕES

O ponto de partida para o cálculo do custo das operações de E/S é a estimativa do número de páginas ocupadas pelas coleções da expressão de caminho (BERTINO e FOSCOLI, 1997). Este parâmetro não reflete exatamente o volume de *bytes* ocupados pela coleção, devido às diferentes possibilidades de armazenamento físico da mesma. Entretanto, ele influencia diretamente o cálculo do número de páginas acessadas em uma consulta.

Inicialmente, seja uma coleção C_i, cujos objetos pertencem à classe T_i e estão agrupados fisicamente pela extensão de classe. Assim, existe apenas um tipo de partição física para armazenar C_i, a qual contém somente objetos de C_i. O número de páginas ocupadas por esta coleção é:

$$|C_i| = \left\lceil \frac{\|C_i\|}{\left\lfloor S_P / S_{C_i} \right\rfloor} \right\rceil . \tag{43}$$

O termo divisor de (43) é dito o fator de bloco da coleção C_i (ELMASRI e NAVATHE, 1994) e corresponde ao número de objetos de C_i armazenados em uma página do banco de dados.

4.5.1 AGRUPAMENTO FÍSICO POR REFERÊNCIA

Agora, sejam C_A e C_B coleções agrupadas fisicamente através de um relacionamento (ambas não fragmentadas), onde C_A é a raiz do agrupamento. Ou seja, os objetos de C_B apontados por um objeto de C_A serão armazenados fisicamente logo após este objeto. Neste caso, o agrupamento por referência destas coleções será armazenado em três tipos de partições físicas:

- A partição Cl_A armazenará somente objetos de C_A que não possuem referências para objetos da classe C_B;
- A partição Cl_B armazenará somente objetos de C_B; e
- Por fim, temos a partição $Cl_{A \to B}$, que agrupa objetos pertencentes à C_A e à C_B, armazenados juntos conforme o relacionamento.

Observe que a partição Cl_B possui ainda duas subpartições: Cl_B^*, a qual reúne os objetos de C_B que não possuem referências a partir de objetos de C_A; e Cl_B^{**}, a subpartição de Cl_B referente aos objetos de C_B que são armazenados fora da página onde seu respectivo objeto de C_A está armazenado (página-raiz do agrupamento).

O objetivo de um agrupamento por referência é minimizar o número de páginas acessadas na recuperação dos objetos de uma classe e de seus objetos relacionados. Para isso, objetos relacionados são armazenados com proximidade física no disco. No melhor caso, todos os objetos referentes a uma unidade do agrupamento caberão em uma página de dados. Caso contrário, ao recuperar um objeto de C_A, por exemplo, apenas os objetos de C_B contidos na página-raiz serão carregados automaticamente. Serão necessárias operações de E/S adicionais para recuperar os demais objetos relacionados. Observe que pelo menos um objeto de cada coleção envolvida no agrupamento deve caber na página-raiz, senão o agrupamento físico entre C_A e C_B não representará uma boa escolha. Isto é, a seguinte condição deve ser respeitada:

$$\left(S_{C_A} + S_{C_B}\right) \leq S_P \ . \tag{44}$$

O número de páginas ocupadas pela coleção C_A será dado por:

$$\left|C_A\right| = \left|C_A\right|_{Cl_A} + \left|C_A\right|_{Cl_{A\to B}} \ , \tag{45}$$

onde, de acordo com (GARDARIN et al., 1995):

$$\left|C_A\right|_{Cl_A} = \left\lceil X_{A,B} \Big/ \left\lfloor S_P / S_{C_A} \right\rfloor \right\rceil \tag{46}$$

e

$$\left|C_A\right|_{Cl_{A\to B}} = \begin{cases} \left\lceil \dfrac{\left\|C_A\right\| - X_{A,B}}{\left\lfloor \dfrac{S_P}{S_{Cl_{A\to B}}} \right\rfloor} \right\rceil & se\ S_{Cl_{A\to B}} < S_P \\ \left\|C_A\right\| - X_{A,B} & caso\ contrário \end{cases} \ , \tag{47}$$

onde $S_{Cl_{A\to B}}$ é o tamanho de uma unidade do agrupamento, ou seja,

$$S_{Cl_{A\to B}} = S_{C_A} + \left(S_{C_B} \times Z_{A,B}\right) . \tag{48}$$

Por outro lado, a estimativa do número de páginas ocupadas pela coleção C_B requer um tratamento especial, pois o tamanho final de uma unidade do agrupamento pode ser maior que uma página do banco de dados. Consideramos que a política de alocação física dos objetos deve respeitar o objetivo de minimizar o número de páginas ocupadas por cada unidade do agrupamento, mesmo que algum espaço seja desperdiçado nas páginas de dados (ELMASRI e NAVATHE, 1994). Por essa razão, um certo número de objetos de C_B será armazenado fora de suas respectivas páginas-raízes, em páginas destinadas a conter somente os objetos de C_B que "sobraram" do agrupamento. Logo, teremos uma subpartição "pura" (Cl_B^*) de objetos de C_B que são

referenciados por C_A, cujo fator de bloco é igual ao da partição Cl_B^{**}. GARDARIN *et al*. (1995) ignoram esta política.

Deste modo, o número de páginas ocupadas pela coleção C_B será:

$$|C_B| = |C_B|_{Cl_B} + |C_B|_{Cl_{A\to B}} \ . \tag{49}$$

A partição composta da coleção C_B ocupa o mesmo número de páginas da partição composta de C_A. Ou seja:

$$|C_B|_{Cl_{A\to B}} = |C_A|_{Cl_{A\to B}} \ . \tag{50}$$

Para a partição que armazena somente objetos de C_B, nós temos:

$$|C_B|_{Cl_B} = |C_B|_{Cl_B^*} + |C_B|_{Cl_B^{**}} \ . \tag{51}$$

Os objetos de C_B que não participam do relacionamento com C_A ocuparão:

$$|C_B|_{Cl_B^*} = \left\lceil \frac{\|C_B\| - D_{A,B}}{\left\lfloor S_P / S_{C_B} \right\rfloor} \right\rceil . \tag{52}$$

Se $S_{Cl_{A\to B}} \le S_P$, a página-raiz do agrupamento armazenará todos os objetos de C_B que são referenciados por objetos de C_A. Caso contrário, o número médio de objetos de C_B que serão armazenados fora de sua página-raiz, em cada unidade do agrupamento, será:

$$O_{A,B} = Z_{A,B} - \left\lfloor \frac{S_P - S_{C_A}}{S_{C_B}} \right\rfloor . \tag{53}$$

Quando os $O_{A,B}$ objetos não cabem em uma página do banco de dados, uma parte destes objetos ($O^{*}_{A,B}$) sobrará após o preenchimento de quantas páginas inteiras forem possíveis. Esses $O^{*}_{A,B}$ certamente caberão em uma página, sendo calculados por:

$$O^{*}_{A,B} = O_{A,B} - \left\lfloor O_{A,B} \Big/ \left\lfloor S_P \Big/ S_{C_B} \right\rfloor \right\rfloor \times \left\lfloor S_P \Big/ S_{C_B} \right\rfloor . \tag{54}$$

Finalmente, o número de páginas ocupadas pela coleção C_B na partição Cl^{**}_B é:

$$\left|C_B\right|_{Cl^{**}_B} = \begin{cases} \left\lceil \dfrac{\|C_A\| - X_{A,B}}{\left\lfloor \dfrac{S_P}{O_{A,B} \times S_{C_B}} \right\rfloor} \right\rceil & \text{se } \left(O_{A,B} \times S_{C_B}\right) \le S_P. \\ \text{Caso contrário,} & \\ \left(\|C_A\| - X_{A,B}\right) \times \left\lfloor \dfrac{O_{A,B}}{\left\lfloor S_P \Big/ S_{C_B} \right\rfloor} \right\rfloor & \left\{ + \left\lceil \dfrac{\|C_A\| - X_{A,B}}{\left\lfloor \dfrac{S_P}{O^{*}_{A,B} \times S_{C_B}} \right\rfloor} \right\rceil \quad \text{se } O^{*}_{A,B} \neq 0 \right\} \end{cases} \tag{55}$$

Quando tratamos de agrupamentos aninhados em uma expressão de caminho (do tipo onde C_A agrupa C_Z, tal que C_Z é um agrupamento onde C_B agrupa C_C), devemos considerar que $S_{C_Z} = S_{Cl_{B \to C}}$. Vale ressaltar que dificilmente ocorrerá $S_{C_Z} > S_P$, pois neste caso o agrupamento provavelmente não será vantajoso, já que a condição (44) não é respeitada. Nos agrupamentos físicos envolvendo mais de duas coleções, a complexidade das fórmulas irá crescer proporcionalmente ao número de coleções envolvidas, estando este tema fora do escopo deste trabalho.

4.5.2 ESTIMATIVA EM COLEÇÕES FRAGMENTADAS

O número total de páginas ocupadas por uma coleção fragmentada C_i, considerando que os objetos de C_i estão agrupados pela extensão de classe, é dado por:

$$\left|C_i\right| = \sum_{j=1}^{f_i} \left|F_j^i\right| , \tag{56}$$

$$\text{onde} \quad \left|F_j^i\right| = \left\lceil \left\| F_j^i \right\| \Big/ \left\lfloor S_P \Big/ S_{F_j^i} \right\rfloor \right\rceil . \tag{57}$$

Os tamanhos médios de objeto ($S_{F_j^i}$) dos fragmentos horizontais de uma coleção C_i são praticamente idênticos. Assim, o que diferencia o número de páginas ocupadas em cada fragmento horizontal é a cardinalidade de F_j^i. O contrário acontece na fragmentação vertical, onde todos os fragmentos têm a mesma cardinalidade, mas podem possuir tamanhos médios de objeto distintos entre si.

A aplicação de técnicas de agrupamento físico em coleções fragmentadas deve respeitar alguns critérios e restrições. Primeiramente, consideramos que o agrupamento por referência só poderá gerar uma fragmentação adicional sobre as coleções envolvidas se esta for realizada implícita e automaticamente. Ou seja, não dever ser necessária qualquer interferência no projeto de fragmentação dos objetos. Na Tabela 3 são mostradas as combinações entre técnicas básicas de fragmentação e o agrupamento físico por referência. Em um agrupamento do tipo $Cl_{A \to B}$, entre as coleções C_A e C_B, são representados os seguintes tipos de fragmentação sobre as coleções: $F(i)$ – para qualquer tipo de fragmentação; $H(i)$ – para a fragmentação horizontal, primária ou derivada; $P(i)$ para a fragmentação horizontal primária; $D(i)$ para a fragmentação horizontal derivada; e $V(i)$ para a fragmentação vertical. Assumimos que não há replicação de objetos após o agrupamento físico, ou seja, $share_{A,B} = 1$.

Tabela 3. Técnicas de Fragmentação ´ Agrupamento por Referência

COMBINAÇÃO	POSSÍVEL?	CONSEQÜÊNCIA	PRÉ-CONDIÇÃO
$A \rightarrow V(B)$	Não	–	–
$A \rightarrow H(B)$	Sim	A sofre uma fragmentação horizontal automática, derivada em relação à fragmentação de C_B.	$fan_{A,B} = 1$.
$F(A) \rightarrow V(B)$	Não	–	–
$V(A) \rightarrow B$	Sim	Somente o fragmento vertical de C_A que contém o atributo de relacionamento participa do agrupamento com B.	–
$H(A) \rightarrow B$	Sim	Ocorre uma fragmentação horizontal automática de C_B, derivada em relação à C_A, onde cada fragmento de C_B será agrupado com seu respectivo fragmento primário em C_A.	–
$V(A) \rightarrow H(B)$	Sim	Somente um fragmento de C_A participa do agrupamento. Este fragmento vertical sofre uma fragmentação automática, derivada em relação à C_B. Deste modo, são gerados fragmentos híbridos em C_A.	$fan_{A,B} = 1$.
$P(A) \rightarrow P(B)$	Não	–	–
$D(A) \rightarrow P(B)$	Sim	Cada fragmento de C_A será agrupado com seu respectivo fragmento primário em C_B.	A fragmentação de C_A é derivada em relação a C_B e $fan_{A,B} = 1$.
$H(A) \rightarrow D(B)$	Sim	Cada fragmento de C_B será agrupado com seu respectivo fragmento primário em C_A.	A fragmentação de C_B é derivada em relação a C_A.

O casos $A \rightarrow V(B)$ e $F(A) \rightarrow V(B)$ da Tabela 3 não são considerados possíveis porque o agrupamento anularia a fragmentação vertical. Já $P(A) \rightarrow P(B)$ não é tido como possível porque demandaria modificações no projeto de fragmentação dos objetos. A combinação $H(A) \rightarrow D(B)$ só é possível se a fragmentação horizontal de C_B for

derivada em relação a C_A. De maneira análoga, $D(A) \rightarrow P(B)$ só ocorre se a fragmentação de C_A for derivada em relação à coleção C_B e se $fan_{A,B} = 1$. Os casos $A \rightarrow H(B)$ e $V(A) \rightarrow H(B)$ também requerem que $fan_{A,B} = 1$. Quando a combinação é considerada possível, analisamos na Tabela 3 as conseqüências da mesma quanto ao armazenamento físico das coleções. Pelo exposto, uma coleção fragmentada verticalmente só poderá participar de um agrupamento físico por referência se for a raiz do agrupamento. Do mesmo modo, se $fan_{A,B} \neq 1$, uma coleção cuja fragmentação é horizontal primária só poderá representar a raiz de um agrupamento.

O número de páginas ocupadas por cada fragmento nas combinações possíveis é obtido de maneira semelhante ao cálculo no agrupamento de coleções não fragmentadas, substituindo-se as coleções pelos respectivos fragmentos. Nos casos $H(A) \rightarrow B$ e $H(A) \rightarrow D(B)$, cada fragmento horizontal F_j^B de C_B é armazenado com seu respectivo fragmento horizontal primário F_j^A de C_A, formando um agrupamento físico do tipo $Cl_j^{A,B}$. Note que em $H(A) \rightarrow B$ a fragmentação horizontal de C_B é realizada automaticamente pelo agrupamento físico dos objetos. Analogamente, após o agrupamento, em $A \rightarrow H(B)$, $D(A) \rightarrow P(B)$ e $V(A) \rightarrow H(B)$ cada fragmento horizontal F_j^A de C_A é armazenado com seu respectivo fragmento horizontal primário F_j^B de C_B. Em particular, o caso $V(A) \rightarrow H(B)$ gera uma fragmentação híbrida da coleção C_A. Por fim, nas combinações $V(A) \rightarrow B$ e $V(A) \rightarrow H(B)$ o agrupamento físico será formado somente pelos fragmentos verticais que contêm o atributo de referência utilizado para agrupar as coleções.

Portanto, para cada fragmento F_j^i que participa de um agrupamento por referência, onde C_i é a raiz do agrupamento, o número de páginas ocupadas é:

$$\left|F_j^i\right| = \left|F_j^i\right|_{Cl_j^i} + \left|F_j^i\right|_{Cl_{j \rightarrow i+1}^i} , \tag{58}$$

onde:

$$\left|F_j^i\right|_{Cl_j^i} = \left\lceil \frac{X_{j,i+1}^i}{\left\lfloor S_P / S_{F_j^i} \right\rfloor} \right\rceil \tag{59}$$

e

$$\left|F_j^i\right|_{Cl_{j\to i+1}^i} = \begin{cases} \left\lceil \dfrac{\left\|F_j^i\right\| - X_{j,i+1}^i}{\left\lfloor S_P / S_{Cl_{j\to i+1}^i} \right\rfloor} \right\rceil & \text{se } S_{Cl_{j\to i+1}^i} < S_P, \\ \left\|F_j^i\right\| - X_{j,i+1}^i & \text{caso contrário.} \end{cases} \tag{60}$$

Análogo a $S_{Cl_{A\to B}}$, o tamanho de uma unidade do agrupamento $Cl_{j\to i+1}^i$ é:

$$S_{Cl_{j\to i+1}^i} = S_{C_{F_j^i}} + \left(S_{C_{i+1}} \times Z_{j,i+1}^i\right). \tag{61}$$

Conforme dito, em um agrupamento do tipo $Cl_{j\to i+1}^i$, se a coleção C_{i+1} está fragmentada, deve-se utilizar nas fórmulas (59), (60) e (61) os parâmetros referentes ao fragmento de C_{i+1} que está agrupado com F_j^i.

