ALOCAÇÃO DINÂMICA DE OBJETOS EM SISTEMAS DE BANCOS DE DADOS PARALELOS

Marco Antônio Ferreira Duran

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 OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof: Geraldo Bonorino Xexéo, D.Sc.

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

Prof : Josefino Cabral Melo Lima, D.Sc.

RIO DE JANEIRO, RJ –BRASIL
MARÇO DE 1999

DURAN, MARCO ANTÔNIO FERREIRA

Alocação Dinâmica de Objetos em Sistemas de Bancos de Dados Paralelos [Rio de Janeiro] 1999

VIII, 105 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1.Distribuição e Paralelismo

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.)

ALOCAÇÃO DINÂMICA DE OBJETOS EM SISTEMAS DE BANCOS DE DADOS PARALELOS

Marco Antônio Ferreira Duran

Março/1999

Orientador: Prof.° Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Em modelos de distribuição de Sistemas de Banco de Dados Paralelos, informações a respeito do acesso à base de dados constituem importantes requisitos na execução de estratégias de distribuição. Neste sentido, foi implementado um gerente de distribuição que registra informações de acesso à base e sugere uma nova distribuição dos dados.

Aspectos que afetam a alocação de dados em sistemas de arquitetura de memória distribuída são abordados nesta tese. A implementação do identificador de objetos, a troca de informações entre os nós e a distribuição de carga foram analisados neste trabalho.

A implementação inclui uma estratégia de distribuição que se baseia na freqüência de ocorrência de consultas, tempos de execução, de processamento, de entrada e saída, de comunicação na rede, e nos atributos utilizados em seus predicados.

As configurações sugeridas pelo sistema são comparadas com uma fragmentação horizontal distribuída estaticamente e outra alocação onde a base é dividida em um número muito grande de fragmentos.

Os testes foram realizados em uma rede de estações de trabalho POWERPC no laboratório da COPPE/UFRJ. O software PVM foi utilizado para configuração da máquina virtual paralela. O benchmark OO7 foi utilizado como modelo de análise de desempenho.

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

DYNAMIC ALLOCATION OF OBJECTS IN PARALLEL DATABASE SYSTEMS

Marco Antônio Ferreira Duran

March/1999

Advisor: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

In Distribution models of Parallel Database Systems, access information constitute important requirements related to distribution strategies execution. Therefore, a distribution management system has been implemented in order to suggest a database reallocation based on data access information.

This thesis deals with data allocation in the Shared-Nothing model, in what concerns the object identifier implementation, information exchange among the system nodes and load balancing.

A declustering strategy was adapted and it is also part of this implementation. This strategy is based on the following aspects: frequency execution, CPU, Input-Output and communication time, and the attributes used in query's predicate.

The configurations created by the system are compared not only to a horizontal fragmentation and static distribution, but also to another allocation where the database is declustered into a great number of fragments.

The tests took place in a POWERPC workstation network, located in COPPE/UFRJ laboratory. The parallel machine was configured by the PVM software and the benchmark OO7 was used in the performance analysis.

ÍNDICE

Capítulo I

INTRODUÇÃO

I.1 MOTIVAÇÃO

Um projeto de distribuição define como será feito o armazenamento dos dados entre os nós da rede do sistema distribuído. Segundo (ÖSZU, VALDURIEZ, 1991a), uma base de dados pode ser fragmentada ou replicada pelos nós da rede. A replicação de dados almeja dar maior confiabilidade ao sistema, no caso de falhas. A fragmentação dos dados visa à economia de recursos, evitando acessos desnecessários na consulta à base de dados. (LIMA, MATTOSO, 1996) conseguiu ganhos significativos de desempenho, fragmentando a base de dados OO7 (CAREY *et al.*, 1994) no SGBDOO O_2. Em seu trabalho, também foram abordadas questões relacionadas à realocação dos objetos.

Outra característica de divisão da base é a possibilidade de utilização de processamento paralelo na execução de consultas, aumentando o desempenho do sistema. Porém, alguns trabalhos demonstraram que o projeto de distribuição deve ser analisado com atenção. (BAIAO, 1997) propõe uma estratégia para distribuição de bases de dados orientada a objetos. Em seu trabalho, foi construído uma ferramenta que auxilia o projetista de banco de dados. A ferramenta consiste em uma interface gráfica onde são entrados os dados necessários à estratégia definida. Em poder destes dados, a estratégia é executada formando um roteiro de distribuição da base de dados OO. Entre as informações pedidas pela ferramenta, estão os tipos de consultas solicitadas além de suas freqüências de ocorrência. Informações como estas são referenciadas por outros trabalhos da literatura, como em (EZEIFE, BARKER, 1995) (KARLAPALEM, *et al.*, 1994), (CERI, *et al.*, 1987).

I.2 Objetivo da Tese

Neste sentido, o objetivo principal desta tese é criar uma ferramenta capaz de sugerir uma nova distribuição da base de dados para aumentar o desempenho de um sistema de banco de dados **paralelo**, ora aumentando o número de fragmentos da base ou mudando a forma de distribuição. Além disso a ferramenta deverá ser capaz de alocar os dados automaticamente. Os estudos foram desenvolvidos em ambientes paralelos com modelo de base de dados Orientado a Objetos, porém a estratégia adota veio do modelo relacional e poderia ser utilizada também em SGBDs relacionais paralelos.

Desta forma, devem ser analisados aspectos como as estratégias disponíveis, a realocação dos dados e o problema da distribuição de carga dentro do sistema. Estes temas são abordados nos capítulos II e III desta tese.

Na linha de pesquisa de Banco de Dados da COPPE/UFRJ vários trabalhos foram desenvolvidos utilizando as vantagens do processamento paralelo (MATTOSO, 1993) (TAVARES, *et al.*, 1996) (MEYER, 1997). Objetivando seguir esta tendência a ferramenta foi implementada para um sistema de banco de dados paralelo.

Além de apresentar a implementação de um gerente de distribuição, esta tese analisa as possíveis realocações sugeridas pelo sistema, comparando-as com outras configurações criadas manualmente. Os resultados mostraram que, na maioria das consultas utilizadas como testes, as configurações sugeridas resultaram em ganhos super-lineares e superiores às configurações manualmente criadas para comparação.

I.3 Estrutura da Tese

O capítulo II compõe a referência bibliográfica desta tese. Esta aborda as principais áreas de atuação em Banco de Dados Distribuídos. Trata também o aspecto do projeto de distribuição de bases de dados e as características principais de sistemas paralelos de bancos de dados.

No capítulo III são abordados diversos tipos de estratégias de distribuição de dados, segundo o aspecto de distribuição de carga.

No capítulo IV é mostrada a estratégia utilizada no sistema de distribuição construído nesta tese. São apresentadas as principais características da estratégia e as

adaptações que foram realizadas. O Anexo I inclui o algoritmo da estratégia em português estruturado.

O capítulo V descreve a implementação do gerente de distribuição dinâmico. Também apresenta os tipos de consultas, como foi feita a análise de desempenho, o software utilizado para configuração da máquina paralela e o modelo de dados usado na avaliação.

O capítulo VI apresenta os resultados dos testes realizados e faz uma análise entre as diversas configurações sugeridas pelo sistema. O estilo dos gráficos é em formato de colunas. Foram realizados testes com o sistema inicializado e com o sistema se beneficiando do buffer de páginas. A última seção do capítulo VI contém considerações finais sobre os resultados.

Finalmente, o capítulo VII apresenta as conclusões deste trabalho, indicando realizações futuras.

Capítulo II

BANCOS DE DADOS DISTRIBUÍDOS E PARALELOS

II.1 INTRODUÇÃO

A tecnologia de banco de dados distribuídos tem como objetivo integrar de forma eficiente e transparente a tecnologia de banco de dados à de processamento distribuído. A contribuição do processamento distribuído para a tecnologia de banco de dados está na descentralização das tarefas, aplicações e armazenamento de dados, sem, contudo, perder a integração e a transparência. Segundo (MATTOSO, 1993), a contribuição do processamento distribuído para banco de dados está no acesso paralelo ao armazenamento das bases de dados e no processamento paralelo desses dados. A integração de bases de dados distintas, gerenciadas por SGBDs diferentes (PIRES, MATTOSO, 1996) através de padrões (SOLEY, KENT, 1995) constitui outra contribuição da tecnologia. A seção II.2 apresenta as características gerais dos SGBDs distribuídos, suas vantagens e desvantagens e as principais áreas de atuação.

Atualmente é grande a pesquisa na área de projeto distribuído de base de dados no modelo orientado a objetos (KARLAPALEM, *et al.*, 1994) (LIMA, MATTOSO, 1995). A fragmentação da base pode influenciar o desempenho do sistema, tanto positivamente como negativamente. A princípio, existem trabalhos desenvolvidos que abordam os principais aspectos de distribuição (ÖZSU, VALDURIEZ, 1991b). BAIÃO e MATTOSO (1998) apresentam várias estratégias de distribuição de dados em bancos de dados orientados a objetos. ÖZSU e VALDURIEZ (1991a) apresentam os principais aspectos de projeto distribuído em bases de dados relacionais.