Pela Tabela 3, em agrupamentos físicos do tipo $Cl_{i\to j}^{i+1}$, o tipo de fragmentação da coleção C_{i+1} será sempre horizontal, primária ou derivada. Neste caso, temos pares de agrupamento entre os fragmentos envolvidos, cada par do tipo $Cl_j^{i,i+1}$, reunindo um fragmento primário e um derivado. Não existe uma partição física de objetos que não são referenciados em cada fragmento derivado. Ao invés disso, existe um fragmento não agrupado, definido pela cláusula *ELSE*, cujo número de páginas ocupadas é dado pela fórmula (57). O número de páginas ocupadas por um fragmento horizontal F_j^{i+1} é obtido de maneira análoga a (49), (50) e (55).

4.6 PÁGINAS ACESSADAS ALEATORIAMENTE

Quando uma coleção é percorrida seqüencialmente, a estimativa de páginas acessadas é diretamente proporcional ao número de objetos recuperados a partir do disco. Porém, na travessia por ponteiros, própria da avaliação de expressões de caminho, o acesso ao disco é geralmente aleatório e esta estimativa requer um tratamento mais específico. Uma vez calculados o número de objetos recuperados em cada coleção e o número de páginas ocupadas pelas coleções na avaliação de uma expressão de caminho, é possível estimar quantas páginas serão acessadas aleatoriamente na consulta.

Nós utilizamos a fórmula proposta em YAO (1977) (vide Anexo I), a qual representa uma referência clássica para estimar o número de páginas acessadas quando REF_i objetos distintos são recuperados aleatoriamente a partir de uma coleção C_i, tal que este número correspondente a:

$$Yao\left(\|C_i\|, |C_i|, REF_i\right) . \tag{62}$$

Entretanto, se alguma coleção C_i da expressão de caminho estiver agrupada por referência com outra coleção, esta fórmula não pode ser aplicada diretamente. Isso ocorre porque as diferentes partições físicas da coleção possuem densidades distintas (fator de bloco). Para solucionar este problema, utilizaremos a fórmula *Yao'*, proposta por GARDARIN *et al.* (1995). Assim, seja a coleção C_i pertencente a uma expressão de caminho, a qual possui p_i partições físicas, cada partição com $\|C_i\|_k$ objetos. Se REF_i objetos de C_i ($REF_i \leq \|C_i\|$) são selecionados aleatoriamente a partir do disco, o número esperado de páginas acessadas é dado por:

$$Yao'\left(C_i, REF_i\right) = \sum_{k=1}^{p_i} Yao\left(\|C_i\|_k, |C_i|_k, REF_{i,k}\right) , \tag{63}$$

onde $REF_{i,k}$ corresponde ao número de objetos selecionados na partição k de C_i, sendo obtido de acordo com a seletividade da partição k em relação à C_i.

4.6.1 ANÁLISE DAS PARTIÇÕES FÍSICAS ENVOLVIDAS

Caso exista agrupamento físico entre as classes C_{i-1} e C_i, durante a avaliação de uma expressão de caminho, para carregar C_i é necessário acessar apenas as páginas referentes aos objetos de C_i armazenados fora de sua respectiva página-raiz. Isso ocorre porque, ao carregar uma página de C_{i-1}, os objetos de C_i que estiverem nesta página serão automaticamente carregados na memória. Logo, o número esperado de páginas acessadas quando REF_i objetos são recuperados na coleção C_i em uma estratégia de avaliação descendente é:

$$\#\, pageHits_i(REF_i) = \begin{cases} Yao'(\|C_i\|, REF_i) & \text{se } cl_{i\text{-}1,i} = FALSO. \\ \text{Caso contrário,} & \\ \quad 0 & \text{se } O_{i\text{-}1,i} = 0, \\ \quad Yao(\|C_i\|_{Cl_i^{**}}, |C_i|_{Cl_i^{**}}, REF_{i,Cl_i^{**}}) & \text{se } O_{i\text{-}1,i} > 0. \end{cases} \tag{64}$$

$$\text{onde} \quad \|C_i\|_{Cl_i^{**}} = (\|C_{i-1}\| - X_{i-1,i}) \times O_{i-1,i} \tag{65}$$

$$\text{e} \qquad cl_{i-1,i} = \begin{cases} VERDADEIRO, & \text{se a coleção } C_i \text{ está agrupada com } C_{i-1}, \\ FALSO, & \text{caso contrário.} \end{cases} \tag{66}$$

Na estratégia ascendente, as fórmulas acima são definidas sobre C_i e C_{i+1}. Quando utilizarmos a função $\#pagesHits_i$ sem informar o parâmetro de entrada, estaremos considerando que este parâmetro corresponde a REF_i.

GARDARIN *et al*. (1995) não consideram o acesso à partição Cl_i^{**}. Todavia, de acordo com o grau de saída e o com o grau de compartilhamento entre as coleções C_{i-1} e C_i, esta partição pode representar um volume significativo de páginas. Por exemplo, suponha o agrupamento entre as coleções *CompositeParts* (C_c - raiz do agrupamento) e *AtomicParts* (C_a) na configuração média do *benchmark* OO7, considerando $S_{C_c} = 47$,

$S_{C_a} = 66$ e uma página de dados de tamanho $S_P = 2048$ *bytes*. Neste caso, nós estimamos que os objetos de *AtomicParts* que são apontados por objetos de *CompositeParts* e não couberam na página raiz do agrupamento ocupam $\left|C_a\right|_{Cl_a^{**}} = 2750$ páginas. Por GARDARIN *et al.* (1995), esta partição não existiria e o número de páginas acessadas na coleção C_a na avaliação descendente da expressão de caminho $EC : c.parts - a.to - t$ (definida na Figura 8) seria zero ($\#pageHits_a = 0$). Todavia, pelos nossos cálculos, teríamos $\#pageHits_a = 2750$, pois as páginas da partição Cl_a^{**} não são carregadas ao acessar a coleção C_c.

4.6.2 COLEÇÕES FRAGMENTADAS

O número esperado de páginas acessadas em um fragmento F_j^i quando ref_j^i objetos distintos são recuperados aleatoriamente é dado por uma extensão à função Yao', onde:

$$Yao'\left(\left\|F_j^i\right\|, ref_j^i\right) = \sum_{k=1}^{p_j^i} Yao\left(\left\|F_j^i\right\|_k, \left|F_j^i\right|_k, ref_{j,k}^i\right). \tag{67}$$

A partir deste número, podemos estimar o total de páginas acessadas em uma coleção fragmentada C_i, tal que:

$$\#pageHits_i = \sum_{F_j^i \notin E_i} \#pageHits_j^i\left(ref_j^i\right), \tag{68}$$

onde $\#pageHits_j^i$ na avaliação descendente de uma expressão de caminho é obtido por:

$$\#pageHits_j^i\left(ref_j^i\right) = \begin{cases} Yao'\left(\left\|F_j^i\right\|, ref_j^i\right) & \text{se } cl_{i-1,j}^i = 0. \text{ Caso contrário,} \\ 0 & \text{se } O_{i-1,j}^i = 0, \\ Yao\left(\left\|F_j^i\right\|_{Cl^{**i}_j}, \left|F_j^i\right|_{Cl^{**i}_j}, ref_{i,Cl^{**i}_j}\right) & \text{se } O_{i-1,j}^i > 0. \end{cases} \tag{69}$$

4.7 ANÁLISE DE CUSTO DOS OPERADORES

O custo final da árvore de processamento (AP) de uma expressão de caminho é obtido a partir do custo de cada operador que a compõe. Iremos analisar os algoritmos básicos dos operadores mais relevantes na avaliação de uma expressão de caminho. Para a análise das principais estratégias de avaliação, segundo a combinação de direção de avaliação e algoritmo utilizado, estamos considerando que as árvores de processamento são lineares (com apenas uma direção de avaliação) e homogêneas (com apenas um tipo de operador). Entretanto, a aplicação do modelo de custo em árvores densas e com diversos tipos de operadores pode ser composta a partir das funções existentes.

O custo das operações de E/S é baseado no número de páginas acessadas nas coleções envolvidas, considerando que a memória pode não ser suficiente para comportar todos as páginas solicitadas. Quanto ao custo de CPU, são analisadas as operações referentes à validação dos predicados aninhados, à travessia propriamente dita e à manutenção de estruturas intermediárias.

4.7.1 CUSTO DOS PREDICADOS ANINHADOS

Uma expressão de caminho de comprimento ℓ possui ℓ possíveis predicados aninhados. Quando definido, um predicado aninhado p_i é composto por $\#p_i$ predicados de seleção simples, tal que todos os objetos da coleção C_i recuperados na travessia serão validados por cada predicado simples de p_i. Todas as estratégias para avaliação de expressões de caminho que estamos considerando envolvem a validação de predicados aninhados, se estes forem especificados. Deste modo, para evitar redundância, iremos apresentar separadamente os custos envolvidos nessa validação.

Consideramos que, para a realização desta validação, as páginas de objetos de C_i já foram carregadas na memória durante a travessia da expressão de caminho. Portanto, o custo desta operação resume-se ao custo das instruções de CPU realizadas. Quando o predicado p_i não é definido na expressão de caminho, não há custo de validação do mesmo. Deste modo, o custo de validação de p_i em C_i é:

$$custoCPU\_p_i = \begin{cases} 0\ , & \text{se } p_i = \mathbf{f}. \\ \#p_i \times \wedge\, REF_i \times (dot + comp), & \text{caso contrário.} \end{cases} \quad (70)$$

Logo, para toda a expressão de caminho:

$$custoCPU\_Aninhado = \sum_{i=1}^{\ell} custoCPU\_p_i \ . \quad (71)$$

O custo de validação dos predicados aninhados deve ser somado ao custo de acesso às coleções durante a travessia da expressão de caminho.

4.7.1.1 VALIDAÇÃO NOS FRAGMENTOS

Caso a coleção C_i esteja fragmentada, o custo de validação de p_i em cada fragmento F_j^i que não for eliminado da consulta é obtido de maneira análoga:

$$custoCPU\_p_j^i = \begin{cases} 0\ , & \text{se } p_i = \mathbf{f}. \\ \#p_i \times \wedge\, ref_j^i \times (dot + comp), & \text{caso contrário.} \end{cases} \quad (72)$$

O custo de validação de p_i na coleção C_i fragmentada é:

$$custoCPU\_p_i = \sum_{F_j^i \notin E_i} custoCPU\_p_j^i \ . \quad (73)$$

4.7.2 MANUTENÇÃO DE RESULTADOS INTERMEDIÁRIOS

Dependendo da estratégia adotada para avaliação de uma expressão de caminho, pode ser necessário manter resultados intermediários armazenados em tabelas de suporte. Este custo adicional é típico dos operadores de junção, baseada em valor ou baseada em ponteiros.

Os resultados intermediários gerados em uma expressão de caminho consistem em tuplas de IDOs qualificados e, eventualmente, atributos que irão compor o resultado final. Em geral, o *cache* auxiliar do SGBD é suficiente para manter os resultados

intermediários, pois o tamanho de um IDO varia em torno de 4-16 *bytes*, assim pouca memória é necessária para conter uma tabela de suporte, mesmo em coleções volumosas. Além disso, existem técnicas especiais que permitem diminuir os custos de E/S referentes à propagação dos atributos do resultado final, quando são definidas projeções nos predicados aninhados da expressão de caminho (CLAUSSEN *et al.*, 2000).

Se a memória principal disponível não for suficiente para a manutenção dos resultados intermediários, o custo de E/S referente a uma tabela de suporte ST é:

$$custoES\_Suporte(ST) = |ST| \ , \tag{74}$$

$$\text{onde} \quad |ST| = \left\lceil \frac{\|ST\|}{\left\lfloor S_P / S_{ST} \right\rfloor} \right\rceil . \tag{75}$$

Observe que este custo refere-se às operações de escrita em disco. O custo de E/S referente à leitura de uma tabela de suporte a partir do disco deve ser considerado caso esta tabela seja utilizada como entrada de um operador da AP.

4.7.3 CUSTO DE COMUNICAÇÃO DE DADOS

Iremos estimar o custo de comunicação de dados através da relação entre o número de páginas acessadas nas coleções (operações de E/S) e o número de páginas transmitidas durante a avaliação de uma expressão de caminho.

4.7.3.1 FATOR DE ACESSIBILIDADE LOCAL

Na avaliação distribuída de uma expressão de caminho P, cada fragmento F_j^s da coleção de partida C_s representa o subconjunto P_j da extensão desta expressão, referente aos caminhos que começam nos objetos de F_j^s. Seja α_n $(0 \le \alpha \le 1)$ o fator de acessibilidade do nó n em relação às páginas de dados que contêm os objetos envolvidos em P_n, tal que:

$$P_n = \bigcup_{F_j^s \in frags_s^n} P_j \ . \tag{76}$$

O conjunto $frags_i^n$ representa os fragmentos da coleção C_i que não foram eliminados da expressão de caminho e estão alocados no nó n. A seletividade de um nó em relação aos objetos envolvidos em uma expressão de caminho depende da seletividade dos fragmentos da coleção de partida C_s que pertencem a $frags_s^n$, tal que:

$$selReqs_s^n = \begin{cases} 1 & \text{se } C_s \text{ está fragmentad a verticalm ente} \\ & \text{ou não está fragmentad a,} \\ \displaystyle\sum_{F_j^s \in frags_s^n} \left(\frac{sel_j^s \times SEL_j^s}{SEL_j^s} \right) & \text{caso contrário.} \end{cases} \tag{77}$$

Quando $\alpha_n = 1$, todas as páginas solicitadas durante a avaliação da subexpressão P_n estão armazenadas localmente em n e não há custo de comunicação de dados. Deste modo, na estratégia descendente, temos que:

$$\acute{a}_n = \frac{\displaystyle\sum_{i=2}^{\ell} \alpha_i^n \times \#pageReqs_i^n}{\displaystyle\sum_{i=2}^{\ell} \#pageReqs_i^n} \ , \tag{78}$$

onde $\#pageReqs_i^n$ é o número de páginas requisitadas na coleção C_i pelo nó n durante a avaliação da expressão de caminho, sendo calculado de acordo com o algoritmo que opera sobre C_i. Além disso, α_i^n é o fator de acessibilidade da coleção C_i pelo nó n, tal que $\alpha_i^n = 0$ se não há fragmentos de C_i alocados em n e:

$$\acute{a}_i^n = \begin{cases} \dfrac{\#frags_i^n}{f_i - \#E_i} & \text{se a fragmentação de } C_i \text{ é vertical,} \\ selFrags_i^n & \text{se a fragmentação de } C_i \text{ é horizontal primária,} \\ 1 & \text{se a fragmentação de } C_i \text{ é derivada de } C_{i-1}, \\ & \text{ou } C_i \text{ não está fragmentada.} \end{cases} \tag{79}$$

$$\text{e} \qquad selFrags_i^n = \frac{\sum sel_j^i}{1 - selE_i} \ . \tag{80}$$

Consideramos que fragmentos horizontais derivados estão alocados no mesmo nó de seus respectivos fragmentos primários. O cálculo de α_n na estratégia ascendente é obtido de maneira análoga a (78), variando i de $\ell - 1$ a 1.

Finalmente, na estratégia descendente, o custo de comunicação referente às páginas remotas solicitadas em um nó n é estimado por:

$$custoCOMTravessia_n = \lceil (1 - \alpha_n) \times \#pageReqs_n \rceil \times send \ , \tag{81}$$

$$\text{onde} \quad \#pageReqs_n = \sum_{i=2}^{\ell} \#pageReqs_i^n \ . \tag{82}$$

Note que $send$ é o custo para transmitir uma página de dados na rede e o termo $\#pageReqs_n$ representa o volume de páginas solicitadas na travessia da expressão de caminho a partir da segunda coleção avaliada.

Por conseguinte, o custo total de comunicação na avaliação distribuída de uma expressão de caminho é calculado por:

$$custoCOM\_Travessia = \sum_{n=1}^{\#nós} custoCOM\_Travessia_n \ , \tag{83}$$

$$\text{tal que } \#nós = \begin{cases} 1 & \text{se a coleção de partida } (C_s) \text{ está fragmentad a verticalm ente} \\ & \text{ou não está fragmentad a,} \\ (\mathrm{f_s} - \#E_s) & \text{caso contrário.} \end{cases} \quad (84)$$

4.7.4 RECONSTRUÇÃO DE RESULTADOS INTERMEDIÁRIOS

A avaliação de uma expressão de caminho definida sobre coleções fragmentadas muitas vezes requer que resultados intermediários provenientes de fragmentos de uma coleção sejam "unificados", tal que a coleção de objetos globais participantes desses resultados intermediários seja reconstruída .

No caso da fragmentação horizontal, a reconstrução de uma coleção intermediária é realizada através de um operador n-ário de união aplicado aos resultados intermediários obtidos em cada fragmento (CASANOVA e MOURA, 1985). Consideramos que o custo deste operador de união equivale à transmissão dos resultados intermediários para um determinado nó do sistema. Deste modo, assumindo que não será necessário transmitir um dos fragmentos, temos:

$$custoCOM\_Reconstrução_i = \sum_{j=1}^{(f_i - \#E_i)-1} \left| ST_j^i \right| . \quad (85)$$

Já na fragmentação vertical, a reconstrução de uma coleção intermediária é feita a partir de $(f_i - \#E_i) - 1$ junções, baseadas em valor de IDO, sobre os resultados intermediários provenientes de cada fragmento dessa coleção. Consideramos que não é possível utilizar junções baseadas em ponteiros porque o atributo de junção (IDO do objeto global) pode não representar o endereço de seus respectivos fragmentos. É desejável que estas junções sejam realizadas a partir do fragmento onde o predicado aninhado apresenta maior seletividade, para reduzir o volume dos resultados intermediários gerados durante a reconstrução. Deste modo, na avaliação de uma expressão de caminho, o custo de CPU de cada junção de reconstrução de resultados intermediários da coleção C_i fragmentada verticalmente, provenientes de dois fragmentos verticais (F_{j1}^i e F_{j2}^i), é dado pela seguinte fórmula:

$$custoCPU\_ReconstruçãoV^{i}_{j1,j2} = \left(ref_{*}^{\,i}\right)^2 \times SEL^{i}_{j1} \times SEL^{i}_{j2} \times comp + ref_{*}^{\,i} \times SEL^{i}_{j2} \times move \quad , \tag{86}$$

considerando que $SEL^{i}_{j2} \leq SEL^{i}_{j1}$. Lembramos que a cardinalidade do resultado intermediário advindo de um fragmento vertical F^{i}_{j} é dada por $ref_{*}^{\,i} \times SEL^{i}_{j}$. Além disso, $ref_{*}^{\,i}$ corresponde ao número de objetos de C_i recuperados pela seletividade da expressão de caminho, o qual é o mesmo para todos os fragmentos verticais de C_i que não foram eliminados da travessia. A equação (86) refere-se ao custo para comparar entre si os objetos dos resultados intermediários e ao custo para gerar uma tabela de suporte com a coleção intermediária parcialmente reconstruída. Logo, o custo total de CPU para a reconstrução de C_i é:

$$custoCPU\_ReconstruçãoV_i = \sum_{j=1}^{(f_i - \#E_i)-1} custoCPU\_ReconstruçãoV^{i}_{j,j+1} \quad , \tag{87}$$

onde os fragmentos do somatório estão ordenados pela seletividade do predicado aninhado, em ordem crescente. Consideramos que os resultados intermediários advindos de cada fragmento são mantidos em memória e, portanto, não há custo de E/S envolvido na reconstrução da coleção intermediária. Quanto ao custo de comunicação, assumimos que cada fragmento está alocado em um nó próprio e será necessário enviar a tabela de suporte contendo os resultados intermediários de um dos fragmentos em cada junção de reconstrução. Assim, o número de páginas transmitidas neste processo é dado igualmente por (85).

4.7.5 OPERADOR UNÁRIO DE SELEÇÃO

Neste modelo de custo, estamos considerando que o operador unário de seleção pode ser utilizado na avaliação de uma expressão de caminho de tamanho $\ell = 1$, ou ainda na aplicação de predicados aninhados antes da realização de uma junção. Iremos considerar para a análise de custos o algoritmo para varredura seqüencial de uma coleção. O custo de E/S de dados deste algoritmo independe da estratégia de avaliação, sendo estimado de maneira trivial por:

$$custoES\_Seqüencial(C_i) = \begin{cases} |C_i| & \text{se } C_i \text{ não estiver fragmentad a,} \\ \sum_{F_j^i \notin E_i} |F_j^i| & \text{caso contrário.} \end{cases} \quad (88)$$

O respectivo custo de CPU relativo à varredura dos objetos é calculado pela seguinte equação:

$$custoCPU\_Seqüencial(C_i) = \begin{cases} \hat{}REF_i \times hash & \text{se } C_i \text{ não estiver fragmentad a,} \\ \sum_{F_j^i \notin E_i} \hat{}ref_j^i \times hash & \text{caso contrário.} \end{cases} \quad (89)$$

Lembramos que deve ser acrescentado o custo de CPU referente à validação do predicado aninhado p_i, quando for o caso. Para coleções fragmentadas, deve-se atentar também para o eventual custo de reconstrução do resultado deste algoritmo.

4.7.6 OPERADOR BINÁRIO

Consideramos que a avaliação de expressões de caminho através do operador binário de junção pode ser realizada por dois tipos básicos de algoritmos: (i) por algoritmos de junção baseada em valor, equivalentes aos algoritmos clássicos de junção relacional; ou (ii) por algoritmos de junção baseada em ponteiros, que são inerentes de bancos de dados baseados em objetos. Existem várias propostas de algoritmos de junção na literatura (laços aninhados, *sort-merge*, *hybrid-hash*, etc.), sejam baseados em valor (MISHRA *et al*., 1992) ou em ponteiros (BRAUMANDL *et al*., 2000, SHEKITA e CAREY, 1990).

4.7.6.1 AVALIAÇÃO POR JUNÇÃO BASEADA EM VALOR

Na avaliação de expressões de caminho, a aplicação típica desta categoria de algoritmo ocorre na estratégia ascendente, quando o predicado aninhado da coleção C_ℓ possui um alto fator de seletividade e não estão definidos relacionamentos inversos ao caminho.

Para a análise de custos da junção baseada em valor no contexto da avaliação de expressões de caminho, iremos considerar o tradicional algoritmo de laços aninhados,

amplamente conhecido em SGBDs relacionais. Sejam duas coleções C_A e C_B relacionadas através do atributo A_A. Assumimos que os predicados aninhados p_A e p_B foram aplicados e apenas os objetos qualificados irão participar da junção. Inicialmente, são criadas entradas em uma tabela de suporte por dispersão (*hashing*) para os objetos de C_B qualificados por p_B. Cada entrada nesta tabela aponta diretamente para o endereço físico do respectivo objeto de C_B. Em seguida, cada objeto de C_A é carregado na memória e comparado pelo atributo A_A com todos os objetos de C_B da tabela de suporte por dispersão. Uma página de memória é alocada para manter os objetos de C_A, enquanto as $m-1$ páginas restantes no *cache* de dados são utilizadas para carregar os objetos de C_B durante as comparações. Logo, o número de páginas acessadas nas coleções C_A e C_B durante uma junção baseada em valor de IDO é calculado como:

$$custoES\_VJ(C_A, C_B) = |C_A| + \#pageReqs_B \ , \tag{90}$$

$$\text{onde} \quad \#pageReqs_B = |C_B| + M_B \times Yao'(C_A, \|C_A\| \times SEL_A) \times \frac{Yao'(C_B, \|C_B\| \times SEL_B)}{m-1} \tag{91}$$

$$\text{e} \qquad M_B = \begin{cases} 0, & \text{se } m > Yao'(C_B, \|C_B\| \times SEL_B) \\ 1, & \text{caso contrário.} \end{cases} . \tag{92}$$

A variável M_B indica se o *cache* de dados é suficiente para manter todas as páginas solicitadas em C_B. Quando não há memória suficiente, as páginas de C_B serão recarregadas de acordo com o *cache* disponível. Para cada par de objetos de C_A e de C_B que satisfazem a junção, é criada uma entrada em uma tabela de suporte simples, contendo os respectivos IDOs. Por sua vez, o custo de CPU é dado por:

$$\begin{aligned} custoCPU\_VJ(C_A, C_B) = {} & (\|C_A\| + \|C_B\|) \times hash + \|C_B\| \times SEL_B \times store \\ & + \|C_A\| \times SEL_A \times fan_{A,B} \times dot \\ & + \|C_A\| \times SEL_A \times fan_{A,B} \times \|C_B\| \times SEL_B \times (hash + comp) \\ & + \|C_B\| \times SEL_B \times move \end{aligned} . \tag{93}$$

A utilização do algoritmo de junção baseada em valor na avaliação de uma expressão de caminho de tamanho ℓ implica na realização de $\ell - 1$ junções sucessivas sobre as coleções, tomadas duas a duas, além das seleções relativas aos predicados aninhados, quando for o caso. Os resultados intermediários gerados por cada junção são utilizados como entrada da junção posterior. Se uma das coleções é uma tabela de suporte, referente ao resultado intermediário de um outro algoritmo, assumimos que não há custo de leitura desta coleção a partir do disco. Na estratégia ascendente, o número de páginas acessadas durante a avaliação de uma expressão de caminho é:

$$custoES\_VJ = C_{\ell} + \sum_{i=\ell-1}^{1} \#pageReqs_i \ , \tag{94}$$

$$\text{onde } \#pageReqs_i = |C_i| + M_i \times \left(Yao'(C_{i+1}, REF_{i+1} \times SEL_{i+1}) \times \frac{Yao'(C_i, \|C_i\| \times SEL_i)}{m-1} \right) \tag{95}$$

$$\text{e} \qquad M_i = \begin{cases} 0, & \text{se } m > Yao'(C_i, \|C_i\| \times SEL_i) \\ 1, & \text{caso contrário.} \end{cases} . \tag{96}$$

O custo de CPU deste algoritmo pode ser muito alto, pois são mantidas tabelas de suporte por dispersão e cada objeto qualificado na coleção externa será comparado com todos os objetos qualificados na coleção interna. Logo:

$$\begin{aligned} custoCPU\_VJ = {} & \|C_{\ell}\| \times hash + \\ & \sum_{i=\ell-1}^{1} \left(\begin{array}{l} \|C_i\| \times hash + \|C_i\| \times SEL_i \times store \\ + REF_{i+1} \times SEL_{i+1} \times \|C_i\| \times SEL_i \times (hash + fan_{i,i+1} \times (dot + comp)) \\ + REF_i \times SEL_i \times move \end{array} \right) . \end{aligned} \tag{97}$$

Para a estratégia descendente, os custos são obtidos de maneira análoga ao apresentado para a estratégia ascendente. Entretanto, dificilmente utiliza-se a junção baseada em valor quando é possível o uso de algoritmos baseados em ponteiros.

⇨ EFEITO DA FRAGMENTAÇÃO

No caso de uma coleção C_i estar fragmentada segundo uma fragmentação horizontal primária, será realizada uma junção entre cada fragmento F_j^i que não foi eliminado e a coleção C_{i-1}, considerando a estratégia ascendente. Caso C_{i-1} também possua uma fragmentação horizontal primária, todos os fragmentos não eliminados de C_{i-1} deverão ser enviados para o nó onde F_j^i está localizado. Estas junções podem apresentar altos custos de processamento, os quais são estimados pelas fórmulas (90) e (93) substituindo-se as coleções pelos respectivos fragmentos. Entretanto, se a fragmentação da coleção C_i é derivada em relação à C_{i-1}, ou vice-versa, serão realizadas somente as junções de cada fragmento F_j^i com seu respectivo fragmento primário (ou derivado) em C_{i-1}. Isso reduz bastante o volume de operações e a comunicação de dados na avaliação de uma expressão de caminho.

Consideramos que cada fragmento vertical de uma coleção C_i é inicialmente validado por p_i e que os resultados intermediários destas validações são unificados em uma tabela de suporte por dispersão, a qual aponta para o fragmento vertical de C_i que contém o atributo de referência utilizado pela expressão de caminho. Somente após esta unificação, C_i participa da junção baseada em valor.

Deste modo, na estratégia ascendente, em cada nó n teremos:

$$custoES\_VJ_n = \sum_{F_j^\ell \in frags_\ell^n} \left|F_j^\ell\right| + \sum_{i=\ell-1}^{1} \#pageReqs_i^n \quad , \tag{98}$$

onde

$$\#pageReqs_i^n = \sum_{F_j^i \notin E_i} \left|F_j^i\right| + M_i \times Yao'\left(C_{i+1}, selReqs_\ell^n \times REF_{i+1} \times SEL_{i+1}\right) \times \frac{Yao'\left(C_i, \|C_i\| \times SEL_i\right)}{m-1} \; . \tag{99}$$

Por fim, o custo de CPU em cada nó n é estimado por:

$$custoCPU\_VJ_n = \sum_{F_j^\ell \in frag_\ell^n} \left(\left\| F_j^\ell \right\| \times hash \right) + \sum_{i=\ell-1}^{1} \left(\left\| C_i \right\| \times hash + \left\| C_i \right\| \times SEL_i \times store \right) + selReqs_\ell^n \times \sum_{i=\ell-1}^{1} \begin{pmatrix} REF_{i+1} \times SEL_{i+1} \times \left\| C_i \right\| \times SEL_i \times \left(hash + fan_{i,i+1} \times \left(dot + comp \right) \right) \\ + REF_i \times SEL_i \times move \end{pmatrix} . \quad (100)$$

4.7.6.2 JUNÇÃO BASEADA EM PONTEIROS

A análise de desempenho deste tipo de algoritmo será de acordo com o algoritmo de laços aninhados para junção baseada em ponteiros. Assim, sejam duas coleções C_A e C_B, relacionadas através do atributo A_A, onde C_A é a coleção externa na junção. Assumimos que o predicado aninhado p_A já foi aplicado à coleção C_A, tal que apenas os objetos qualificados participarão da junção. Segundo este algoritmo, uma página de memória é usada para manter os objetos de C_A enquanto o restante do *cache* de dados é utilizado para carregar somente as páginas dos objetos de C_B que são referenciados por C_A. Para cada objeto de C_A carregado na memória, os ponteiros contidos no atributo A_A são decodificados e as páginas dos respectivos objetos de C_B são carregadas na memória. É criada uma entrada em uma tabela de suporte para cada par de IDOs qualificados na junção de C_A e C_B. Logo, o número de páginas acessadas nas coleções C_A e C_B durante uma junção baseada em ponteiros por laços aninhados é estimado pela seguinte equação:

$$custoES\_PJ(C_A, C_B) = |C_A| + \#pageReqs_B \ , \quad (101)$$

onde

$$\#pageReqs_B = \#pageHits_B + M_B \times \left(1 - \frac{1}{\#pageHits_B} \right)^{m-1} \times \#pageHits_B \times \left(share_{A,B} \times \Delta_{A,B} - 1 \right) \quad (102)$$

e

$$M_B = \begin{cases} 0, & \text{se } m \geq (1 + \#pageHits_B), \\ 1, & \text{caso contrário.} \end{cases} \quad (103)$$

A variável M_B indica se há recarga de páginas de C_B caso o *cache* não seja suficiente para armazenar todas as páginas envolvidas. Esta recarga é estimada de acordo com a probabilidade de uma página solicitada não estar presente na memória disponível $(m-1)$ e com o grau de compartilhamento (*share*) de C_B em relação à coleção C_A. Além disso, o termo $\Delta_{A,B}$ representa um fator de correção matemática para refletir o grau de distribuição física dos objetos de C_B no disco, em relação à alocação física dos objetos de C_A. Este fator confere flexibilidade à função de custo, devendo ser atribuído arbitrariamente ($1/share_{A,B} \leq \Delta_{A,B} \leq 1$). Quando $\Delta_{A,B} = 1/share_{A,B}$, os objetos de C_B estão agrupados com C_A. Por outro lado, quando $\Delta_{A,B} = 1$, os objetos estão totalmente espalhados no disco.

O custo de CPU corresponde às operações de tradução dos IDOs, de recuperação dos atributos de referência e de criação das entradas na tabela de suporte. Ou seja:

$$\begin{aligned} custoCPU\_PJ(C_A, C_B) = {} & \|C_A\| \times hash \\ & + \hat{}REF_B \times (dot + hash) + REF_B \times SEL_B \times move \end{aligned} \quad (104)$$

Em uma expressão de caminho com $\ell \geq 2$, são realizadas $\ell - 1$ junções sucessivas sobre as coleções, tomadas duas a duas, tal que os resultados intermediários gerados por cada junção são utilizados como entrada da junção posterior. O número de páginas acessadas durante a avaliação de uma expressão de caminho, na estratégia descendente, é dado pela seguinte fórmula:

$$custoES\_PJ = |C_1| + \sum_{i=2}^{\ell} \#pageReqs_i \ , \quad (105)$$

onde

$$\begin{aligned} \#pageReqs_i = {} & \#pageHits_i \\ & + M_i \times \left(1 - \frac{1}{\#pageHits_i}\right)^{m-1} \times \#pageHits_i \times (share_{i-1,i} \times \Delta_{i-1,i} - 1) \end{aligned} \quad (106)$$

$$\text{e} \qquad M_i = \begin{cases} 0, & \text{se } m \geq (1 + \#pageHits_i), \\ 1, & \text{caso contrário.} \end{cases} \tag{107}$$

Por outro lado, o custo de CPU é calculado por:

$$custoCPU\_PJ = \|C_1\| \times hash + \sum_{i=2}^{\ell} (\hat{}REF_i \times (dot + hash) + REF_i \times SEL_i \times move) \ . \tag{108}$$

⇨ EFEITO DA FRAGMENTAÇÃO

Durante o acesso via ponteiros a uma coleção com fragmentação horizontal primária, algumas páginas de objetos poderão estar localizadas remotamente. Já na fragmentação horizontal derivada, considerando que os fragmentos primários estão alocados com seus respectivos fragmentos derivados, todas as páginas solicitadas são acessadas localmente. Por outro lado, no caso da fragmentação vertical de uma coleção, todos os fragmentos envolvidos na expressão de caminho sempre são acessados.