Outros trabalhos já propõem algoritmos e estratégias para a distribuição da base de dados (EZEIFE, BARKER, 1995). Sendo assim, a seção II.3 aborda essa área de pesquisa mostrando conceitos, metodologias e trabalhos atuais desenvolvidos.

O paralelismo é abordado na seção II.4, mostrando as principais caraterísticas, arquiteturas, divisão dos dados e execução de consultas.

II.2 CARACTERIZAÇÃO DE SGBDs DISTRIBUÍDOS

(ÖSZU, VALDURIEZ, 1991a) define bases de dados distribuídas como uma coleção de múltiplas bases de dados, logicamente relacionadas e divididas sobre uma rede de computadores. Um Sistema Gerenciador de Banco de Dados Distribuídos é aquele que viabiliza a gerência da base de dados e proporciona uma transparência de distribuição para o usuário.

Uma coleção de arquivos espalhados por uma rede de computadores cujo acesso e transações são controlados por um sistema de arquivos distribuído não pode ser considerado um SGBDD (ÖSZU, VALDURIEZ, 1991a). Um SGBDD deverá conter toda a funcionalidade que um sistema de banco de dados possui, além de proporcionar ao usuário acesso transparente às diversas bases distribuídas (ÖSZU, *et al.*, 1994).

Outro conceito, mais amplo e genérico, define um sistema de banco de dados distribuído com sendo um conjunto de SGBDs federados ou independentes, que têm a capacidade de trocar serviços e informações entre si (MOLINA, HSU, 1995).

II.2.1 VANTAGENS E DESVANTAGENS

Sistemas de Gerenciadores de Banco de Dados Distribuídos (SGBDDs) são naturalmente mais complexos que SGBDs. Além de herdarem os problemas provenientes dos SGBDs, novos problemas surgem com a inclusão da distribuição de dados, processamento distribuído, uso de redes de computadores, etc. A tecnologia de SGBDDs apresenta vantagens e desvantagens.

Entre as vantagens, destacam-se:

- *Aumento de desempenho:*

Com o processamento distribuído é possível acessar dados e executar tarefas concomitantemente, utilizando os diversos nós da rede. (TAVARES, *et al.*, 1996) é um exemplo de ganho de desempenho conseguido através de processamento em paralelo. (DEWITT, *et al.*, 1988) mostra como o acesso paralelo à base consegue ganhos em desempenho. Com o projeto distribuído de banco de dados é possível tirar proveito da distribuição de dados para reduzir o tempo de processamento das consultas.

- *Sistema com maior confiabilidade:*

A existência de dados replicados através da rede é uma estratégia utilizada para sistemas onde a rede de computadores é instável. Nesses sistemas, pode ocorrer pane ou defeito em uma parte da topologia. Portanto, o SGBDD deve conter mecanismos de controle para atualização e gerência dos dados replicados, como objetivos de evitar inconsistência.

- *Expansibilidade*:

Em ambientes distribuídos é mais fácil aumentar o tamanho e a capacidade do sistema. A expansão pode ser conseguida com a inclusão de mais uma máquina à rede de computadores, aumentando também a capacidade de processamento e armazenamento do sistema.

A existência de sistemas distribuídos facilita a captação de dados situados em outras localidades geográficas. A captação da informação é feita na localidade podendo ser compartilhada pelo resto do sistema. Isto torna o acesso mais rápido e eficiente.

As **desvantagens** apresentadas são:

- *Custo*:

Uma das características observadas em sistemas distribuídos é a utilização de hardware e mecanismos de controle de comunicação e manutenção. Estes requisitos aumentam o custo do sistema, exigindo hardware e software adicionais. A mão-de-obra também deve ser levada em conta, uma vez que, quanto mais complexo, mais pessoas capacitadas são utilizadas na implementação e manutenção do sistema.

- *Controle da distribuição*:

A distribuição de dados caracteriza um problema no que diz respeito à coordenação e ao controle. Atualizações, mudanças de configuração, consultas e outras operações devem ser coordenadas pelo sistema. Políticas de controle de transações correspondem a uma área de estudo em sistemas distribuídos.

- *Segurança*:

Em sistemas centralizados a base de dados fica armazenada em uma única localização. Em contrapartida, informações são replicadas e divididas pela rede em sistemas distribuídos. Desta forma mecanismos de segurança na rede

devem ser implementados, aumentando o custo e a complexidade do sistema. Em (NYANCHAMA, OSBORN, 1994) são apresentados aspectos considerados na segurança de SGBDDs Orientados a Objetos.

- *Conversão para nova tecnologia*

A transferência para uma nova tecnologia não é um fator simples. Os investimentos feitos em SGBDs não distribuídos não podem ser ignorados. Existem áreas de pesquisa que procuram buscar uma solução de integração de base de dados ou de interoperabilidade entre sistemas diferentes.

Na próxima seção serão apresentadas as áreas de pesquisa atuais em SGBDDs que buscam soluções para os problemas citados anteriormente.

II.2.2 Áreas de Atuação

Com o objetivo de assegurar toda a potencialidade de um SGBDD, os pesquisadores procuram resolver problemas relacionados com a distribuição dos dados, gerência de transações, replicação de dados, integração de bases de dados heterogêneas, etc. A seguir são mostradas as principais áreas de pesquisas relacionadas a SGBDDs:

II.2.2.1 Projeto de distribuição de bases de dados

Um projeto de distribuição define como será feito o armazenamento dos dados entre os nós da rede do sistema distribuído. Segundo (ÖSZU, VALDURIEZ, 1991a), uma base de dados pode ser dividida ou replicada pelos nós da rede. A replicação de dados almeja dar maior confiabilidade ao sistema no caso de falha e queda de algum nó da topologia. Um sistema pode ser parcial ou inteiramente replicado. A fragmentação dos dados visa a economia de recursos, evitando acessos desnecessários na consulta à base de dados. Outra característica da fragmentação é a possibilidade de utilização de processamento paralelo na execução de consultas, aumentando o desempenho do sistema. A seção II.5 aborda essa área de pesquisa mais detalhadamente.

II.2.2.2 Processamento e otimização de consultas

Um banco de dados, ao receber uma consulta, constrói um plano de execução. Tal plano consiste de uma seqüência de operações que devem ser executadas para acessar os dados e executar a consulta. Um otimizador de consultas gera planos

alternativos e os avalia, escolhendo o de menor custo. Nessa avaliação entram aspectos como custo de comunicação, localização dos dados, disponibilidade dos dados, etc.

II.2.2.3 Gerência de transações

A gerência de transações tem como objetivo manter a integridade da base de dados, assim como sincronizar as operações executadas no sistema. O acesso concorrente a bases de dados pode gerar inconsistências se não for tratado. A distribuição é um aspecto que vai influenciar o controle de concorrência. Basicamente, existem duas soluções de controle de concorrência (BERSTEIN, GOODMAN, 1981) utilizadas. A primeira se caracteriza por um tratamento pessimista, sincronizando as transações antes de suas execuções. A segunda executa as operações e, na hora da atualização, verifica se a integridade da base de dados será mantida. Estes mecanismos utilizam duas primitivas conhecidas como *locking* e *timestamp*. Na primeira, o acesso ao dado é exclusivo a uma operação, não sendo permitido outro acesso. Na segunda, transações de dados são executadas em alguma ordem garantindo a sincronização. Em (HSIAO, ÖZSU, 1981) é apresentado um conjunto de algoritmos de controle de concorrência propostos.

Outro ponto investigado é a parada do sistema por espera de recursos ou *deadlock*. Basicamente, existem três maneiras de se detectar *deadlock*. Na primeira, um nó do sistema fica responsável por detectar *deadlock*. Na segunda, a parada é detectada através de uma hierarquia de processos existentes em cada nó. Na terceira, cada nó fica responsável por detectar uma parada por espera de recurso. Existem trabalhos (KNAPP, 1987) (BARBOSA, 1990) (ELMAGARMID *et al.*, 1988) que apontam maneiras para detectar e tratar possíveis paradas por *deadlock* em SGBDDs.

II.2.2.4 Mecanismos contra falhas na rede

Uma parte da base de dados pode ficar inacessível durante um determinado tempo, devido a uma queda de comunicação na rede. O SGBDD deve ter a capacidade de tratar tais situações (GRAY, 1979). Dados replicados podem ser utilizados para manter a operacionalidade do sistema. Desta forma, quando os nós voltarem à rede, as bases serão atualizadas garantindo a integridade das informações. No caso de falhas planejadas, dados podem migrar para outras áreas de armazenamento.

II.2.2.5 Integração de bases heterogêneas e Interoperabilidade

A existência de várias bases de dados com diferentes modelos de dados e diversos modos de acesso torna-se um desafio para a integração e para o acesso a essas informações. A integração de bases de dados pode ser feita através de sistemas que formam um modelo global (SOUZA, 1986) (BATINI *et al.*, 1986), resolvendo todos os conflitos encontrados na integração de duas ou mais bases. Alguns trabalhos falam com mais detalhe sobre tais sistemas, denominados Sistemas de Múltiplas Bases de Dados (KIM *et al.*, 1995) (PITOURA *et al.*, 1995).

Outra forma de integração é a comunicação de diferentes sistemas de bancos de dados que gerenciam bases de dados heterogêneas. Segundo (UCHÔA, SECIM, 1995), o uso de padrões para a orientação a objetos surge como uma estratégia para coordenar com flexibilidade e integrar os recursos de processamento de informações interligados por redes de comunicação.

A interoperabilidade de banco de dados é tratada no projeto HIMPAR através de uma arquitetura proposta em (PIRES, MATTOSO, 1996), que utiliza o padrão CORBA e está de acordo com a norma ODMG-93 (SOLEY, KENT, 1995) (CATTEL, 1994).

II.3 PROJETO DE DISTRIBUIÇÃO DE BASES DE DADOS

O projeto de distribuição é a tomada de decisões relativas à localização dos dados e das aplicações pelos nós que compõem o sistema. Esta seção trata do primeiro aspecto, isto é, a distribuição dos dados através da rede do sistema. Segundo Karlapalem, Navathe e Morsi (KARLAPALEM *et al.*, 1994), o projeto de distribuição pode aumentar o desempenho do sistema de duas maneiras: reduzindo a quantidade de dados irrelevantes acessados pelas consultas e diminuindo a transferência de dados no processamento distribuído. Estes objetivos são alcançados pela fragmentação dos dados através da rede, e da alocação destes fragmentos entre os nós do sistema. As seções II.3.2 e II.3.3 abordam esses dois aspectos, respectivamente. A seção II.3.4 trata dos aspectos considerados na distribuição de bases de dados orientadas a objetos.

II.3.1 Tipos de Estratégias de Projeto Distribuído

Existem duas estratégias de projeto de distribuição (CERI *et al.*, 1987) denominadas estratégia ascendente (*bottom-up*) e descendente (*top-down*). As duas próximas subseções se referem a estas metodologias.

II.3.1.2 Metodologia Ascendente

Essa metodologia é utilizada na integração de bases de dados heterogêneas armazenadas em vários nós da rede. A estratégia ascendente parte dos esquemas conceituais locais, formando um esquema global, de tal forma que a manipulação das bases de dados pelo sistema seja feita através desse último. Os sistemas de múltiplas bases de dados, mencionados na seção II.2.2.5, utilizam a metodologia ascendente.

II.3.1.3 Metodologia Descendente

Essa estratégia consiste na construção de modelos locais a partir de um modelo geral único, através da **fragmentação** e **alocação** da base de dados . Cada nó vai conter partes dos dados representados pelo modelo global. A metodologia top-down é utilizada em sistemas cujos componentes são fortemente acoplados e em projetos de distribuição de bases de dados iniciais. Öszu e Valduriez (ÖSZU, VALDURIEZ, 1991a) explicam com mais detalhes o processo top-down de distribuição de bases de dados.

II.3.2 FRAGMENTAÇÃO: VANTAGENS E DESVANTAGENS

A fragmentação da base de dados permite que dados irrelevantes para certas aplicações não sejam acessados, economizando recursos de entrada e saída, e, consequentemente, ganhando desempenho. Em (LIMA, MATTOSO, 1995) é mostrado como um sistema pode tirar proveito da distribuição dos dados para acessar um volume menor de informação obtendo ganho de desempenho. Outra vantagem é a execução de transações concorrentemente, além do processamento paralelo de consultas. Tavares e Soares (TAVARES, MATTOSO, 1996) mostram ganhos de desempenho conseguidos através do acesso a porções menores da base e do processamento distribuído das consultas. (MEYER, 1997) é um exemplo de implementação com acesso paralelo à base de dados.

Porém a fragmentação exige do sistema mecanismos de controle capazes de manter a consistência da base de dados. Outra desvantagem é a perda de desempenho em operações que acessam vários fragmentos. A próxima subseção apresenta as alternativas de fragmentação existentes e suas características.

II.3.2.2 Fragmentação Horizontal

A fragmentação horizontal consiste na divisão de uma relação ao longo de seus registros, ou seja, em um banco de dados relacionais, uma tabela é dividida em várias partes, cada uma delas contendo um determinado número de tuplas. Já em banco de dados orientados a objetos, a fragmentação horizontal ocorre na partição das instâncias da classe. Uma fragmentação feita em uma extensão de classe resultará em conjuntos de objetos que serão armazenados através dos nós da rede. Segundo (KARLAPALEM *et al.*, 1994), tanto a fragmentação quanto a alocação de objetos são aplicadas aos dados. Porém em SGBDOO, a fragmentação e a alocação ocorrem nos dados representados pelo objeto, assim como nas suas operações representadas pelos seus métodos (comportamento dos dados).

No. Emp.	Nome	Função	Salário	No. Dept.
1	José dos Santos	Analista de Sistemas	2500,00	2
2	Sandra Rodrigues	Contador	3000,00	1
3	Marcos Dantas	Administrador de Empresas	5500.00	1
4	Célio Lima	Engenheiro	6000,00	2

No. Dept.	Nome Dept.
1	Finanças
2	Informática

Figura II.1 - Relação Empregados e Departamentos

A fragmentação pode ser feita segundo critérios físicos ou lógicos. Em sistemas de bancos de dados distribuídos a fragmentação segue o critério lógico. Em sistemas parelos o critério físico também é adotado. No critério físico a capacidade de armazenamento de cada nó é avaliada, determinando o tamanho de cada fragmento e sua localização na rede. Já no no critério lógico a fragmentação é realizada por técnicas ou algoritmos que consideram algumas informações como os predicados das consultas e suas freqüências no sistema. Técnicas de faixa de valores possibilitam a divisão da base em fragmentos, segundo a variação do valor de um atributo. Técnicas de *hashing* são utilizadas a partir da aplicação de uma função no tipo de operador da consulta, localizando em que nós a informação requisitada está armazenada.

Basicamente, existem dois tipos de fragmentação horizontal: a primária e a derivada. Na primeira a fragmentação é realizada diretamente sobre um atributo da

relação-alvo. A figura II.1 mostra duas tabelas de uma base de dados de uma empresa. A partir dessas relações:

(**Ex.**: *Select * from Empregados where [Empregados].salario > 5000*)

Uma fragmentação cabível no exemplo acima seria a divisão da relação Empregados em duas fatias. A primeira teria registros dos empregados que ganham abaixo ou igual a 5000 e a segunda determinaria os empregados que recebem acima de 5000.

A fragmentação horizontal derivada é feita de acordo com predicados definidos em outras relações.

Ex.: A relação Empregados se relaciona com a relação Departamentos através do atributo "No. do Dept.". Uma fragmentação possível é a divisão dos empregados segundo os seus departamentos. Os predicados utilizados na fragmentação foram: Departamentos = "Finanças" (Figura II.1) e Departamentos = "Informática". A relação Empregados é fragmentada de acordo com este predicado: um fragmento conterá empregados que trabalham no dept. de informática e no outro estarão os empregados do dept. de finanças.

II.3.2.3 Fragmentação Vertical

A fragmentação vertical é feita no nível dos atributos da relação. Uma tabela será fragmentada de maneira que cada fragmento contenha um conjunto de atributos da relação. Isto é, a tabela não será dividida em relação aos registros como na fragmentação horizontal, mas sim, na sua estrutura.

Um exemplo de divisão vertical aplicado ao da figura II.1 seria a separação do atributo salário da relação Empregados. Sendo assim, a tabela de Empregados seria dividida em duas outras relacionadas pelo atributo "No. do Emp.". A figura II.2 mostra o exemplo citado acima.

No. Emp.	Salário
1	2500
2	3000
3	5500
4	6000

No. Emp.	Nome	Função	No. Dept.
1	José dos Santos	Analista de Sistemas	2
2	Sandra Rodrigues	Contador	1
3	Marcos Dantas	Administrador de Empresas	1
4	Célio Lima	Engenheiro	2

Figura II.2 - Exemplo de fragmentação vertical

A fragmentação vertical pode ser feita basicamente de duas maneiras. A primeira considera cada atributo da relação como sendo um fragmento num estágio inicial. Depois os fragmentos resultantes são unidos segundo um critério. Alguns trabalhos abordam esta heurística que é utilizada tanto em bancos de dados centralizados (HAMMER, NIAMIR, 1979) quanto em bancos de dados distribuídos (SACCA, WIEDERHOLD, 1985). A outra forma de decidir que atributos serão fragmentados baseia-se no comportamento do sistema, isto é, quais são os atributos acessados pelas diversas consultas do sistema. Alguns algoritmos propostos utilizam esta metodologia (NAVATHE *et al.*, 1984).