Na estratégia descendente, em cada nó n temos que:

$$custoES\_PJ_n = \sum_{F_j^1 \in frags_1^n} \left| F_j^1 \right| + \sum_{i=2}^{\ell} \#pageReqs_i^n \ , \tag{109}$$

onde

$$\begin{aligned} \#pageReqs_1^n &= \#pageHits\_Reqs_i^n \\ &+ M_i \times \left(1 - \frac{1}{\#pageHits\_Reqs_i^n} \right)^{m-1} \times \#pageHits\_Reqs_i^n \times (share_{i-1,i} \times \Delta_{i-1,i} - 1) \end{aligned} \tag{110}$$

$$\text{e} \qquad M_i = \begin{cases} 0, & \text{se } m \geq (1 + \#pageHits\_Reqs_i^n), \\ 1, & \text{caso contrário.} \end{cases} \tag{111}$$

O termo $\#pageHits\_Reqs_i^n$ representa o número de páginas requisitadas pelo nó n na coleção C_i, sendo estimado pela seguinte fórmula:

$$\#pageHits\_Reqs_i^n = \begin{cases} \#pageHits_i\left(selReqs_1^n \times REF_i\right) & \text{se } C_i \text{ não está fragmentada.} \\ \text{Caso contrário,} & \\ \quad \sum_{F_j^i \notin E_i} \#pageHits_j^i\left(selReqs_1^n \times ref_j^i\right) & \end{cases} \tag{112}$$

Observe que a fragmentação reduz os requisitos de memória disponível em cada nó. Essa redução ocorre de acordo com a seletividade do nó n em relação aos fragmentos da coleção C_1.

Além disso, o custo de CPU em cada nó n é dado como:

$$custoCPU\_PJ = \sum_{F_j^1 \in frags_1^n} \left(\left\|F_j^1\right\| \times hash\right) + selReqs_1^n \times \sum_{i=2}^{\ell} \left(\hat{}REF_i \times \left(dot + hash\right) + REF_i \times SEL_i \times move\right) \tag{113}$$

.

4.7.7 OPERADOR N-ÁRIO

O operador n-ário representa a maneira mais intuitiva para avaliar uma expressão de caminho, cujo algoritmo clássico é conhecido como "busca ingênua por ponteiros" (*naïve pointer-chasing*). Este algoritmo percorre todos os caminhos possíveis em uma expressão de caminho, completos e incompletos, um por vez. Uma vantagem relevante a ser considerada neste algoritmo é que não são gerados resultados intermediários durante a avaliação da expressão de caminho, o que reduz substancialmente o custo de CPU do mesmo.

Consideramos que o número de páginas disponíveis para a execução deste algoritmo é maior ou igual ao comprimento da expressão de caminho ($m \geq \ell$). Assim, há sempre uma página de memória disponível para cada coleção envolvida. Caso contrário, a aplicação deste algoritmo não é viável (GARDARIN *et al.*, 1995).

Quando a memória principal é suficiente para armazenar todas as páginas referentes às coleções envolvidas na expressão de caminho, os objetos correspondentes serão lidos apenas uma vez na execução da consulta. Logo, se:

$$m \geq \left(1 + \sum_{i=2}^{\ell} \#pageHits_i \right), \tag{114}$$

o custo de E/S da avaliação descendente de uma expressão de caminho de tamanho $\ell \geq 2$ será dado por:

$$custoES\_NP = |C_1| + \#pageReqs \ , \tag{115}$$

$$\text{onde} \quad \#pageReqs = \sum_{i=2}^{\ell} \#pageHits_i \ . \tag{116}$$

Caso a memória não seja suficiente, o custo de E/S deste operador pode ser muito alto, pois as páginas de dados poderão ser recarregadas várias vezes, de acordo com o grau de compartilhamento das coleções envolvidas e com o espalhamento em disco dos objetos refenciados. Nesta situação de memória restrita, temos que o número total de páginas requisitadas é:

$$\#pageReqs = \sum_{i=2}^{\ell} \left(\#pageHits_i \times \Delta_{i-1,i} \times \prod_{k=2}^{i} share_{k-1,k} \right), \tag{117}$$

Observe que, se a expressão de caminho for monovalorada, o produtório contido em (117) será igual a 1.

Se as coleções envolvidas estão agrupadas fisicamente no sentido dos relacionamentos da expressão de caminho, o requisito de memória exposto em (114) cai drasticamente e este algoritmo apresenta um desempenho bastante satisfatório.

Continuando, o custo de CPU corresponde somente ao tempo para decodificar os IDOs e recuperar os atributos de referência, tal que:

$$custoCPU\_NP = \left\|C_1\right\| \times hash + \wedge REF_2 \times dot + p \times \sum_{i=2}^{\ell-1} \left(\left(REF_i \times hash + \wedge REF_{i+1} \times dot\right) \times \prod_{j=2}^{i} share_{j-1,j} \right) + REF_\ell \times hash \times \prod_{i=2}^{\ell} share_{i-1,i} \quad (118)$$

$$\text{onde} \quad p = \begin{cases} 1, & \text{se } \ell > 2, \\ 0, & \text{caso contrário.} \end{cases} \quad (119)$$

⇨ EFEITO DA FRAGMENTAÇÃO

Na estratégia descendente, o número de páginas acessadas em um nó n, localizadas remotamente ou não, é obtido por:

$$custoES\_NP_n = \sum_{F_j^1 \in frags_1^n} \left|F_j^1\right| + \sum_{i=2}^{\ell} \#pageReqs_i^n \quad . \quad (120)$$

$$\text{onde} \quad \#pageReqs_1^n = \begin{cases} \#pageHits\_Reqs_i^n & \text{se } m \geq \left(1 + \sum_{i=2}^{\ell} \#pageHits\_Reqs_i^n\right) \\ \text{Caso contrário,} & \\ \left(\#pageHits\_Reqs_i^n \times \Delta_{i-1,1} \times \prod_{k=2}^{i} share_{k-1,k}\right) & \end{cases} \quad (121)$$

O termo $\#pageHits\_Reqs_i^n$ é definido na fórmula (112).

Por fim, o custo de CPU em cada nó n é calculado por:

$$custoCPU\_NP_n = \sum_{F_j^1 \in frags_1^n} \left(\left\|F_j^1\right\| \times hash\right) + selReqs_1^n \times \wedge REF_2 \times dot + selReqs_1^n \times p \times \sum_{i=2}^{\ell-1} \left(\left(REF_i \times hash + \wedge REF_{i+1} \times dot\right) \times \prod_{j=2}^{i} share_{j-1,j} \right) + selReqs_1^n \times REF_\ell \times hash \times \prod_{i=2}^{\ell} share_{i-1,i} \quad (122)$$

4.8 CONSIDERAÇÕES FINAIS

Se um algoritmo baseado em ponteiro é utilizado, a análise de custo da estratégia ascendente corresponde à aplicação do operador equivalente ao caminho inverso da expressão de caminho, o qual é definido pelos atributos de relacionamento inverso. Deste modo, as fórmulas para a estratégia ascendente são análogas às apresentadas.

Os algoritmos considerados para representar os operadores analisados no nosso modelo de custo podem ser otimizados e existem diversas propostas na literatura abordando tais otimizações (BRAUMANDL *et al.*, 2000). Entretanto, as características básicas de cada operador, como a manutenção ou não de resultados intermediários, permanecem as mesmas. Portanto, nossas considerações continuarão válidas e com aplicação genérica.

Capítulo 5

ANÁLISE DO MODELO DE CUSTO

Neste capítulo são realizadas comparações entre os resultados gerados pelo modelo de custo proposto e alguns resultados experimentais obtidos em trabalhos anteriores do grupo de Banco de Dados da COPPE/UFRJ. Além disso, o comportamento do modelo é examinado através de resultados obtidos pela simulação de expressões de caminho sobre configurações sintéticas do benchmark OO7.

5.1 INTRODUÇÃO

O objetivo deste capítulo é verificar os fundamentos considerados no modelo de custo proposto. Para isso, examinamos o comportamento das nossas funções de custo através da comparação com resultados experimentais apresentados em TAVARES (1999). Em uma análise complementar à validação experimental, simulamos baterias de resultados gerados pela variação de um conjunto de aspectos do processamento de expressões de caminho em configurações sintéticas do *benchmark* OO7 (CAREY *et al*., 1993). RUBERG *et al*. (2001b) apresentam heurísticas para o processamento de expressões de caminho obtidas pela análise de resultados simulados, gerados por um protótipo programado em *Borland Delphi 5.0* que implementa as funções do nosso modelo de custo sobre bases unificadas. Nós estendemos este protótipo para abranger as funções referentes à distribuição dos objetos.

A base da análise apresentada neste capítulo consiste na variação das características que influenciam o desempenho dos operadores típicos para a avaliação de expressões de caminho em bases de objetos distribuídos. Na validação experimental, utilizamos resultados reportados em TAVARES (1999) relativos às avaliações centralizada e paralela de expressões de caminho, a última sobre fragmentos horizontais das coleções. Além das expressões de caminho extraídas de trabalhos experimentais, elaboramos um conjunto de cenários visando explorar características tratadas no modelo de custo, como o grau de compartilhamento das coleções e as diferentes políticas de armazenamento físico dos objetos.

Os resultados gerados a partir de um *benchmark* envolvem uma grande quantidade de números, o que pode confundir ou tirar o foco da análise de desempenho (CAREY *et al*., 1993). Por outro lado, a geração de um indicador único de desempenho não permite uma compreensão abrangente dos itens analisados. Deste modo, é necessário estabelecer que conjunto de critérios orientarão a análise dos diversos cenários. Inicialmente, realizamos a validação experimental do nosso modelo de custo, onde o desempenho é apreciado de acordo com o número de operações de E/S, em relação a páginas de dados locais e remotas, quando for o caso. Em seguida, avaliamos o comportamento dos resultados gerados pelo modelo de custo em várias simulações sobre o *benchmark* OO7, em operações de E/S.

Os cenários que elaboramos são determinados pelos seguintes aspectos: (i) o fator de seletividade de predicados aninhados; (ii) a quantidade de memória disponível no *cache* de dados; (iii) a participação das coleções nos relacionamentos da expressão de caminho; e (iv) o grau de compartilhamento das coleções. Cada um desses aspectos é analisado segundo diferentes estratégias para a avaliação de expressões de caminho, nas direções descendente e ascendente.

Iremos representar as estratégias de avaliação segundo a combinação de algoritmo utilizado (junção baseada em valor – VJ, junção baseada em ponteiro – PJ e "*naïve pointer-chasing*" – NP) e direção de avaliação (descendente – D, ascendente – A). Assim, a estratégia NP-D corresponde à avaliação descendente utilizando o algoritmo NP.

5.2 O *BENCHMARK* OO7

O *benchmark* OO7 é uma proposta para a avaliação de desempenho em bancos de dados baseados em objetos, a qual é adotada por um grande número de trabalhos na literatura (OSTHOFF *et al*., 2000, SOARES, 2000, BAIÃO *et al.*, 2000, TAVARES *et al*., 2000, SU *et al.*, 2000, DEWITT *et al.*, 1996, GARDARIN *et al.*, 1996, GRUSER, 1996, LIMA e MATTOSO, 1996, SHRUFI, 1994). Este *benchmark* permite testar o impacto de importantes características do modelo de dados OO no desempenho do processamento de consultas, como a existência de atributos multivalorados e os diferentes tipos de navegação entre ponteiros.

O diagrama de classes do *benchmark* OO7 representa um modelo de aplicação do tipo CAD/CAM/CASE, mas não é referente a nenhuma aplicação específica. A idéia geral é o projeto de objetos através da montagem de um ou mais módulos, onde cada módulo constitui um conjunto de peças complexas (*CompositParts*) formadas a partir de peças mais simples (*AtomicParts*), interconectadas entre si (através de *Connections*). Cada peça complexa possui um manual associado (*Document*). Existem ainda as *BaseAssembly*, peças de alto nível formadas por *CompositParts* privadas (*ComponentsPriv*) e compartilhadas (*ComponentsShared*). Utilizaremos uma adaptação do *benchmark* OO7, a qual exclui aspectos que não interessam à nossa análise, a fim de

simplificar os cenários que serão estudados. O esquema simplificado é mostrado na Figura 9.

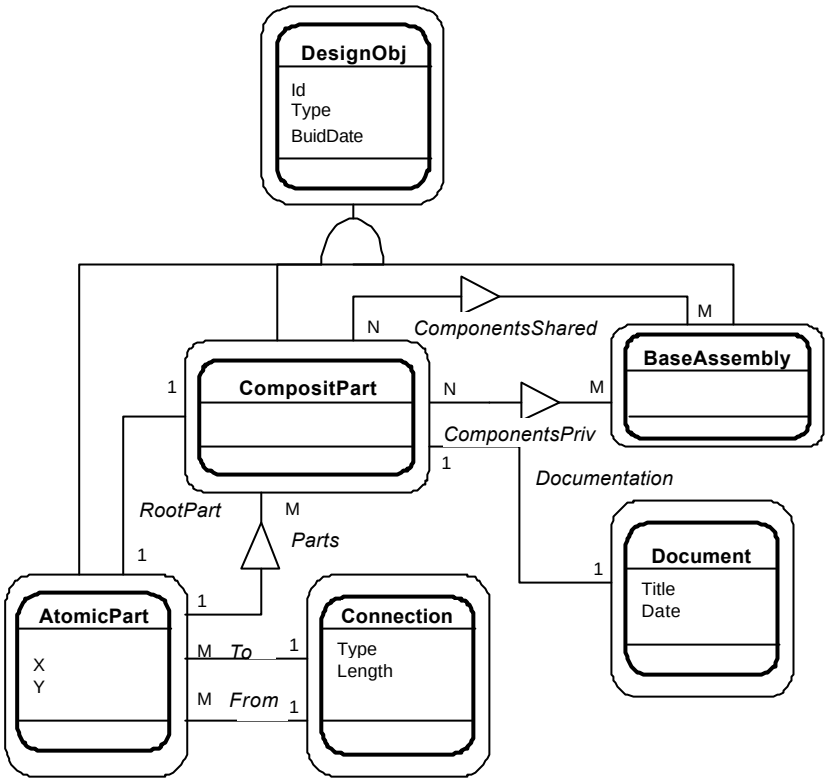


Figura 9. Diagrama de classes adaptado do *benchmark* OO7

O OO7 define três configurações básicas de bases de dados: (i) pequena, elaborada de tal forma que a base de dados cabe inteiramente no *cache* de dados; (ii) média, cuja base não cabe na memória principal disponível; e (iii) grande, onde a base de dados não cabe no *cache* e são explorados recursos de ambiente multi-usuário e compartilhamento de objetos. A cardinalidade das coleções relativas às classes do *benchmark* OO7 na configuração média, sobre a qual realizamos nossa análise e foi desenvolvido o trabalho experimental que consideramos, são apresentadas na Tabela 4.

Tabela 4. Cardinalidade das coleções do *benchmark* OO7 na configuração média

CLASSE	NÚMERO DE OBJETOS
CompositeParts	500
Documents	500
AtomicParts	100000
Connections	300000

5.3 RESULTADOS EXPERIMENTAIS UTILIZADOS

5.3.1 BASES UNIFICADAS E BASES FRAGMENTADAS HORIZONTALMENTE

O trabalho apresentado em TAVARES (1999) reporta os resultados obtidos experimentalmente através da execução paralela de um conjunto de consultas especificadas pelo *benchmark* 007 no sistema de gerenciamento de objetos GOA++ (MAURO *et al.*, 1997). O objetivo deste estudo foi uma análise do comportamento das diferentes alternativas para execução de consultas OO em um ambiente de processamento paralelo. Os experimentos utilizaram a fragmentação proposta por BAIÃO *et al.* (1998) para o esquema do *benchmark* 007, que consiste em:

- *CompositeParts* – fragmentação primária por faixa de valores (atributo *BuildDate*);
- *Documents* – fragmentação derivada em relação a *CompositeParts*;
- *AtomicParts* – fragmentação primária por faixa de valores (atributo *BuildDate*);
- *Connections* – fragmentação derivada em relação a *AtomicParts.*

Foram analisadas expressões de caminho monovaloradas e multivaloradas, conforme a Tabela 5. A avaliação descendente foi realizada pelo algoritmo NP (*naïve pointer-chasing*) e a ascendente por um algoritmo de junção baseada em valor (*hybrid hash*). Os experimentos foram realizados em um supercomputador paralelo IBM 9076 SP/2 do LNCC (Laboratório Nacional de Computação Científica), o qual apresenta uma arquitetura de memória distribuída e executa o sistema operacional IBM AIX 4.1.4. Este supercomputador possui 40 processadores, sendo 32 com capacidade de 256 Mb de memória RAM e desempenho de 266 M*flops*. Os processadores restantes possuem capacidade de 1 G*byte* de RAM e 308 M*flops* de CPU.