Geralmente os algoritmos de fragmentação vertical baseiam-se em uma matriz de afinidades, criada através das informações de acesso à base de dados. Esta matriz vai determinar os atributos que devem ser agrupados e os que devem estar em fragmentos diferentes. O algoritmo para determinar o agrupamento e a fragmentação de atributos é explicado com mais detalhe em (HOFFER, SEVERANCE, 1975) (ÖSZU, VALDURIEZ, 1991a).

II.3.4.5 Fragmentação Híbrida

Uma outra alternativa de fragmentação é chamada híbrida, onde são aplicados os dois tipos de fragmentação. É possível dividir horizontalmente um fragmento que foi constituído de uma divisão vertical da base. Um exemplo seria a divisão do fragmento vertical do exemplo da figura II.2 em dois fragmentos separados, segundo o predicado "Salário > 5000" utilizado no exemplo da figura II.1. A figura II.3 mostra como ficou o fragmento vertical segmentado em dois outros.

No. Emp.	Salário
1	2500
2	3000

No. Emp.	Salário
3	5500
4	6000

Figura II.3 - Exemplo de fragmentação Híbrida

II.3.4 ALOCAÇÃO

Após a determinação dos fragmentos, a próxima etapa é a associação e a gravação destes através da rede. Além da etapa de alocação dos fragmentos existe outro aspecto que deve ser considerado. A alteração da base de dados por uma consulta pode

afetar localização atual dos dados. Isto significa que uma determinada informação não pode ser mais armazenada no mesmo local devido à modificação. Desta forma, o dado modificado deve ser realocado em outro fragmento do sistema.

A alocação dos fragmentos da base de dados procura distribuí-los na rede de forma a melhorar o tempo de resposta, além de tentar aumentar o número de consultas executadas simultaneamente (*throughput ou vazão do sistema*).

Segundo (ÖSZU, VALDURIEZ, 1991a), o problema da alocação está em diminuir os custos de armazenamento, de consulta, e de atualização dos fragmentos nos diversos nós da rede. Porém, é muito difícil conseguir uma solução única para todos os problemas. A atualização de dados replicados no sistema também constitui um problema em aberto na área de alocação, tanto em bancos de dados relacionais como em SGBDOOs. Tem sido observado que o custo de comunicação é muito alto em sistemas distribuídos (HUANG, WOLFSON, 1994). Alguns trabalhos mostram que o esforço de economizar tráfego de comunicação da rede é recompensado com aumento de desempenho no sistema (MEYER, 1997). Logo, a transferência de dados de um nó para outro da rede, assim como a atualização de dados replicados, caracterizam-se como problema relacionado com o custo de comunicação. Quando estes aspectos são considerados num ambiente dinâmico, mais problemas surgem. O capítulo III aborda o problema da alocação dinâmica de bases de dados orientados a objetos.

II.3.3 Fragmentação e Alocação em SGBDOO

Como já foi dito anteriormente, o projeto distribuído da base de dados permite aumento de desempenho através de processamento distribuído ou da redução de informação a ser consultada pelo sistema. Em bases de dados orientadas a objetos esta redução é conseguida aplicando-se fragmentação vertical e horizontal como no relacional. Porém, o modelo de dados orientado a objetos adiciona novos aspectos a serem considerados no problema de fragmentação, tais como: a hierarquia de classes, o agrupamento de objetos complexos, o compartilhamento de objetos através de identificadores, as chamadas a métodos e a alocação destes.

Esses problemas são tratados em diversos trabalhos na literatura. Em (MAIER ET AL, 1992) é abordada a questão do agrupamento de objetos complexos em ambientes distribuídos. O acesso de um objeto complexo formado por várias referências

a outros objetos pode acarretar perda de desempenho devido ao alto custo de comunicação no acesso a seus atributos, assim como na chamada de seus métodos.

Os aspectos relacionados com o projeto distribuído em base de dados orientados a objetos são discutidos em (KARLAPALEM *ET AL.*, 1994). (BAIAO, 1997) faz uma comparação entre as diversas metodologias de projeto distribuído de bases de dados OO. BAIAO e MATTOSO (1998) propõe um algoritmo que, através de informações fornecidas pelo usuário gera um esquema de distribuição para bases de dados. Este algoritmo utiliza tanto fragmentação vertical como horizontal.

II.4 SISTEMA DE BANCO DE DADOS PARALELOS

O aumento de desempenho em Sistemas de Gerenciadores de Banco de Dados está relacionado com a otimização do acesso à base. A diminuição do uso de recursos de entrada e saída através do projeto distribuído de banco de dados aliado a utilização de equipamentos mais modernos melhoram o desempenho do sistema. Além disso, paralelismo representa para a tecnologia de banco de dados uma possibilidade de melhora de desempenho através do acesso e do processamento paralelo. Essa seção introduz algumas das principais características de Sistemas Gerenciadores de Banco de Dados Paralelos, como: acoplamento ao hardware, níveis de paralelismo, modelos de arquitetura, fragmentação dos dados e execução de consultas.

II.4.1 ACOPLAMENTO AO HARDWARE

Um dos aspectos que ajuda a caracterizar um Sistema de Banco de Dados Paralelo é o acoplamento ao hardware. Segundo (MATTOSO, 1993), "o acoplamento ao hardware, indica a categoria de concepção do SBDP quanto ao grau de acoplamento ao seu ambiente hospedeiro". Sistemas que são totalmente acoplados são conhecidos como máquinas de banco de dados. Esses sistemas caracterizam-se por ter um hardware projetado especificamente para suportar suas operações.

Existem SBDPs que são desenvolvidos para um computador paralelo de uso geral. Nesses sistemas o acoplamento ao hardware é inferior ao do caso anterior já que a máquina utilizada pelo sistema não foi desenvolvida especificamente para o SBDP.

Há sistemas que caracterizam-se pelo não acoplamento ao hardware. Estes sistemas, classificados em (MATTOSO , 1993) como servidores de banco de dados

paralelos, não possuem um hardware específico, mas tiram proveito de um sistema operacional que facilita as operações com dados.

(MATTOSO, 1993) relaciona uma série de sistemas representantes da categoria conhecida como máquinas de banco de dados. Entre eles estão: NonStopSQL e GAMMA (DEWITT *et al.*, 1988). Um exemplo da segunda categoria apresentada, com pouco grau de acoplamento, é o sistema XPRS (STONEBRAKER *et al.*, 1988). Exemplos de servidores de banco de dados são EDS e SHORE (CAREY *et al.*, 1994).

II.4.2 NÍVEIS DE GRANULARIDADE DO PARALELISMO

O nível de granularidade do paralelismo está relacionado à quantidade de operações que será realizada simultaneamente no processamento de consultas do Sistema de Banco de Dados Paralelos. Existem três níveis de granularidade de paralelismo: entre consultas, dentro de consultas e entre operações. Esses três níveis podem coexistir. Entretanto, na maioria dos casos, a implementação de um sistema privilegia um determinado nível. A seguir será apresentada a definição dos três níveis mencionados acima:

- *Paralelismo entre consultas*

Neste nível de consulta acontece a execução de mais de uma consulta concomitantemente, isto é, duas ou mais consultas podem ser executadas em paralelo pelo sistema. Sistemas com essa característica tem como objetivo atender ao maior número de usuários ao mesmo tempo através da execução simultânea das consultas de diversos usuários. O número de consultas executadas por segundo (vazão) constitui uma medida de desempenho de sistemas de banco de dados paralelos.

- *Paralelismo dentro de consultas*

Esse nível de paralelismo tem como objetivo otimizar consultas complexas e ad-hoc. O paralelismo dentro de consultas está ligado à execução de uma mesma consulta em paralelo. A execução de cada operador da consulta em paralelo possibilita um ganho de desempenho. A otimização pode ser feita através da geração de vários planos de execução, utilizando para isto, estratégias de paralelismo. Após a geração, o plano de menor custo é escolhido e executado.

- *Paralelismo dentro de operações*

Uma consulta é mapeada para um conjunto de operações que a resolvem. Em SGBD relacionais tais operações compõem a álgebra relacional. Este nível de granularidade tem como objetivo paralelizar os algoritmos responsáveis pelas suas execuções. Geralmente, o paralelismo é obtido através da replicação da operação que representa a consulta pelos diversos nós do sistema. Em um sistema de memória distribuída o predicado de uma consutla de seleção seria enviada aos nós, que executariam a consulta em paralelo. Ao final, o resultado da consulta seria a união de todos os resultados obtidos em cada nó. Em arquiteturas de memória compartilhada e disco distribuído os fragmentos seriam distribuídos para os nós dos sistemas que avaliariam cada predicado em paralelo.

II.4.3 MODELOS DE ARQUITETURA

Os modelos apresentados estão relacionados com o grau de compartilhamento das memórias principais e secundárias. Essas arquiteturas, que também são denominadas de modelos de memória são classificadas em três: modelo de memória compartilhada, modelo de disco compartilhado e modelo de memória distribuída. As subseções seguintes abordam cada tipo de arquitetura.

II.4.3.1 Modelo de memória compartilhada

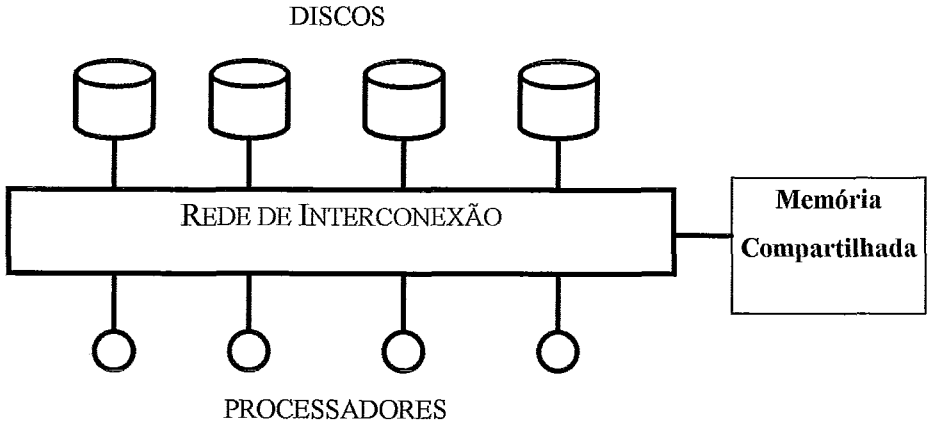


Figura II.4 - Memória Compartilhada

Neste modelo, mostrado na figura II.4, cada nó do sistema é composto por um único processador. A memória principal e os discos são compartilhados pelos processadores através de um canal de comunicação de alta velocidade.

Essa arquitetura facilita o balanço de carga entre os processadores, podendo até ser feito um controle dinâmico através da observação do sistema. O paralelismo entre consultas é conseguido de maneira muito simples. A divisão dos dados é bastante dinâmica, pois estes não estão distribuídos fisicamente, permitindo um bom balanceamento de carga.

Porém qualquer falha no esquema de interconexão resulta em pane geral do sistema. Com a utilização de processadores cada vez mais velozes os conflitos de acesso ao módulo de memória são cada vez mais freqüentes, degradando a performance do sistema. Outro aspecto negativo está no alto custo da implantação da rede de interconexão dos processadores com os módulos de memórias e discos. Segundo (VALDURIEZ, 1993) esse tipo de arquitetura também tem uma limitação quanto à expansão da capacidade de processamento.

II.4.3.2 Modelo de disco compartilhado

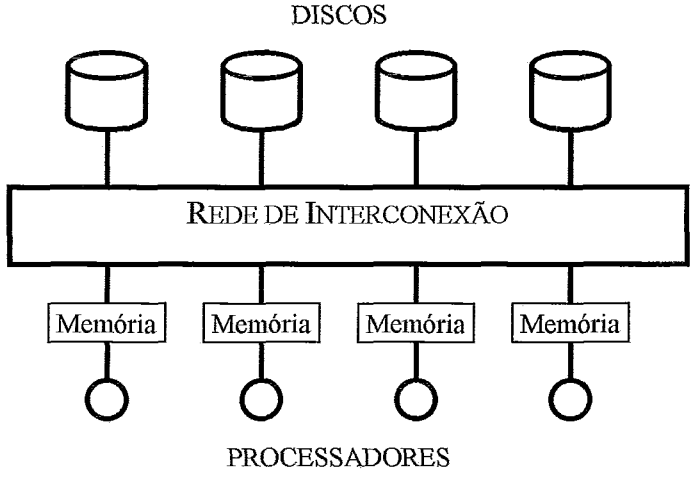


Figura II.5 - Disco Compartilhado

Essa arquitetura é caracterizada pelo compartilhamento da memória secundária. O acesso compartilhado à base de dados pode ser compensado através da manutenção de buffers de disco na memória de cada processador. O custo é inferior ao do modelo anterior. O sistema pode ser composto de estações de trabalho que acessam um ou mais discos simultaneamente através da rede. Desta forma, além do custo ser infeiror ao do modelo anterior, a capacidade de expansão do sistema também é beneficiada. Um bom balanço de carga é garantido.

O acesso compartilhado de disco é um problema de performance em potencial. O sistema deve dispor de mecanismo de controle de transações (mencionados na seção

II.2.2.3), aumentando a complexidade do sistema. Um exemplo de protótipo que usa este modelo de memória é o Pargoa-V (TAVARES *et al.*, 1996), que através da carga equilibrada dos dados pelos seus processadores consegue aumentos significativos de desempenho.

II.4.3.2.3 Modelo de memória distribuída

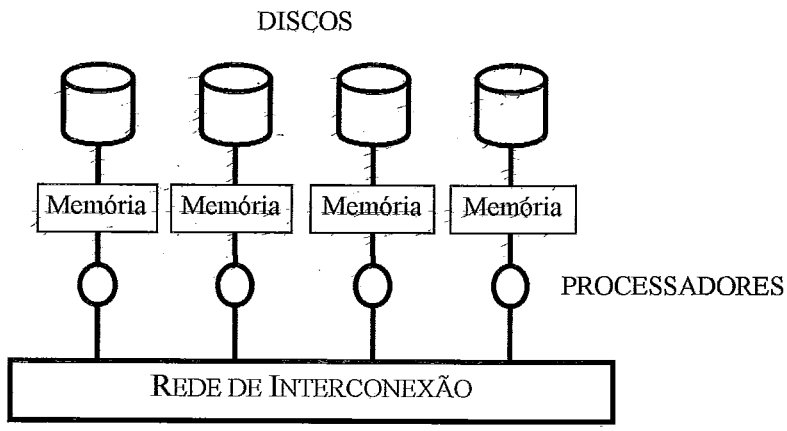


Figura II.6 - Memória Distribuída

No modelo de memória distribuída, cada nó do sistema tem acesso exclusivo as suas unidades de disco e memória principal. Os dados são distribuídos fisicamente, constituindo um problema para se conseguir um bom balanço de carga. Técnicas usadas no projeto distribuído de base de dados podem ser utilizadas para tentar uma distribuição equilibrada da base. Dados podem ser replicados a fim de garantir a disponibilidade da informação em caso de falha. O custo também é reduzido, assim como no modelo de disco compartilhado. O acesso concorrente de informação é possível através da execução em paralelo de operações em cada nó. A sua capacidade de expansão é superior aos modelos anteriores.

II.4.4 DISTRIBUIÇÃO DOS DADOS

A eficiência conseguida através do processamento paralelo dos dados está relacionada com o balanço de carga do sistema. Na arquitetura de disco compartilhado busca-se um balanço de carga de processamento, uma vez que o acesso à base de dados é compartilhado. Desta forma, é possível dar uma carga maior de processamento para aqueles processadores mais rápidos e que possuem mais memória principal. Esses processadores levarão menos tempo para processarem as consultas diminuindo o tempo de resposta. Este assunto é comentado mais detalhadamente no capítulo III.

II.4.5 PROCESSAMENTO DE CONSULTAS

O processamento paralelo de consultas baseia-se na tradução de uma consulta em planos de execução eficientes e na execução em paralelo do mais eficiente. Os planos devem representar corretamente a consulta traduzida, formando o resultado esperado pelo usuário. A tarefa de tradução é feita através de leis de transformação para operações da álgebra relacional.

Após a tarefa de tradução da consulta em planos de execução é feita a análise destes, escolhendo o de menor custo. A estimativa dos custos de entrada e saída, de comunicação e tempo de processamento podem avaliar tanto o tempo de resposta, como o tempo total de um conjunto de consultas. A otimização do tempo de resposta do sistema é obtida através da escolha de planos de execução que utilizam toda a capacidade de paralelismo oferecida pelo sistema. No segundo caso, os planos são escolhidos de modo que consultas diferentes possam ser executadas simultaneamente, conseguindo um ganho no tempo total.

A distribuição de dados é um importante fator na execução de consultas. Dependendo do tipo de fragmentação que foi aplicada, é possível conseguir os níveis de paralelismo mencionados na seção II.4.3. Porém existem casos onde o custo de rearrumar a base de dados aumentando o paralelismo é menor do que o ganho conseguido.

A distribuição por faixa de valores, onde uma relação é fragmentada na rede segundo valores de um dos seus atributos, possibilita um melhor balanço de carga dos dados pelo sistema. As técnicas de *hashing* possibilitam que operações de *junção (join)* (DEWITT, GERBER, 1985), união e diferenças possam ser executadas concomitantemente. Duas relações podem ser divididas pelos nós da rede, tendo como base o atributo utilizado nestes tipos de operação. Esta divisão é efetuada através da aplicação de uma função de hashing sobre o atributo de junção, descobrindo a localização apropriada. Uma operação de *join* poderia ser executada como a união de várias operações equivalentes, isto é, *joins* entre os fragmentos de cada nó, executados em paralelo. Outra forma execução de operações de junção em paralelo é fragmentar apenas uma das relações e replicar a o outra relação entre os nós participantes. Geralmente a maior relação é fragmentada permanecendo em cada nó (para futuras consultas), enquanto a menor relação é transmitida. Alguns trabalhos apresentam

algoritmos para paralelizar operações de *join,* como (VALDURIEZ, 1986), (WOLF *et al.*, 1990), (SCHNEIDER, DEWITT, 1990) (GRAEFE, 1993).