Os tempos dos experimentos foram medidos como tempo de espera do usuário, ou seja, a hora do sistema é registrada imediatamente antes da execução da consulta e imediatamente após a geração do resultado da mesma. Este tipo de medição implica que o tempo obtido em cada consulta corresponde a todas as fases de sua execução, desde a escolha de quais nós participarão do processo até a construção do resultados, incluindo o tempo gasto com operações de E/S e com a comunicação entre os nós. Nós utilizamos

as medições referentes às médias das execuções “frias” (primeira execução da consulta), onde a execução sofre menos influência da gerência de memória *cache* no SGBD.

Tabela 5. Consultas avaliadas por TAVARES (1999)

CONSULTA		SELETIVIDADE
Q1	`select a from a in AtomicParts where a.part_of.document.date<10/01/1990`	90%
Q2	`select a from a in AtomicParts where a.part_of.document.date<10/01/1910`	10%
Q3	`select c from c in CompositeParts where c.rootPart.builddate<10/01/1940`	40%
Q4	`select a from AtomicParts where a.to.length<20`	50%
Q5	`select c from c in CompositeParts where c.document.date<10/01/1940`	40%

5.3.2 BASE FRAGMENTADA VERTICALMENTE

Recursos para a fragmentação vertical de coleções de objetos não estão disponíveis na maioria dos SGBDs, como o *Oracle8i* (LEVERENZ *et al*., 1999), o *Sybase MPP* (STRINGER *et al*., 1996), o *Informix* (INFORMIX CORPORATION, 1999) e o GOA++ (MAURO *et al.*, 1997), o que dificulta a realização de trabalhos experimentais.

LIMA (1996) apresenta um estudo sobre técnicas de distribuição de objetos no SGBD O_2 (O_2 TECHONOLOGY, 1996), destacando parâmetros para avaliar o ganho de um projeto de distribuição. Foram realizadas várias medições de desempenho sobre uma base de objetos distribuídos horizontalmente e verticalmente, alocados em diferentes discos de uma rede de estações de trabalho. A base foi gerada segundo o *benchmark* OO7. Os resultados apresentados demonstram que a maioria das operações distribuídas oferece ganhos significativos de desempenho em relação ao processamento centralizado. As medições referentes à fragmentação vertical da coleção *AtomicParts* consideram que o projeto de distribuição isola os atributos *BuildDate* e *Type*. Foram

executadas consultas que variam a seletividade sobre esta coleção, de modo a eliminar a participação de alguns fragmentos.

Entretanto, não foi possível aplicar nosso modelo de custo aos cenários avaliados em LIMA (1996), pois as medições realizadas utilizaram acesso indexado às coleções durante a execução das consultas, alterando o volume de operações de E/S. Além disso, não estavam disponíveis alguns parâmetros relativos à configuração do SGBD, como o tamanho do *cache* de dados.

5.4 VALIDAÇÃO EXPERIMENTAL DO MODELO DE CUSTO

Esta Seção visa averiguar a fidedignidade das funções que compõem o nosso modelo de custo a partir dos resultados obtidos nos trabalhos experimentais. Como o tempo de resposta das medições pode apresentar interferências, porque não houve isolamento dos nós utilizados em relação à rede de trabalho, focalizamos nossa análise no número de operações de E/S e de páginas transferidas, no caso da execução distribuída. Não consideramos o custo de apresentação dos resultados das consultas.

A estratégia descendente foi executada experimentalmente pelo algoritmo NP. O desempenho do algoritmo de junção baseado em valor *hybrid hash*, utilizado na direção ascendente em TAVARES (1999), é comparável ao da junção baseada em ponteiro. Por isso, nas respectivas comparações, consideraremos que a junção experimental equivale à junção baseada em ponteiros.

5.4.1 BASE UNIFICADA

O número de páginas disponíveis no *cache* de dados foi determinado nos experimentos como $m = 2000$, onde uma página do banco de dados possui tamanho $S_P = 2048$. Este tamanho garante que o *cache* de dados, de 4 Mb, não será suficiente para conter todas as páginas acessadas durante a avaliação das expressões de caminho e ocorrerá recarga de algumas páginas, de acordo com o algoritmo utilizado.

As consultas utilizadas nesta validação possuem expressões de caminho com diferentes comprimentos, fatores de seletividade e volumes de dados. Além disso, elas abrangem diversas características do modelo de dados orientado a objetos, como

atributos multivalorados (Q1-A, Q2-A e Q4) e participação parcial das coleções nos relacionamentos (Q4). A Figura 10 mostra os resultados experimentais e os gerados por nosso modelo de custo, em número de operações de E/S. Somente as consultas *Q1* e *Q2* possuem resultados referentes à avaliação ascendente. Podemos observar que a estimativa do modelo de custo está muito próxima dos valores reais em todas as situações.

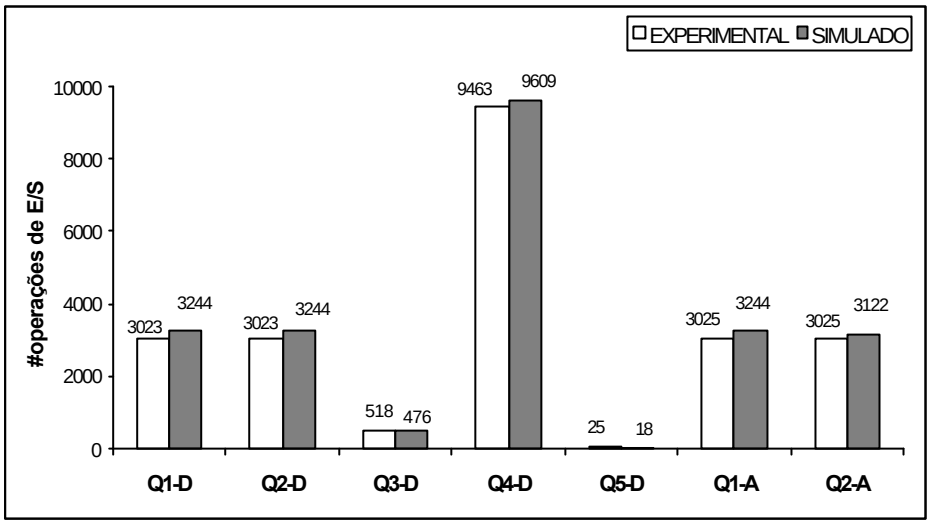


Figura 10. Validação do custo de E/S em uma base unificada

Na maioria dos resultados, o custo gerado pelo modelo de custo apresenta-se um pouco superior que os valores experimentais. Estas diferenças, segundo mostra a Figura 11, são bem pequenas. Além disso, elas são esperadas pois as funções de custo estimam o pior caso de acesso aleatório às páginas de objetos.

Tabela 6. Adequação do modelo de custo em uma base unificada

CONSULTA	CUSTO TOTAL DE E/S		
	EXPERIMENTAL	SIMULADO	DIFERENÇA (%)
Q1-D	3023	3244	7,3
Q2-D	3023	3244	7,3
Q3-D	518	476	-8,1
Q4-D	9463	9609	1,5
Q5-D	25	18	-28
Q1-A	3025	3244	7,2
Q2-A	3025	3122	3,2

Nas consultas onde o volume de páginas de dados recuperadas é pequeno (Q3-D e Q5-D), a estimativa do modelo de custo apresentou-se um pouco inferior ao desempenho real. Isso ocorreu porque, neste caso, o custo de estruturas e procedimentos próprios do SGBD (por exemplo, acesso ao catálogo durante a execução da consulta) apresenta peso significativo diante do pequeno número de páginas acessadas.

5.4.2 FRAGMENTAÇÃO HORIZONTAL

Neste caso, as consultas Q1-D, Q2-D, Q4-D foram executadas paralelamente em 2, 4, 8 e 12 nós, tal que as coleções foram fragmentadas de maneira uniforme, segundo o número de nós de processamento. Os resultados apresentados nos gráficos abaixo, em número de operações de E/S realizadas por nó, mostram que as predições geradas pelo modelo de custo são compatíveis com a execução experimental.

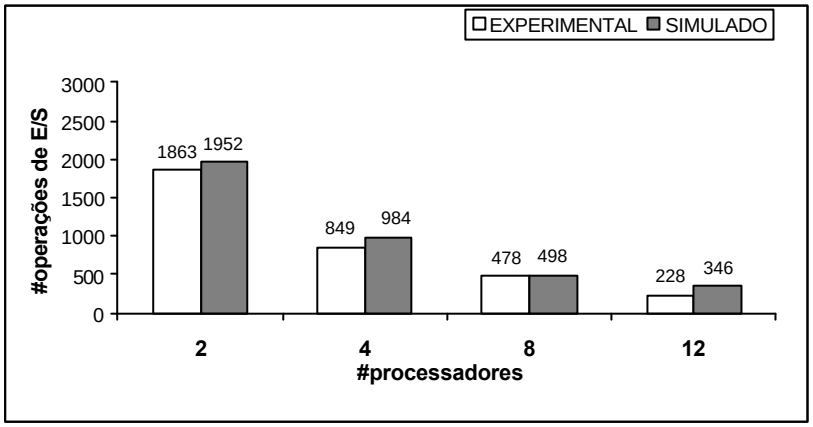


Figura 11. Custo de E/S na execução distribuída da consulta Q1-D

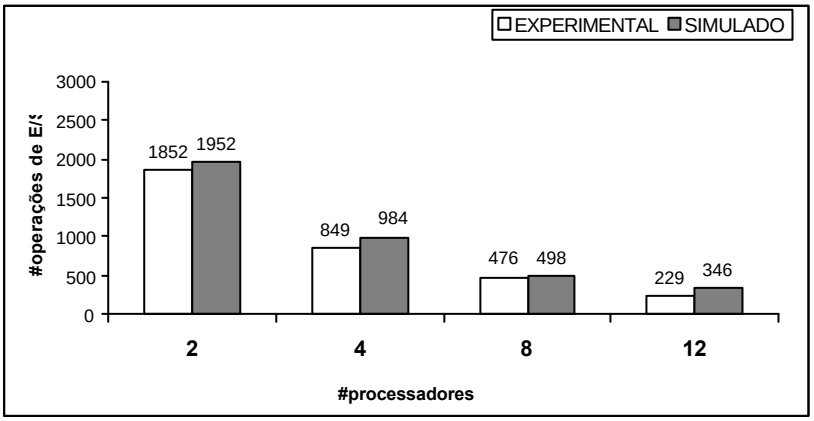


Figura 12. Custo de E/S na execução distribuída da consulta Q2-D

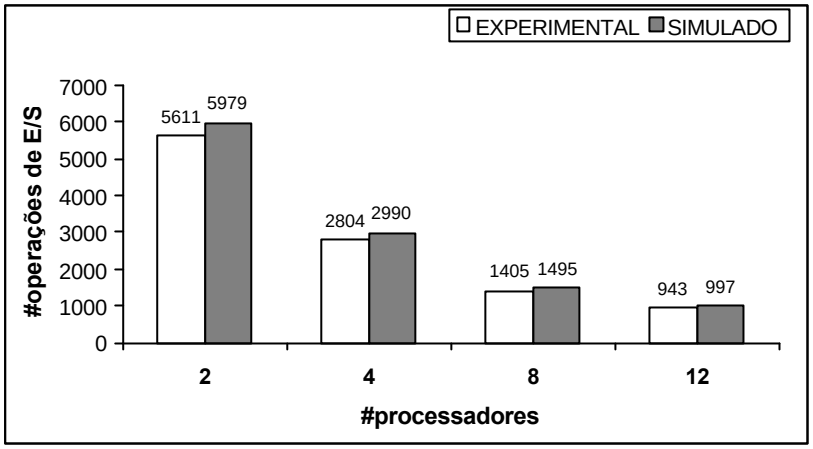


Figura 13. Custo de E/S na execução distribuída da consulta Q4-D

Na prática, em TAVARES (1999) foram realizadas várias otimizações quanto ao povoamento das coleções e à manutenção do *cache* de dados, as quais influenciaram o custo de comunicação entre os nós. Por isso, não foi possível comparar os resultados experimentais relativos ao volume de páginas remotas acessadas em cada consulta.

5.5 ANÁLISE DO CUSTO DE E/S EM BASES UNIFICADAS

Verificamos na Seção anterior a adequabilidade dos resultados gerados por nosso modelo de custo em relação ao desempenho de um determinado conjunto de consultas executadas experimentalmente. As expressões de caminho utilizadas na validação experimental são bastante significativas quanto à exploração das características inerentes do modelo de dados OO. Todavia, também é interessante que o comportamento das funções de custo também seja examinado em simulações sobre bases sintéticas, onde os parâmetros do modelo de custo podem ser variados mais efetivamente. De maneira similar ao apresentado em RUBERG *et al*. (2001b), nós analisamos o comportamento de seis estratégias para avaliação de expressões de caminho (VJ-D, VJ-A, PJ-D, PJ-A, NP-D e NP-A), comparando-as em diferentes cenários. Cada cenário foi resultante da variação de parâmetros como o fator de seletividade na última coleção, o tamanho da memória, o grau de compartilhamento das coleções envolvidas e a participação dessas coleções nos relacionamentos da travessia. Estes aspectos são pouco explorados em trabalhos relacionados (BRAUMANDL *et al*., 2000, GARDARIN *et al*., 1996, 1995), os quais limitam-se a características que geralmente não ocorrem na prática.

As estratégias para avaliação de expressões de caminho examinadas consideram a configuração do *benchmark* OO7 que foi utilizada na validação experimental. As análises foram realizadas sobre o custo de operações de E/S relativo à avaliação da expressão de caminho definida na Figura 14. Esta é uma expressão de caminho multivalorada de comprimento $\ell = 3$, que foi elaborada visando explorar ao máximo as características do modelo de dados OO que possuem impacto no desempenho das estratégias de avaliação. A combinação entre os graus de saída das coleções no sentido da travessia (maiores que 3) e os graus de compartilhamento no sentido inverso (maiores que 3) permite que sejam realçadas diferenças de desempenho decorrentes da direção na qual a expressão de caminho é avaliada.

```
select t
from c in CompositeParts, a in c.parts, t in a.to
```

```
EC: c.parts-a.to-t
```

Figura 14. Expressão de caminho utilizada nas simulações

O tamanho de uma página de dados é $S_P = 2048$. Nós fixamos o tamanho do *cache* de dados considerado nas simulações em $m = 2000$, com exceção do cenário onde a memória é variada. Na variação da seletividade, do tamanho da memória e do grau de compartilhamento das coleções envolvidas, nós assumimos que a participação das coleções nos relacionamentos da expressão de caminho é total. De maneira análoga, o fator de seletividade padrão dos predicados aninhados e o grau de compartilhamento padrão das coleções no sentido da expressão de caminho são considerados como 1. Todos os cenários consideram que os objetos estão reunidos fisicamente por extensão de classe, isto é, de acordo com a classe a qual pertencem.

5.5.1 VARIAÇÃO DA SELETIVIDADE

Nós variamos o fator de seletividade do predicado aninhado da última coleção na expressão de caminho (*Connections*), conforme a Figura 15. Como era esperado, os algoritmos baseados em ponteiros executados na estratégia descendente (NP-D e PJ-D) não se beneficiam da diminuição da seletividade na última coleção, visto que todos os objetos envolvidos precisam ser acessados para que o predicado seja aplicado.

Um predicado aninhado com fator de seletividade baixo (poucos objetos selecionados) ajuda a identificar qual a melhor coleção de partida, ou seja, a direção adequada para a avaliação de uma expressão de caminho. Nos nossos exemplos, como a seletividade foi aplicada na ultima coleção, a estratégia ascendente apresentou melhor desempenho. Essa diferença não foi maior porque a última coleção é a que possui maior volume e todos os seus objetos são acessados na avaliação ascendente.