Em sistemas como Bubba (BORAL *et al.*, 1990), o plano de execução escolhido pelo otimizador é transformado em um programa que será executado ao mesmo tempo. Esse tipo de paralelismo está relacionado com a utilização de linguagens de banco de dados paralelos. No sistema XPRS (STONEBRAKER *et al.*, 1988) o plano de execução escolhido pelo otimizador de consultas é executado simultaneamente de acordo com o balanço de carga de processamento atual. Dependendo do grau de utilização do sistema, a consulta será realizada por mais ou menos processadores.

(VALDURIEZ, 1993) ressalta as diferenças entre consultas simples e complexas. Custos elevados gastos em consultas ad-hoc que são executadas apenas uma vez devem ser evitados. Portanto, o sistema deve ter estratégias diferentes que irão variar de acordo com o tipo de consultas (simples ou complexas) e com os requisitos da aplicação (consultas repetitivas ou executadas somente uma vez).

Capítulo III

DISTRIBUIÇÃO DE CARGA

III.1 INTRODUÇÃO

No capítulo II foi apresentado um panorama geral de sistemas de banco de dados distribuídos e sistemas paralelos de banco de dados. O objetivo do capítulo III é mostrar como a distribuição de carga de processamento e dados modifica o desempenho destes sistemas. Porém, para haver uma distribuição de carga de recursos é necessário que o sistema esteja preparado para alocar estes recursos. No caso específico dos dados, um sistema pode ter uma distribuição física nos diversos nós, ou pode haver alocação feita no momento da execução de alguma transação.

Já foi observado no capítulo II que em arquiteturas de memória e disco compartilhados esta alocação dinâmica de dados pode ser feita sem maiores problemas quando se trata de execução de consultas de seleção. Quanto a consultas de atualização existem problemas como a atualização do cache do sistema quando um objeto é modificado. Isto se deve ao fato de que todos os nós têm acesso às unidades de memória secundária do sistema. No caso da arquitetura de memória distribuída, outros aspectos devem ser analisados. Um deles é a constatação de que os dados estão armazenados em unidades locais a cada nó e, por isso, o acesso só é feito através da comunicação de um nó com outro. Esse problema afeta bastante uma tentativa de alocar dados dinamicamente em sistemas de arquitetura de memória distribuída.

O objetivo principal deste capítulo é ressaltar a importância da utilização de estratégias de distribuição de carga em um sistema de banco de dados paralelo. Estas estratégias devem ser efetuadas de maneira dinâmica permitindo que o sistema se adapte a situações inesperadas.

A seção III.2 apresenta o assunto da distribuição de carga. A seção III.3 descreve algumas estratégias de distribuição de dados em sistemas de banco de dados paralelos. A seção III.4 compara as estratégias apresentadas na seção anterior.

III.2 DISTRIBUIÇÃO DE CARGA

A distribuição de carga de tarefas entre os diversos processadores de um sistema possibilita o ganho no tempo de execução, diminuindo desta forma o tempo de resposta. A distribuição dos dados entre os nós do sistema permite o acesso simultâneo da informação, assim como o processamento distribuído.

Uma tarefa pode ser dividida em um conjunto menor de tarefas que serão executadas ao mesmo tempo pelos processadores de cada sistema. Inicialmente, tarefas com a mesma complexidade são divididas entre os nós do sistema. Este tipo de distribuição é denominada *uniforme*.

Com o aumento da tecnologia de hardware, foi possível desenvolver sistemas compostos de processadores com capacidade de processamento diferente. Desta forma, surgiu a necessidade de fazer uma divisão diferenciada, baseada na capacidade de processamento de cada nó.

Em sistemas paralelos de banco de dados, esta distribuição não uniforme é utilizada nas três arquiteturas. Outro fator que a distribuição não uniforme atinge é o tempo de inicialização de uma tarefa em cada processador. Em sistemas com arquitetura de memória distribuída e disco compartilhado, a inicialização das tarefas a serem executadas em cada nó é feita serialmente por um nó mestre. Desta forma, supondo-se que cada nó levará a mesma quantidade de tempo para executar uma tarefa, a figura III.1 representa o tempo que o sistema gastou para a execução da tarefa total, através de uma distribuição de carga uniforme.

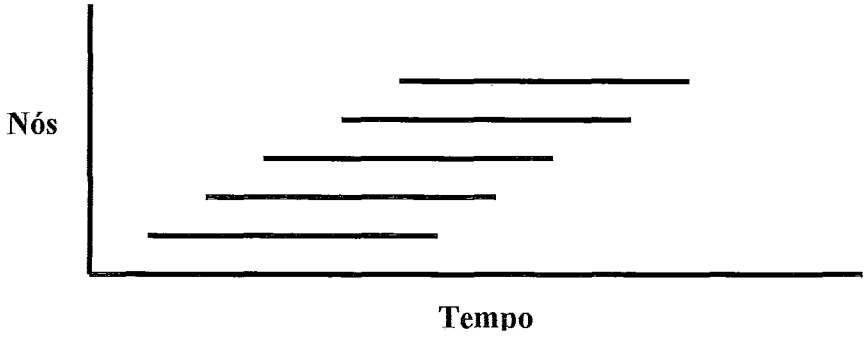


Figura III.1 - Distribuição de carga uniforme

É importante registrar que no exemplo acima foi considerado constante o tempo de inicialização de uma tarefa em um nó. Uma distribuição não uniforme poderia fazer uma distribuição de carga de forma que as tarefas inicializadas mais tarde fossem

menores, acarretando no término de todas as tarefas em tempos muito próximos. A figura III.2 demonstra essa forma de divisão:

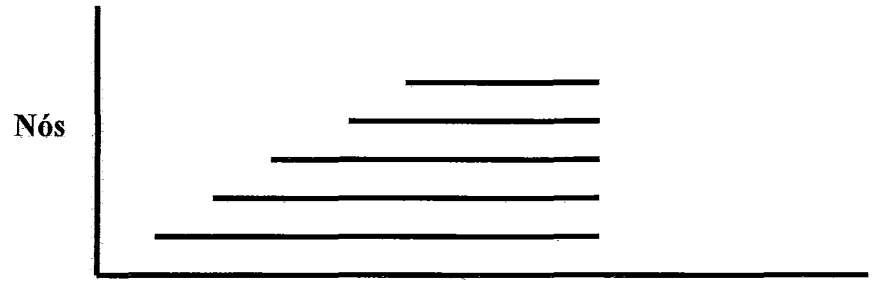


Figura III.2 - Distribuição de carga não uniforme

Quanto à distribuição de dados, aplicam-se os conceitos descritos anteriormente. A distribuição dos dados pode ser uniforme, isto é, porções iguais da base são distribuídas pelos nós do sistema. Numa distribuição não uniforme, quantidades diferentes da base são associadas a cada nó. Um outro aspecto que deve ser mencionado é o relacionamento existente entre as várias relações que compõem a base. Consultas feitas somente em uma relação podem ser divididas em tarefas independentes, onde o resultado final será a união dos resultados das tarefas executadas em cada nó. Porém, em consultas complexas como as de junção (*join*), onde duas relações estão envolvidas, a divisão da consulta em outras menores e independentes é um problema mais complexo.

É importante ressaltar que a distribuição de carga de dados é um problema mais complexo em ambientes em que a base está distribuída fisicamente, isto é, os dados encontram-se armazenados em unidades onde só um determinado nó tem acesso. Neste tipo de ambiente (memória distribuída), a distribuição da base geralmente é feita previamente às execuções das consultas. Em sistemas onde a base está armazenada em um ou mais discos acessados por todos os nós através da rede (memória compartilhada, disco compartilhado), a distribuição de carga dos dados pelos processadores na execução de uma operação é feita em tempo de execução.

(BASSILIADES, VLAHAVAS, 1995) apresenta o tempo de resposta de um sistema de banco de dados OO paralelo que utilizou a fragmentação uniforme da base de dados. Este sistema tem como características a arquitetura de memória distribuída e a execução de consultas simples à base de dados. O tempo de resposta deste sistema é apresentado pela equação abaixo:

$$^{u}R = \alpha N + \frac{cQ}{N} \tag{3.1}$$

Tempo de resposta para uma fragmentação uniforme

Onde α é o tempo gasto para iniciar as consultas que serão executadas em cada nó do sistema. N é o número de nós existentes no sistema e c é o tempo gasto para processar cada unidade de informação da base de dados. Esta unidade pode ser uma tupla de valores ou um objeto. Q representa a quantidade de objetos que compõem a coleção. É importante ressaltar que nestes cálculos, o tempo de processamento de cada máquina é igual, isto é, não há diferenças entre velocidades das máquinas.

Com esta definição é possível calcular o número ótimo de nós para que o sistema tenha um tempo de resposta mínimo:

$$^{u}N_0 = \sqrt{\frac{cQ}{\alpha}} \tag{3.2}$$

Número ótimo de nós para conseguir um TR mínimo

O tempo de resposta ótimo do sistema é obtido aplicando-se a equação 3.2 na equação 3.1:

$$^{u}R_0 = 2\sqrt{\alpha c Q} \tag{3.3}$$

Tempo de resposta ótimo (mínimo) com fragmentação uniforme

A fragmentação não uniforme dos dados é feita com o objetivo de garantir que todos os nós acabem a execução de uma consulta ao mesmo tempo, como foi mostrado na figura III.2. Desta forma, o tempo de resposta de um nó será diferente dos demais, variando de acordo com a ordem de inicialização da consulta. O tempo de resposta de um nó de ordem $\boldsymbol{i}$ na execução de determinada consulta é mostrado abaixo:

$$^{nu}R_i = \alpha i + cQ_i \tag{3.4}$$

Tempo de resposta de nó de ordem i (fragmentação não uniforme)

$\boldsymbol{\alpha i}$ é obtido pela soma dos tempos de inicialização ($\boldsymbol{i}$ - 1) dos nós anteriores com o tempo gasto pelo próprio elemento. Q_i representa a quantidade de tuplas ou objetos armazenados e manipulados pelo nó $\boldsymbol{i}$. É necessário ressaltar que o tempo de resposta de um nó compreende o tempo no qual ele está sem atividade, isto é, esperando uma inicialização feita pelo nó mestre através da troca de mensagens. Neste tipo de fragmentação os nós terminaram a execução de uma consulta no mesmo tempo, logo os tempos de respostas de cada nó são iguais.

$$R_i = R_{i+1}, \forall\ i \in [1, N]$$

Assim sendo a equação 3.4 é aplicada para i e $i+1$ obtendo-se:

$$^{nu}R_i = {}^{nu}R_{i+1} \Rightarrow \alpha(i+1) + cQ_{i+1} = \alpha i + cQ_i \Rightarrow Q_{i+1} = Q_i - \frac{\alpha}{c}$$

A partir do desenvolvimento acima segue-se:

$$Q_i = Q_1 - (i-1)\frac{\alpha}{c}$$

O valor de Q_1 é obtido através da soma da progressão aritmética acima obtendo-se:

$$Q_1 = \frac{Q}{N} + \frac{N-1}{2}\frac{\alpha}{c} \tag{3.5}$$

Número de objetos armazenados no nó 1.

O tempo de resposta de um sistema com uma fragmentação não uniforme dos dados é igual ao tempo de reposta do primeiro nó a ser inicializado na execução de uma consulta. Com as equações 3.4 e 3.5 é possível calcular o tempo de resposta abaixo:

$$^{nu}R = R_1 = \alpha + cQ_1 = \alpha\frac{N+1}{2} + c\frac{Q}{N} \tag{3.6}$$

Tempo de reposta para uma fragmentação não uniforme

Com a equação acima é calculado o número ótimo de processadores para encontrar um tempo de resposta mínimo. Aplicando este número na equação 3.4, obtém-se o tempo de resposta ótimo do sistema na execução da consulta. Logo:

$$\left.\frac{d\ ^{nu}R}{dn}\right|_{N = {}^{nu}N_0} = 0 \Leftrightarrow \frac{\alpha}{2} - c\frac{Q}{{}^{nu}N_0^2} = 0 \Leftrightarrow {}^{nu}N_0 = \sqrt{2\frac{cQ}{\alpha}}$$

Aplicando esta equação, o tempo de resposta do sistema utilizando o número ótimo de nós é:

$$^{nu}R_0 = \sqrt{2}\sqrt{\alpha c Q} + \frac{\alpha}{2} \tag{3.7}$$

Tempo de resposta ótimo para uma fragmentação não uniforme

Comparando as equações 3.6 e 3.1 é possível gerar o gráfico, representado na figura III.3, inserindo valores para α, Q e c. No gráfico é observado que para valores pequenos de N os tempos de respostas são equivalentes. Em (BASSILIADES,

VLAHAVAS, 1995) observa-se que no ponto ótimo, o tempo de resposta da fragmentação não uniforme é 44% (quarenta e quatro porcento) menor do que o tempo da distribuição uniforme dos dados.

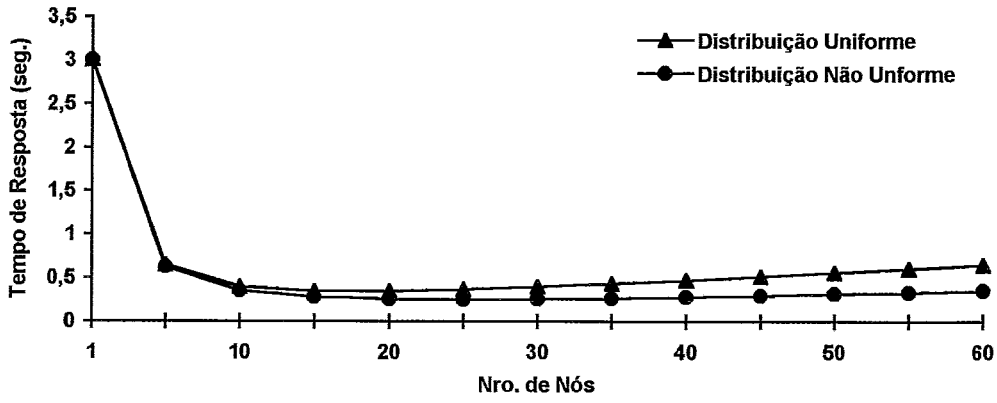


Figura III.3 - Tempos de Resposta ($Q = 3 \text{ X } 10^6$ objetos; $c = 10^{-2}$; $\alpha = 10^{-5}$)

Outro ponto importante a ser lembrado é o fato de que as distribuições demonstradas anteriormente têm como objetivo diminuir o tempo de resposta de uma determinada consulta. Entretanto, na prática, um sistema executa diversas consultas, de vários usuários. Assim, além de diminuir o tempo de resposta, o sistema objetiva também atender consultas simultaneamente (*throughput ou vazão*). Porém, as demonstrações feitas acima podem ser utilizadas nas estratégias de distribuição e alocação de dados. A próxima seção apresenta os principais tipos de estratégias de distribuição dos dados em sistema de banco de dados com memória distribuída.

III.3 ESTRATÉGIAS DE DISTRIBUIÇÃO

Esta seção tem por objetivo analisar as estratégias de distribuição utilizadas para otimizar a execução de consultas simples com predicados de igualdade e faixa de valores à base de dados. Entendem-se como consultas simples aquelas executadas na mesma coleção de objetos, ou consultas de navegação simples de relacionamentos de cardinalidade (1:1). Consultas entre coleções de objetos de classes diferentes estão fora do escopo desta seção.

O balanço de carga em relação aos dados é conseguido sem problemas nas arquiteturas de disco compartilhado e memória compartilhada. Em arquiteturas de memória distribuída procura-se o balanço de carga dos dados, de forma que as operações sejam executadas por cada nó independentemente, em conjuntos de dados

armazenados localmente. Esta independência é conseguida através da divisão dos dados de uma relação segundo critérios.

Os critérios adotados para a distribuição da base de dados podem ser por faixa de valores, por *hashing* ou ainda *por distribuição circular*. Na distribuição circular, em uma coleção de N objetos é distribuída ao longo dos discos de cada processador. Dessa forma, cada objeto fica armazenado em um nó diferente. A figura abaixo exemplifica esta situação. Objetos com identificadores lógicos de valores de *1* até *N* são armazenados em nós diferentes da arquitetura.

III.3.1 DISTRIBUIÇÃO CIRCULAR

Na distribuição circular, em uma coleção de *N* objetos é distribuída ao longo dos discos de cada processador. Dessa forma, cada objeto fica armazenado em um nó diferente. A figura abaixo exemplifica esta situação. Objetos com identificadores lógicos de valores de 1 até N são armazenados em nós diferentes da arquitetura.