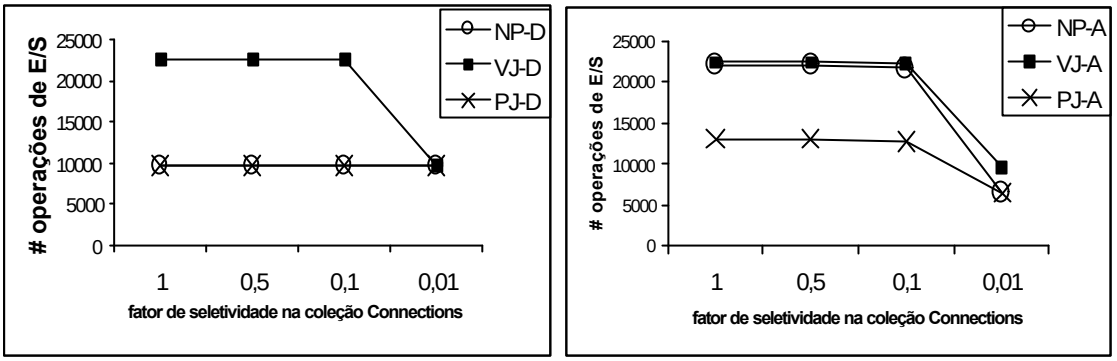


Figura 15. Variação de seletividade de predicado aninhado na coleção *Connections*

À medida que o fator de seletividade diminui, o algoritmo de junção baseada em valor (VJ) passa a ser uma alternativa interessante. Este fato é importante porque nem sempre existem ponteiros inversos definidos na base e o algoritmo VJ pode ser a única alternativa de execução na avaliação ascendente, visto que as estratégias descendentes são bem piores. Os resultados permitem observar também que, quando é realizado acesso baseado em ponteiros a uma coleção, os custos envolvidos na avaliação de expressões de caminho só serão sensibilizados por fatores de seletividade inferiores a 50%.

5.5.2 VARIAÇÃO DA MEMÓRIA

O tamanho do *cache* de dados disponível determina se haverá recarga de páginas durante a avaliação de uma expressão de caminho. Além disso, o volume de páginas de dados a serem recarregadas também é influenciado por esse fator. Nós simulamos a avaliação da expressão de caminho para uma variação de memória entre 1 Mb e 16 Mb.

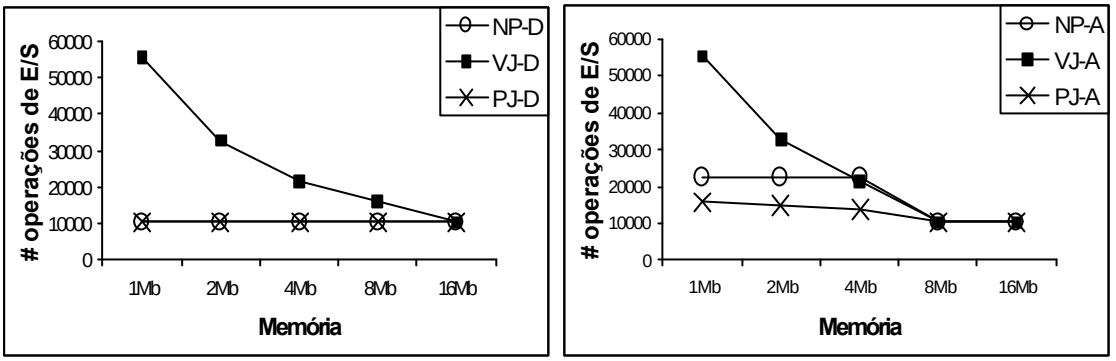


Figura 16. Variação da memória disponível no *cache* de dados

O desempenho dos algoritmos baseados em ponteiros é muito sensível à quantidade de memória disponível quando é alto o grau de compartilhamento no sentido da avaliação da expressão de caminho, podendo até equiparar-se ao desempenho da junção baseada em valor. No nosso exemplo, isto pode ser averiguado nas avaliações ascendentes e ocorre porque os objetos envolvidos na expressão de caminho são recarregados na memória várias vezes durante a navegação. Neste caso, a estratégia NP-A apresenta um desempenho pior que a PJ-A, porque no algoritmo NP a memória deve ser suficiente para conter todas as páginas solicitadas durante a avaliação da expressão de caminho. Por outro lado, a memória necessária no algoritmo PJ é relativa apenas às páginas acessadas na coleção apontada em cada junção realizada, onde as coleções são avaliadas duas a duas.

Na estratégia descendente, o desempenho dos algoritmos baseados em ponteiros não variou porque o grau de compartilhamento das coleções nesta direção é 1. Ou seja, todos os objetos são solicitados apenas uma vez e não há recarga das páginas de dados. Portanto, nestas condições, a memória não tem influência sobre o desempenho dos algoritmos.

Como podemos observar na Figura 16, todos os algoritmos são influenciados positivamente pelo aumento de memória, em especial o de junção baseada em valor (VJ). A quantidade de memória requerida por cada algoritmo é estimada no modelo de custo pelas equações (89), (100) e (110), podendo ser utilizada para auxiliar a configuração adequada do *cache* de dados de um SGBD.

5.5.3 PARTICIPAÇÃO DAS COLEÇÕES NOS RELACIONAMENTOS

A existência de relacionamentos parciais entre as classes de uma base de objetos consiste em um dos principais fatores que determinam a seletividade de uma expressão de caminho. Esta seletividade beneficia principalmente o desempenho dos algoritmos baseados em ponteiros, pois reduz o número de objetos recuperados na avaliação de uma expressão de caminho. Nós simulamos a variação da participação das coleções *AtomicParts* e *Connections* nos relacionamentos utilizados pela expressão de caminho, considerando que 100%, 50%, 10% e 1% dos objetos destas coleções são referenciados. Os resultados podem ser examinados na Figura 17.

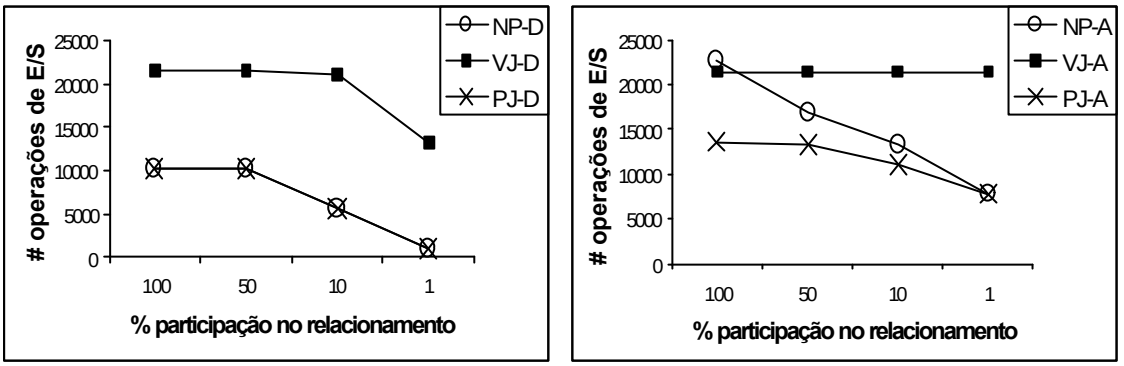


Figura 17. Variação da participação das coleções nos relacionamentos

Um aspecto importante a ser observado nas estratégias ascendentes da Figura 17 é que o algoritmo NP, cujo desempenho é o pior quando as coleções participam totalmente (100%) nos relacionamentos, passa a apresentar o melhor desempenho (empatado com o algoritmo PJ) quando a participação é reduzida a 1%. Vale ressaltar que o algoritmo NP apresenta um baixo custo de CPU em relação ao PJ. Tais aspectos são ignorados pela maioria dos trabalhos que analisam o desempenho da avaliação de expressões de caminho (BRAUMANDL *et al*., 2000, OZKAN *et al*., 1996, SHEKITA e CAREY, 1990), os quais descartam sumariamente o uso do algoritmo NP em decorrência de seu "reconhecido" baixo desempenho.

Como esperado, as estratégias relativas ao algoritmo de junção baseada em valor (VJ-D e VJ-A) são pouco sensíveis à participação das coleções envolvidas na expressão de caminho. Ainda assim, a seletividade dos relacionamentos parciais é propagada pela execução sucessiva das junções, visto que os resultados intermediários serão reduzidos.

5.5.4 VARIAÇÃO DO GRAU DE COMPARTILHAMENTO

O grau de compartilhamento é um fator importante na avaliação de expressões de caminho porque influencia o volume de páginas recarregadas quando a memória disponível não é suficiente, o que geralmente ocorre na prática. É comum supor que o volume dessa recarga de páginas depende do grau de saída das coleções (*fan-out*) (OZKAN *et al*., 1996, GARDARIN *et al*., 1996, 1995). Entretanto, se não há compartilhamento de objetos, a variação do grau de saída não irá implicar na recarga de páginas durante a avaliação de uma expressão de caminho.

De maneira geral, o grau de compartilhamento das coleções prejudica os algoritmos baseados em ponteiros, em especial o algoritmo n-ário (NP). Tal efeito ocorre principalmente se a memória disponível é insuficiente. Observe na Figura 18 que, a partir de um determinado ponto (quando o grau de compartilhamento é aproximadamente 3), as curvas das estratégias NP-D e NP-A são praticamente paralelas ao eixo de custo. Já quando as coleções possuem um alto grau de compartilhamento na direção da estratégia de avaliação de uma expressão de caminho, a junção baseada em valor representa uma alternativa interessante, pois este algoritmo não é sensível a essa característica, conforme mostra a Figura 18. Note que, de um modo isolado, o compartilhamento de objetos custa caro em relacionamentos cujo respectivo grau seja maior ou igual a três.

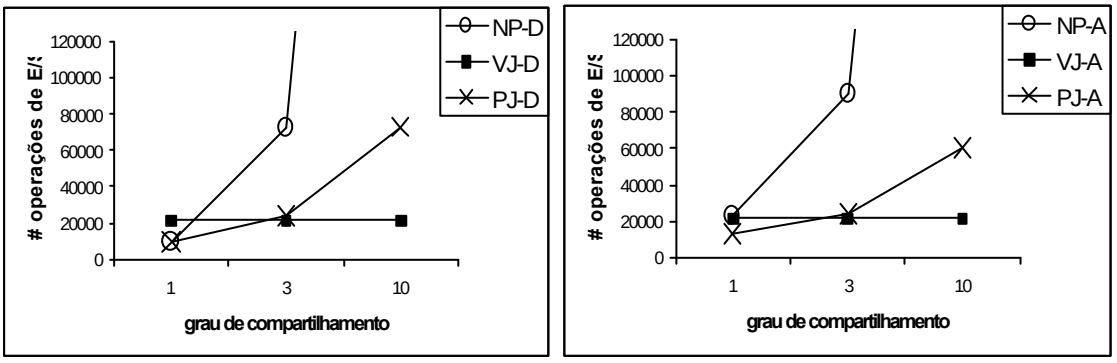


Figura 18. Variação do grau de compartilhamento das coleções

5.6 ANÁLISE DE CUSTOS EM COLEÇÕES FRAGMENTADAS

Agora, iremos verificar o comportamento de algumas estratégias de avaliação de expressões de caminho sobre bases de objetos distribuídos por uma fragmentação horizontal. Utilizamos a expressão de caminho definida na Figura 14. Consideramos que a distribuição da função de fragmentação horizontal é uniforme e proporcional ao número de nós envolvidos no processamento. Os cenários analisados assumem que os fragmentos das coleções estão distribuídos em quatro nós de processamento. Consideramos que o *cache* de dados de cada nó possui capacidade para 2000 páginas, de 2048 *bytes* cada uma.

Esta análise visa explorar dois aspectos importantes na avaliação distribuída de expressões de caminho: o impacto da variação do fator de seletividade dos predicados aninhados, tal que esta seletividade resulte na redução de fragmentos participantes da expressão de caminho; e a otimização dos requisitos de memória por nó nas estratégias de avaliação quando há compartilhamento de objetos. A variação de aspectos como a participação parcial das coleções nos relacionamentos e o agrupamento físico dos objetos apresenta, em cada nó de processamento, um comportamento semelhante ao observado no contexto centralizado.

5.6.1 VARIAÇÃO DA SELETIVIDADE

Nós variamos o fator de seletividade na coleção *CompositeParts*, diferentemente da simulação no contexto de bases unificadas onde essa seleção foi realizada na coleção *Connections*. Com isso, visamos simular a ocorrência de redução de fragmentos e de nós de processamento envolvidos na avaliação da expressão de caminho. Assumimos que o fator de seletividade dos predicados aninhados é igual em todos os fragmentos não eliminados. Foram aplicados os fatores de seletividade 1, 0.5, 0.1 e 0.01.

Como a redução de nós participantes do processamento ocorre devido à eliminação de fragmentos da coleção de partida, a existência de uma predicado aninhado definido sobre esta coleção tem um peso maior na escolha da melhor direção para avaliar uma expressão de caminho. Esse fato pode ser observado na Figura 19. Observe que, no nosso exemplo, as estratégias ascendentes (NP-A, VJ-A e PJ-A) não são sensíveis à eliminação de fragmentos ocorrida na coleção *CompositeParts*, pois a

coleção de partida nestas estratégias é a coleção *Connections*, a qual não sofre redução de fragmentos. Já nas estratégias descendentes baseadas em ponteiros (NP-D e PJ-D), o volume de páginas acessadas nos nós apresenta reduções significativas à medida que o fator de seletividade em *CompositeParts* diminui e, conseqüentemente, temos um número menor de fragmentos que participam da expressão de caminho.

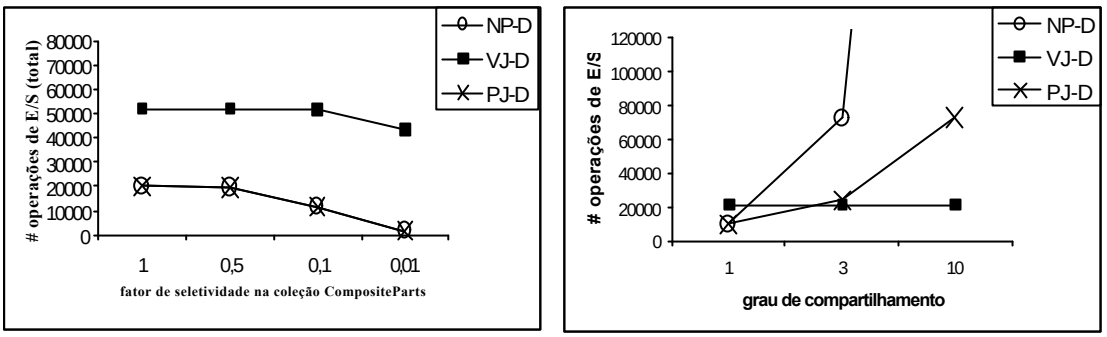


Figura 19. Redução de nós na avaliação distribuída de uma expressão de caminho

A estratégia VJ-D apresenta um desempenho muito ruim na avaliação distribuída de uma expressão de caminho, principalmente se a fragmentação das coleções envolvidas não for derivada da coleção de partida. Isto ocorre porque todos os objetos de cada coleção, com exceção da coleção de partida, são acessados (local ou remotamente) em cada nó. Entretanto, no nosso exemplo, verificamos que a estratégia VJ-A representa uma alternativa interessante, pois o alto grau de compartilhamento das coleções na direção ascendente degrada bastante o desempenho dos algoritmos baseados em ponteiros (NP-A e PJ-A).

5.6.2 VARIAÇÃO DA MEMÓRIA

Um aspecto importante da avaliação distribuída de expressões de caminho é que, de maneira geral, os algoritmos apresentam melhor desempenho com um tamanho menor de memória principal, pois o volume de páginas acessadas em cada nó diminui (LIMA e MATTOSO, 1996). Para analisarmos este aspecto, consideramos a expressão de caminho da Figura 20 tal que o grau de compartilhamento das coleções no sentido descendente é 3. Com isto, simularemos uma situação onde a memória principal não é suficiente para manter todas as páginas de dados solicitadas e há recarga de páginas (devido ao compartilhamento), o que prejudica sobremaneira o desempenho dos algoritmos.

Nós variamos a memória principal de cada nó entre 1 e 16 M*bytes*. Na Figura 20 é apresentado o desempenho da estratégia NP-D, cujo algoritmo é extremamente sensível ao grau de compartilhamento no caso de pouca memória. Com o aumento do número de fragmentos, verificamos que o desempenho local de cada nó responde mais rapidamente ao aumento da memória. Este efeito também ocorre na estratégia PJ-D e na estratégia VJ-A, conforme a Figura 21 e a Figura 22. É interessante observar que a distribuição torna possível a execução de consultas provavelmente inviáveis em um ambiente centralizado, como é o caso da estratégia NP-D no exemplo apresentado.

$$C_2$$

Figura 20. Efeito da distribuição nos requisitos de memória da estratégia NP-D

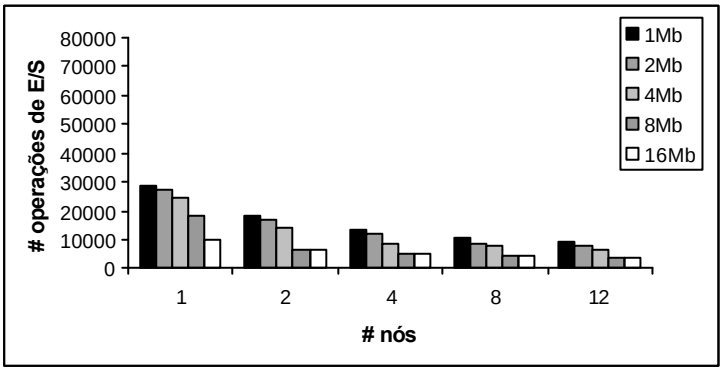


Figura 21. Efeito da distribuição nos requisitos de memória da estratégia PJ-D

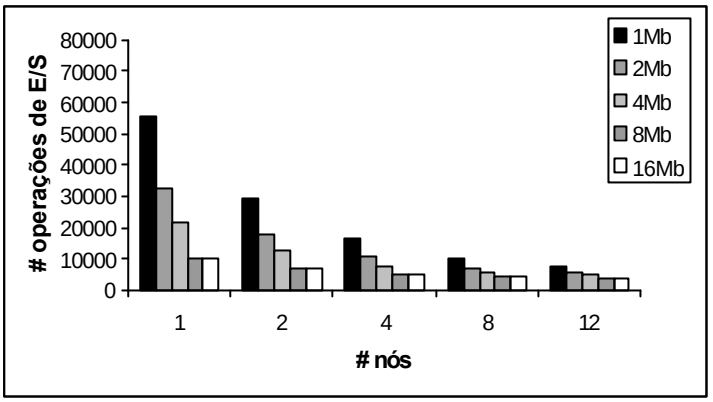


Figura 22. Efeito da distribuição nos requisitos de memória da estratégia VJ-A

Capítulo 6

CONCLUSÕES E TRABALHOS FUTUROS

O epílogo deste trabalho é apresentado neste capítulo, onde são reportadas as conclusões gerais e as contribuições desta tese, bem como algumas indicações para trabalhos futuros.

6.1 CONSIDERAÇÕES FINAIS

Não restam dúvidas sobre a importância do uso de técnicas de otimização para o processamento eficiente de consultas em bancos de dados baseados em objetos, principalmente na avaliação de expressões de caminho. Tais expressões representam travessias definidas sobre uma seqüência de relacionamentos entre coleções de objetos e têm sido o foco de inúmeras pesquisas associadas a tecnologias em destaque, como os Sistemas de Gerência de Bases de Dados Objeto-Relacionais e a linguagem XML.

A maioria dos trabalhos encontrados na literatura sobre técnicas de otimização de consultas considera a existência de um modelo de custo para identificar os melhores planos de execução para cada consulta, onde a predição de desempenho é obtida pela aplicação de fórmulas a parâmetros e estatísticas do banco de dados. Entretanto, os modelos de custo para bases de objetos propostos até então consideram cenários muito restritos, pois não tratam a participação parcial das coleções nos relacionamentos (OZKAN *et al*., 1996, GRUSER, 1996), limitam-se a apenas uma estratégia de avaliação de expressões de caminho (GARDARIN *et al*., 1995) e assumem políticas simplificadas de armazenamento dos objetos (BRAUMANDL *et al*., 2000, GARDARIN *et al*., 1996). Além disso, é notória a ausência de modelos de custo para o processamento de consultas em bases de objetos distribuídos, pois os poucos trabalhos existentes na literatura ignoram o efeito conjunto de diferentes técnicas de fragmentação das coleções, o custo de comunicação entre os nós, a política de armazenamento dos objetos e os diversos algoritmos para execução de consultas (EZEIFE e ZHENG, 1999, BELLATRECHE *et al*., 1998a, 1998b, FUNG *et al.*, 1997).

O ponto de partida para a nossa modelagem de custos foi o trabalho proposto em BELLATRECHE *et al*. (1998a) sobre o processamento de consultas em coleções fragmentadas horizontalmente, que embora enfoque uma idéia relevante para o contexto de bases de objetos distribuídos, apresenta um tratamento muito simplificado para o problema. Na revisão bibliográfica, percebemos que há pouco consenso sobre a separação clara entre o papel das classes e o das coleções na avaliação de uma expressão de caminho, o que dificulta a abordagem deste tema. Além disso, as propostas apresentadas na literatura adotam inúmeras nomenclaturas e representações sobre as possíveis estratégias de avaliação.

Inicialmente, nós utilizamos a fórmula proposta em GARDARIN *et al.* (1995) para estimar o número de objetos recuperados em cada coleção na avaliação de uma expressão de caminho. Entretanto, nós verificamos que os resultados gerados por tais estimativas apresentavam-se muitas vezes distantes dos valores esperados. A partir deste fato, nós levantamos as características desejáveis em um método para esse cálculo e, baseados nesse estudo, propomos uma nova alternativa para a estimativa da seletividade de uma expressão de caminho sobre as coleções envolvidas. Foram utilizadas as idéias lançadas em CHO *et al*. (1996) sobre a participação de coleções de objetos em relacionamentos.

Nós abordamos as implicações do agrupamento físico aninhado por referência (do tipo onde a coleção C_A agrupa C_Z, tal que C_Z é um agrupamento onde a coleção C_B agrupa C_C) na avaliação de expressões de caminho. Verificamos que este tipo de agrupamento acrescenta uma considerável complexidade ao cálculo de páginas ocupadas pelas coleções e à estimativa de páginas acessadas durante a travessia da expressão de caminho. Pudemos constatar neste caso a importância da validação experimental na análise de desempenho quando o problema modelado apresenta alta complexidade, visto que a abordagem matemática torna-se pouco prática e possui um alto custo computacional. Um tratamento matemático conhecido como "modelo de urnas" (GARDY e NÉMIROVSKI, 1999) estende a tradicional fórmula de Yao para o cálculo de páginas acessadas aleatoriamente (YAO, 1977) ao contexto de agrupamentos aninhados.

Na análise de custos de execução dos algoritmos, nós tomamos por base o modelo de custo apresentado em GARDARIN *et al*., 1996. Embora seja comum supor que o volume de páginas recarregadas durante a avaliação de uma expressão de caminho depende do grau de saída das coleções (OZKAN *et al.*, 1996, GARDARIN *et al.*, 1996), verificamos que, se não há compartilhamento de objetos, a variação do grau de saída não irá implicar na recarga de páginas durante a avaliação. Deste modo, utilizamos o grau de compartilhamento das coleções para estimar o número de páginas recarregadas quando a memória principal não é suficiente. Observamos também certos aspectos do modelo de dados OO sugerem algumas otimizações relevantes na execução de algoritmos típicos de SGBDs relacionais, como os de junção baseada em valor. Como o atributo de junção neste caso consiste em um identificador de objeto (IDO),

grande parte das operações podem ser realizadas em memória, dispensando a releitura de páginas do disco.

BELLATRECHE *et al*. (1998a) abordam os custos do processamento de consultas na fragmentação horizontal primária das coleções, cujo modelo de custo é utilizado em BELLATRECHE *et al*. (1998b) para a fragmentação horizontal derivada. A única diferença deste último trabalho em relação ao outro é que se considera a eliminação de um fragmento quando o respectivo fragmento primário é eliminado por um predicado aninhado. Esta eliminação de fragmentos, entretanto, não está claramente definida. Nós apresentamos uma definição formal dos fragmentos eliminados na redução de uma expressão de caminho especificada sobre coleções de objetos distribuídos, considerando a fragmentação horizontal primária, horizontal derivada e vertical. Além disso, verificamos que, se há redução de fragmentos horizontais, a seletividade de uma expressão de caminho nos fragmentos horizontais derivados correspondentes deve ser acrescida do percentual que os fragmentos eliminados representam.

Por fim, realizamos uma validação de resultados gerados pelo modelo de custo proposto através de comparações com resultados experimentais encontrados em TAVARES (1999). Em uma análise complementar, exploramos algumas funcionalidades do modelo através de simulações sobre configurações sintéticas do *benchmark* OO7, variando características relevantes para a avaliação de expressões de caminho em bases de objetos distribuídos. É importante ressaltar que, durante a validação experimental, muitas vezes o modelo de custo ajudou a complementar a análise do trabalho prático e permitiu realçar aspectos relevantes antes não considerados.

6.2 CONTRIBUIÇÕES DESTA TESE

A principal contribuição desta tese consiste em um modelo de custo que permite estimar o desempenho das principais estratégias de avaliação de expressões de caminho em bases de objetos distribuídos. Foram especificadas métricas que representam o custo em tempo total e em tempo de resposta das operações de entrada/saída (E/S) de dados, de CPU e de comunicação entre nós, referentes à avaliação de uma expressão de

caminho. A abrangência do modelo proposto diferencia esta tese dos demais trabalhos apresentados na literatura. São considerados aspectos relevantes do (i) modelo de dados OO, como a seletividade de expressões de caminho e a existência de relacionamentos parciais, (ii) do modelo de armazenamento de dados segundo diferentes políticas para agrupamento dos objetos, (iii) o modelo de execução dos algoritmos e (iv) do projeto de distribuição das coleções, levando em conta diferentes técnicas de fragmentação e a alocação dos fragmentos.

Além disso, entre as contribuições decorrentes do desenvolvimento desta tese, podemos citar a caracterização e a representação de expressões de caminho, o estudo da concepção de modelos de custo para o processamento de consultas, o tratamento de relacionamentos parciais entre coleções de objetos, a análise de técnicas de agrupamento aplicadas a coleções fragmentadas e o levantamento dos algoritmos típicos para a avaliação de expressões de caminho.

O modelo de custo proposto nesta tese apresenta inovações e possui uma ampla abrangência, pois além de reunir características presentes em modelos existentes na literatura (CHO *et al*., 1996, OZKAN *et al*., 1996, GARDARIN *et al*., 1995, 1996, BERTINO e FOSCOLI, 1997, BELLATRECHE *et al*., 1998a, 1998b), apresenta novas métricas para a análise de desempenho do processamento de consultas em bases de objetos distribuídos. Com as fórmulas elaboradas nesta tese, é possível examinar o impacto conjunto de técnicas de fragmentação horizontal primária, horizontal derivada e vertical, antes analisadas individualmente e de maneira muito simplificada. Uma outra novidade relevante é a estimativa dos custos de comunicação na avaliação distribuída de expressões de caminho, considerando a alocação dos fragmentos das coleções envolvidas.

Este trabalho está inserido na linha de pesquisa sobre o processamento de consultas em bases de objetos que vem sendo desenvolvida na COPPE/UFRJ, da qual resultaram um protótipo de SGBD baseado em objetos (MAURO *et al*., 1997), vários trabalhos sobre a análise experimental de desempenho (MATTOSO, 1993, LIMA e MATTOSO, 1996, MEYER, 1997, TAVARES, 1999, OSTHOFF *et al*., 2000, SOARES, 2000, OGASAWARA, 2000, LIMA, 2000) e uma importante metodologia para projeto de distribuição de objetos (BAIÃO *et al*., 2000). O modelo de custo aqui apresentado contribui para o aprofundamento da análise sobre expressões de caminho.

6.3 TRABALHOS FUTUROS

Um modelo de custo pode se tornar muito complexo, de acordo com o número e a abrangência dos fatores considerados. No contexto do processamento de consultas em bases de objetos distribuídos, o universo de aspectos a serem tratados é muito amplo. Neste sentido, alguns pontos importantes podem ser explorados como trabalhos futuros sobre o modelo de custo proposto, como os custos da execução de métodos, o impacto de técnicas de replicação de dados, o acesso às coleções segundo diferentes estruturas de índices e o tratamento de objetos longos.

Uma possível continuidade da proposta desta tese é a implementação das funções de custo apresentadas em otimizadores de consultas como o OPT++ (KABRA e DEWITT, 1996) e o otimizador do *lambda-DB* (FEGARAS, 1999). Ou ainda, a geração de um otimizador de consultas através da ferramenta OPTGEN (FEGARAS, 1998) para um SGBD baseado em objetos, como o GOA++ (MAURO *et al*., 1997) e o PARGOA (TAVARES, 1999), cujo processo de otimização seja orientado pelas métricas de custo desta tese.

Na linha de distribuição de objetos, pode-se integrar as funções de custo aqui apresentadas a uma ferramenta de auxílio ao projeto de distribuição, visando enriquecer os mecanismos de escolha das melhores configurações de esquema em metodologias como a proposta em BAIÃO *et al*. (2000).

Existem várias outras possibilidades para o desenvolvimento de trabalhos futuros a partir das métricas de custo geradas nesta tese. Por exemplo, a implementação de uma ferramenta de auxílio ao projeto físico de banco de dados, a qual gere estimativas como o volume ocupado pelas coleções e os requisitos de memória do *cache* de um SGBD. No contexto da análise de desempenho, o modelo de custo proposto pode ser explorado para avaliar técnicas para agrupamento físico de objetos e a aplicação de estratégias híbridas para a avaliação de expressões de caminho.

Por fim, uma continuação interessante para o nosso trabalho seria a extensão do modelo de custo proposto para o tratamento de dados semi-estruturados, considerando as características próprias das expressões de caminho descritas pela linguagem XML.

REFERÊNCIAS BIBLIOGRÁFICAS

AILAMAKI, A., DEWITT, D.J., HILL, M.D., *et al*., 1999, "DBMSs On A Modern Processor: Where Does Time Go?". In: *Proceedings of 25th VLDB Conference*, pp. 266-277, Escócia, Setembro.

BAIÃO, F., 1997, *Uma Estratégia para o Projeto de Distribuição de Bases de Dados Orientados a Objetos*, Tese de M.Sc., COPPE/UFRJ, Brasil.

BAIÃO, F., MATTOSO, M., 1998, "A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases". In: *International Conference on Computing and Information (ICCI'98)*, pp.141-148, Winnipeg, Junho.

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 1998, "Towards an Inductive Design of Distributed Object Oriented Databases ". *Third International Conference on Cooperative Information Systems,* New York, USA, Agosto.

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 1999, "A Theory Refinement Approach to the Design of Distributed Object Oriented Databases", Relatório Técnico ES-493/99, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Brasil, Abril.

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 2000, "Horizontal Fragmentation in Object DBMS: New Issues and Performance Evaluation". In: *Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference (IPCCC 2000)*, pp. 108-114, Phoenix, Arizona, Fevereiro.

BARROSO, M., 1998, *PROJETOO: Uma Metodologia para Projeto de Banco de Dados Distribuídos Orientado a Objetos*, Tese de M.Sc., Departamento de Informática, CCEN/UFPE, Brasil.

BELLATRECHE, L., KARLAPALEM, K., e BASAK, G., 1998a, "Query-Driven Horizontal Class Partitioning for Object-Oriented Databases", In: *Lecture Notes in Computer Science (N° 1460) of Ninth International DEXA Conference*, pp. 692-701, Áustria, Agosto.

BELLATRECHE, L., KARLAPALEM, K., Li, Q., 1998b, "Derived Horizontal Class Partitioning in OODBs: Design Strategies, Analytical Model and Evaluation", In: *Lecture Notes in Computer Science (N° 1507) of 17th International Conference on the Entity Relationship Approach ER98*, pp 465-479, Cingapura, Novembro.

BENZAKEN, V., 1990, "An Evaluation Model for Clustering Strategies in the O2 Object-Oriented Database System", In: *Proceedings of the Third International Conference on Database Theory (ICDT90)*, pp. 126-140, Paris, Dezembro.

BERTINO, E., FOSCOLI, P., 1995, "Index Organizations for Object-Oriented Database Systems", *IEEE Transactions on Knowledge and Data Engineering*, v. 7, n. 2, pp. 193-209, Abril.

BERTINO, E., FOSCOLI, P., 1997, "On Modeling Cost Functions for Object-Oriented Databases", *IEEE Transactions on Knowledge and Data Engineering*, v. 9, n. 3, pp. 500-508, Maio/Junho.

BOULOS, J., ONO, K., 1999, "Cost Estimation of User-Defied Methods in Object-Relational Database Systems", *ACM SIGMOD Record*, v. 25, n. 3, pp. 22-28, Setembro.

BRAUMANDL, R., CLAUSSEN, J., KEMPER, A., 1998, "Evaluating Functional Joins along Nested Reference Sets in Object-Relational and Object-Oriented Databases", In: *Proceedings of the 24th VLDB Conference*, pp. 110-122, New York, Agosto.

BRAUMANDL, R., CLAUSSEN, J., KEMPER, A., *et al*., 2000. "Functional Join Processing", *The VLDB Journal*, n. 8, n. 3-4, pp. 156-177, Springer-Verlag, Fevereiro.

CAREY, M., DEWITT, D., NAUGHTON, J., 1993, "The 007 Benchmark", In: P*roceedings of the 1993 ACM SIGMOD International Conference on Management of Data,* v. 22, n. 2, Washington, pp.12-21, Maio.

CAREY, M., DEWITT, D., KANT, C., NAUGHTON, J., 1994, "A Status Report on the OO7 OODBMS Benchmarking Effort", In: *ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA),* pp. 414-426, Dezembro.

CASANOVA, M., MOURA, A., 1985, *Princípios de Sistemas de Gerência de Bancos de Dados Distribuídos*. 1ª ed. Rio de Janeiro, Editora Campus.

CATTELL, R., 1994, *Object Data Management*. Revised edition, Addison-Wesley Publishing Company.

CATTELL, R., BARRY, D., BERLER, M., *et al.*, 2000, *The Object Data Standard: ODMG 3.0*. First ed. San Francisco, Morgan Kaufmann Publishers Inc.

CLAUSSEN, J., KEMPER, A., KOSSMANN, D., *et al.*, 2000, "Exploiting early sorting and early partitioning for decision support query processing", The VLDB Journal, v. 9, n. 3, pp. 190-213.

CLUET, S., DELOBEL, C, 1993, "Towards a Unification of Rewrite-based Optimization Techniques for Object-oriented Queries". In: *Query Processing for Advanced Database Systems* (Ed. by J. Freitag, D. Maier and G. Vossen). Morgan Kaufmann Publishers, Califórnia, USA, pp. 246-272.

CHAKRAVARTHY, S., MUTHURAJ, J., VARADARAJAN, R, *et al*., 1994, "An Objective Function for Vertically Partitioning Relations in Distributed Databases and Its Analysis", *Distributed and Parallel Databases*, v. 2, n. 2, pp. 183-207, Abril.

CHO, W., PARK, C., WHANG, K., SON, S., 1996. "A New Method for Estimating the Number of Objects Satisfying an Object-Oriented Query Involving Partial Participation of Classes", *Pergamon Information Systems,* v. 21, n. 3, pp. 253-267, Maio.

DARMONT, J., SCHNEIDER, M., 1999, "VOODB: A Generic Discrete-Event Random Simulation Model to Evaluate the Performances of OODBs". In: *Proceedings of 25th VLDB Conference*, pp. 254-265, Escócia, Setembro.

DEUTSCH, A., FERNANDEZ, M., FLORESCU, D., *et al.*, 1999. "Querying XML Data", In: *Bulletin of IEEE Computer Society Technical Committee on Data Engineering*, v. 22, n. 3, pp. 10-18, Setembro.

DEWITT, D., MAIER, D., FUTTERSACK, P., *et al*., 1990, "A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems". In: *Proceedings of 16th VLDB Conference*, pp. 107-121, Austrália.

DEWITT, D., LIEUWEN, D., MEHTA, M., 1993, "Pointer-based Join Techniques for Object-Oriented Databases", In: *Proceedings of the International IEEE Conference on Parallel and Distributed Information Systems (PDIS93)*, pp. 172-181, San Diego, USA, Janeiro.

DEWITT, D., NAUGHTON, J., SHAFFER, J., *et al*., 1996, "Parallelizing OODBMS traversals: a performance evaluation", In: *The VLDB Journal*, n. 5, pp. 3-18, Springer-Verlag.

EISENBERG, A., MELTON, J., 1999, "SQL:1999, formerly known as SQL3", *ACM SIGMOD Record*, v. 28, n. 1, pp. 131-138, Março.

ELMASRI, R., NAVATHE, S., 1994, *Fundamentals of Database Systems.* Second ed. California, Addison-Wesley.

EZEIFE C.I., BARKER K., 1994, "Vertical Class Fragmentation in a Distributed Object-Based System", v. 2, n. 1, pp. 43-52, In: *Proceedings of the Second International Symposium on Applied Corporate Computing (ISACC94)*,Texas A& M University (publisher), Monterrey.

EZEIFE, C.I., ZHENG, J., 1999, "Measuring the Performance of Database Object Horizontal Fragmentation Schemes", In: *Proceedings of the International Database Engineering and Applications Symposium (IDEAS99)*, pp. 408-414, Montreal, Canadá, Agosto.

FEGARAS, L., e MAIER, D., 1995, "Towards an Effective Calculus for Object Query Languages". In: *Proceedings of the International ACM SIGMOD International Conference on Management of Data*, pp. 47-58, USA, Maio.

FEGARAS, L., 1998, "The OPTGEN Optimizer Generator". In: http://ranger.uta.edu/~fegaras/optimizer. *Department of Computer Science and Engineering, The University of Texas at Arlington*.

FEGARAS, L., 1999, "lambda-DB: System Overview". In: http://lambda.uta.edu/lambda-DB/manual/system.html. *Department of Computer Science and Engineering, The University of Texas at Arlington*.

FIEBIG, T., MOERKOTTE, G., 2000, "Evaluating Queries on Structure with Extended Access Support Relations". In: *Proceedings of the Third International Workshop on the Web and Databases*, Dallas, USA, Maio.

FUNG, C., KARLAPALEM, K., LI, Q., 1997. "Cost-driven evaluation of vertical class partitioning in object oriented databases". In: *Fifth International Conference On Database System For Advanced Applications (DASFAA97)*, pp. 11-20, Melbourne, Austrália, Abril.

GARDARIN, G., GRUSER, J., TANG, Z., 1995, "A Cost Model for Clustered Object-Oriented Databases". In: *Proceedings of 21st VLDB Conference*, pp. 323-334, Zurich, Suiça, Setembro.

GARDARIN, G., GRUSER, J.R., TANG, Z.T., 1996, "Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases" , In: *Proceedings of the 22nd VLDB Conference*, pp. 390-401, Bombay, Índia, Setembro.

GARDY, D., NÉMIROVSKI, L., 1999, "Urn models and Yao's formula", In: *Seventh International Conference on Database Theory*, Jerusalém, Israel, Janeiro.

GRUSER, J., 1996, *Modèles de coût pour l'optimisation de requêtes objet*, Tese de D.Sc., Université Pierre & Marie Curie – Paris IV, Paris, França.

HAAS, L.M., CAREY, M.J., LIVNY, M., *et al*., 1993, *SEEKing the Truth about Ad Hoc Join Costs.* Technical Report RJ9368, IBM Almaden Research Center, USA.

HARRIS, E.P., RAMAMOHANARAO, K., 1996. "Join algorithm costs revisited", *The VLDB Journal*, v. 5, n. 1, pp. 64-84, Springer-Verlag, Janeiro.

INFORMIX CORPORATION, 1999, *Informix Extended Parallel Server 8.3*, Manual nº 000-21924-70, Informix Corporation, USA, Outubro.

JORDAN, D., 1998, *C++ Object Databases: programming with the ODMG standard*. First ed. Massachussetts, Addison Wesley Longman, Inc.

KABRA, N., e DEWITT, D.J., 1999, "OPT++: An Object-Oriented Implementation for Extensible Database Query Optimization" , *The VLDB Journal*, n. 8, n. 1, pp. 55-78, Springer-Verlag, Fevereiro.

KELLER, T., GRAEFE, G., MAIER, D., 1991, "Efficient Assembly of Complex Objects", In: *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 148-157, Colorado, USA, Maio.

KEMPER, A., MOERKOTTE, G., 1995, "Physical Object Management", In: *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Kim, W. (ed.), Addison-Wesley Publishing Company, pp. 175-202.

KOSSMANN, D., 2000, "The State of the Art in Distributed Query Processing", http://www.db.fmi.uni-passau.de/~kossmann (último acesso, Março/2001).

LEE, W.-C., LEE, D.L., 1998. "Path Dictionary: A New Access Method for Query Processing in Object-Oriented Databases", *IEEE Transactions Knowledge and Data Engineering*, v. 10, n. 3, pp. 371-388, Maio/Junho.

LEVERENZ, L., REHFIELD, D., BAIRD, C., 1999, *Oracle8i Concepts – Release 2 (8.1.6)*, Manual nº A76965-01, Oracle Corporation.

LIMA, B., 2000, *Índices Espaciais e Paralelismo no Processamento de Junções Espaciais Utilizando o GOA++*, Tese de M.Sc., COPPE/UFRJ, Brasil.

LIMA, F., MATTOSO, M., 1996, "Performance Evaluation of Distribution in OODBMS: a Case Study with O2". In: *Proceedings of the IX International Conference on Parallel and Distributed Computing Systems (PDCS96)*, pp. 720-726, França.

MAIER, D., DANIELS, S., KELLER, T., *et al*., 1993, "Challenges for Query Processing in Object-oriented Databases". In: *Query Processing for Advanced Database Systems (Ed. by J. Freitag, D. Maier and G. Vossen)*, Morgan Kaufmann Publishers, Califórnia.

MANEGOLD, S., BONCZ, P., KERSTEN, M.L., 2000, "What happens during a Join? Dissecting CPU and Memory Optimization Effects". In: *Proceedings of the 26th VLDB Conference*, pp. 339-350, Cairo, Egito, Setembro.

MATTOSO, M.Q.L., 1993, *Aspectos de Paralelismo na Gerência de Dados e Objetos no Geotaba*, Tese de D.Sc., COPPE/UFRJ, Brasil.

MAURO, R., ZIMBRÃO, G., BRUGGER, T., *et al.*, 1997, "GOA++: Tecnologia, Implementação e Extensões aos Serviços de Gerência de Objetos", In: XII Simpósio Brasileiro de Banco de Dados (SBBD1997), Fortaleza, Brasil, pp. 272-286, Outubro.

MENDES, S.F., SAMPAIO, P.R.F., 1998. "Rule-Based Parallel Query Optimization for OQL Using a Parallelism Extraction Technique", In: *Proceedings of 9th International Workshop on Database and Expert Systems Applications (DEXA98)*, pp. 705-710, Viena, Áustria, Agosto.

MEYER, L.V.C., 1997, *Paralelismo em SGBDOO com Memória Distribuída: Uma Implementação no PARGOA*, Tese de M.Sc., COPPE/UFRJ, Brasil.

MISHRA, P., EICH, M., 1992, "Join Processing in Relational Databases", *ACM Computing Surveys*, v. 24, n. 1, pp. 63-113, Março.

NICOLA, M., JARKE, M., 2000, "Performance Modeling of Distributed and Replicated Databases", *IEEE Transactions on Knowledge and Data Engineering*, v. 12, n. 4, pp. 645-672, Julho/Agosto.

OGASAWARA, E., MATTOSO, M., 1999, *Uma Análise de Índices em Bancos de Dados Orientados a Objetos*, Relatório Técnico ES-497/99, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Brasil, Abril.

OGASAWARA, E., 2000, *Implementação e Avaliação de Índices para Orientação a Objetos*, Tese de M.Sc., COPPE/UFRJ, Brasil.

OSTHOFF, C., MATTOSO, M., SEIDEL, C., *et al*., 2000, *Avaliação do Algoritmo de Coerência de Cache de Disco DSMIO*, In: XV Simpósio Brasileiro de Banco de Dados (SBBD2000), João Pessoa, Brasil, pp. 273-286, Outubro.

ÖZSU, M., BLAKELEY, J., 1995, "Query Processing in Object-Oriented Database Systems", In: *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Addison-Wesley Publishing Company, pp. 146-174.

ÖZSU, M., VALDURIEZ, P., 1999, *Principles of Distributed Database Systems.* Second ed. New Jersey, Prentice-Hall.

OZKAN, C., DOGAC, A., ALTINEL, M., 1996. "A Cost Model for Path Expressions in Object-Oriented Queries", *Journal of Database Management*, v. 7, n. 3, pp. 25-33.

O_2 TECHNOLOGY, 1996, *The O_2 System Administration Guide – Release 5.0*, O_2 Technology.

PIATETSKY-SHAPIRO, G., CONNELL, C., 1984, "Accurate Estimation of the Number of Tuples Satisfying a Condition". In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 256-276, Boston, USA, Junho.

RAMASAMY, K., PATEL, J., NAUGHTON, J., KAUSHIK, R., 2000, "Set Containment Joins: The Good, The Bad and The Ugly". In: *Proceedings of the 26th VLDB Conference*, pp. 351-362, Egito, Setembro.

REIS, R., CAMPOS, M., 2000, "Modelo de Custos para Seleção Dinâmica de Agregados a Materializar em Data Warehouses", In: XV Simpósio Brasileiro de Banco de Dados (SBBD2000), pp. 331-345, João Pessoa, Brasil, Outubro.

ROTH, M. T., ÖZCAN, F., HAAS, L. M., 1999, "Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System". In: *Proceedings of the 25th VLDB Conference*, pp.599-610, Edinburgh, Escócia, Setembro.

RUBERG, G., BAIÃO, F., MATTOSO, M., VICTOR, A., 2001a, "Processamento de Consultas Orientadas a Objetos", Relatório Técnico ES-547/01, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Brasil, Abril.

RUBERG, G., BAIÃO, F., MATTOSO, M., VICTOR, A., 2001b, "A Cost Model for the Evaluation of Path Expressions in Object Databases", In: submetido a *12th International DEXA Conference*, Munich, Alemanha, Setembro.

SHEKITA, E., CAREY, M., 1990, "A Performance Evaluation of Pointer-Based Joins", In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pp. 300-311, Atlantic City, USA, Junho.

SHRUFI, A., 1994, "Performance of Clustering Policies in Object Bases", In: *Proceedings of the Third Conference of Information and Knowledge Management (CIKM94)*, pp. 80-87, Gaithersburg, USA, Dezembro.

SOARES, J., 2000, *Análise de Desempenho da Gerência Paralela de Objetos Armazenados*, Tese de M.Sc., COPPE/UFRJ, Brasil.

STRINGER, G., SIMMONDS, J., LIEF, E., 1996, *Sybase MPP Administration Guide – Release 11.0.x*, Manual nº 35377-01-1100-03, Sybase Inc., Califórnia, EUA.

SU, S., RANKA, S., HE, X., 2000, "Performance Analysis of Parallel Query Processing Algorithms for Object-Oriented Databases", *IEEE Transactions on Knowledge and Data Engineering*, v. 12, n. 6, pp. 979-997, Novembro/Dezembro.

TAVARES, F., 1999, *Avaliação do Processamento Paralelo de Consulta no Modelo Orientado a Objetos*, Tese de M.Sc., COPPE/UFRJ, Brasil.

TAVARES, F., VICTOR, A., MATTOSO, M., 2000, “Parallel Processing Evaluation of Path Expressions”, In: XV Simpósio Brasileiro de Banco de Dados (SBBD2000), João Pessoa, Brasil, pp. 49-63, Outubro.

YAO, S., 1977, “Approximating the Number of Accesses in Database Organizations”, *Communications of the ACM*, v. 20, n. 4, pp. 260-261.

YAO, S., 1979, “Optimization of Query Evaluation Algorithms”, In: *ACM Transactions on Database Systems* (TODS), v.4, n.2, pp. 133-155, Junho.

ZHU, Q., LARSON, P.-A., 1998, “Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems”, *Distributed and Parallel Databases*, v. 6, n.4, pp. 373-421, Outubro.

ANEXO I – FÓRMULA DE YAO

YAO (1977) propõe uma fórmula clássica para estimar o número de blocos acessados quando um certo número de elementos são selecionados, a qual é amplamente utilizada na predição de desempenho do processamento de consultas (DEWITT *et al*., 1993, CHO *et al*., 1996, GARDARIN *et al*., 1995, GRUSER, 1996, GARDARIN *et al*., 1996, BELLATRECHE *et al.*, 1998). A fórmula de Yao diz que, dados ***n*** registros uniformemente distribuídos em ***m*** blocos ($1 < m \leq n$), cada um contendo ***n/m*** registros, se ***k*** registros ($k \leq n$) forem selecionados aleatoriamente, o número esperado de blocos acessados será dado por:

$$Yao(n,m,k) = m \times \left[1 - \prod_{i=1}^{k} \frac{nd - i + 1}{n - i + 1} \right], \tag{123}$$

$$\text{onde} \quad d = 1 - \frac{1}{m}. \tag{124}$$

No nosso modelo de custo, nós utilizamos a fórmula de Yao para estimar o número de páginas acessadas quando um certo número de objetos são selecionados aleatoriamente.