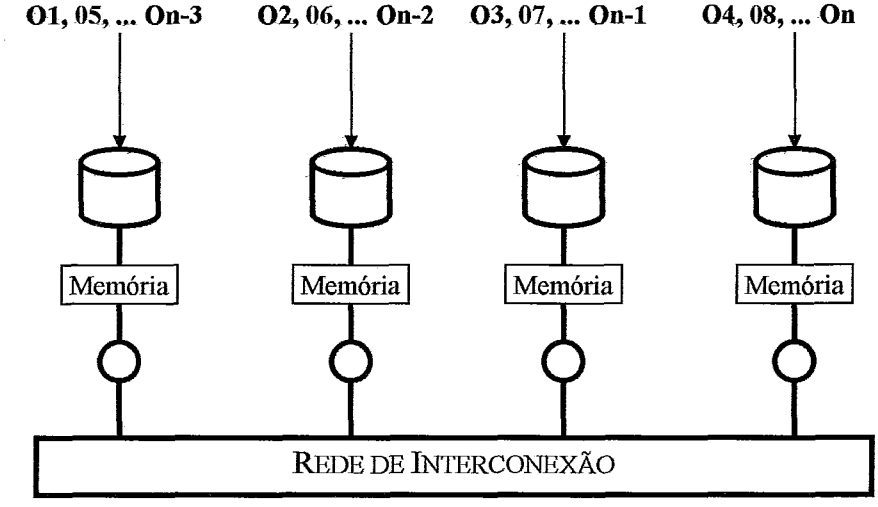


Figura III.4 distribuição circular

Na distribuição circular, as consultas feitas à coleção fragmentada serão executadas por todos os nós do sistema, pois não é possível estabelecer um critério para a identificação da localização dos dados requeridos pelas consultas.

III.3.2 DISTRIBUIÇÃO POR FAIXA DE VALORES

Outro critério utilizado é a fragmentação horizontal de uma coleção segundo os valores de um atributo (por faixa de valores). Cada objeto de uma coleção será armazenado em um nó dependendo do valor apresentado por este atributo. A

modificação do valor daquele atributo deve alterar também a sua localização. Esta distribuição por faixa de valores proporciona a execução de mais de uma consulta ao mesmo tempo. Uma consulta que tem como predicado o atributo utilizado na fragmentação poderá ser executado por um conjunto de processadores, liberando outros nós para executarem novas consultas. A figura III.5 mostra como estão distribuídas uma coleção de objetos de uma classe "Empregado" segundo o seu atributo salário que tem uma variação de 0 até 10000.

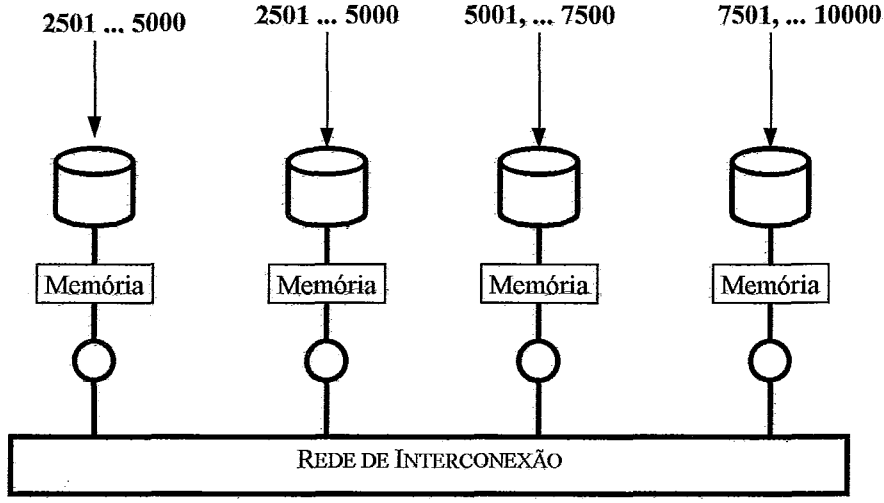


Figura III.5 distribuição por faixa de valores

A figura acima apresenta uma distribuição de carga uniforme. Este tipo de distribuição pode ser prejudicial caso um processador fique muito sobrecarregado, isto é, seja responsável pela grande parte da base de dados. Esta hipótese pode acontecer no exemplo acima, caso exista maior número de empregados recebendo menores salários. Uma melhor distribuição é a formação de fragmentos menores e uniformes e a distribuição circular destes fragmentos. A seção III.5 explica melhor esta maneira. É importante observar que com esta distribuição é possível prever onde uma consulta será executada. Basta que esta consulta tenha em seu predicado uma comparação com o atributo utilizado ("salário") na fragmentação. Dependendo da faixa de valor, a consulta será executada por um ou mais nós.

III.3.3 DISTRIBUIÇÃO POR TABELA DE DISPERSÃO

Esta distribuição caracteriza-se pela aplicação de uma função matemática que tem como parâmetro um atributo de uma coleção da base de dados. Como resultado, a função apresenta o disco do processador onde está armazenada a parcela dos dados

procurados pela consulta. Este tipo de distribuição é mais freqüente na otimização de consultas que envolvem operadores de junção (*join*). Nestes algoritmos, a função é aplicada em cima do identificador do objeto que representa a chave utilizada nas consultas. Porém, existem algoritmos de distribuição por *hashing* (BASSILIADES, VLAHAVAS, 1995), (GHANDEHARIZADEH, DEWITT, 1989) que são utilizados na identificação dos nós que executam consultas simples à base de dados. Estas consultas caracterizam-se pelo uso de predicados de igualdade. Em (BASSILIADES, VLAHAVAS, 1995), a chave pode ser tanto representada por um atributo como pelo próprio identificador do objeto. O identificador é utilizado como chave no caso de consultas onde o predicado será avaliado em todos os objetos da coleção. Quando a chave é representada por um atributo específico, o sistema poderá determinar quais nós serão designados para executar a consulta. Este caso possibilita a execução de consultas simultaneamente, aumentando a *vazão* do sistema.

III.3.4 COMPARAÇÃO DAS ESTRATÉGIAS

As estratégias que utilizam distribuição por faixa de valores e tabela de dispersão permitem um aumento da *vazão* do sistema. Já na distribuição circular, a capacidade de processamento do sistema é utilizada totalmente, isto é, todos os processadores são envolvidos na solução da pesquisa dos dados.

Quanto à uniformidade da distribuição dos dados a distribuição circular apresenta sempre uma uniformidade; já nas outras duas estratégias os dados podem ou não estar distribuídos uniformemente. Como estas estratégias estão associadas aos valores dos atributos de uma coleção, é normal presumir que uma distribuição não uniforme é mais freqüente. Porém, esta pode implicar queda de desempenho do sistema, no caso de consultas que possuam predicados não associados ao atributo utilizado pela distribuição.

Em um sistema de informação é comum uma procura a uma determinada informação especificamente por alguns valores. Nesta hipótese, é comum a maioria dos predicados de um tipo de consulta referir-se a uma determinada faixa de valor de um atributo. Nas estratégias de faixa de valores e tabela de dispersão, alguns nós do sistema ficariam muito sobrecarregados, já que a maioria das consultas seria direcionada para estes nós. Uma solução é fazer uma distribuição não uniforme entre os processadores do sistema para que mais nós possam ser utilizados nas consultas mais freqüentes. Porém, esta solução acarreta perda de desempenho no caso de execuções de consultas que não

contêm em seus predicados o atributo usado na distribuição. Esta perda é caracterizada pelo processamento de maior quantidade de dados pelos nós que armazenam os dados acessados menos freqüentemente. A inicialização prévia dos nós mais carregados poderia ser uma técnica utilizada para amenizar a perda de desempenho no sistema, como mostrado na seção III.2. (GHANDEHARIZADEH, DEWITT, 1989) apresenta uma estratégia de fragmentação de dados que propõe uma melhor solução para este problema. Esta estratégia será comentada mais detalhadamente no capítulo IV.

A estratégia apresentada em (GHANDEHARIZADEH, DEWITT, 1989) baseia-se na freqüência de execução de consultas à base. Esta estratégia possibilita fragmentar a base com relação a vários atributos de uma coleção. Para tal, utiliza-se uma estrutura de armazenamento denominada Grid-File (NIEVERGELT, HINTERBERGER, 1984).

Esta estrutura de armazenamento também é aplicada como ferramenta na distribuição de dados em (LIU, SHASHI, 1996). Este trabalho contém um algoritmo que baseia-se na construção de grafos de similaridade (KARYPIS, KUMAR, 1996a) (KARYPIS, KUMAR, 1996b) e nas informações de consultas como tipo, tamanho e freqüência. Este método maximiza as chances de dados, com as mesmas características, estarem em discos diferentes do sistema. A localização em diferentes discos é proposital, pois o sistema possui a habilidade de acessar várias unidades de disco simultaneamente.

Geralmente, todas as estratégias são empregadas com o objetivo de melhorar o desempenho do sistema nas consultas ocorridas mais freqüentemente. Porém, quando a base de dados está sendo criada, pode ou não existir o conhecimento prévio do volume de operações executadas. Na maioria dos casos, há uma análise, ou seja, uma estimativa de quais os tipos de consultas serão utilizadas com maior freqüência, e não quais os valores empregados em seus predicados aparecem mais freqüentemente. Assim, o capítulo IV trata da utilização destas estratégias de forma dinâmica, possibilitando que as informações relativas à maneira de acesso aos dados, freqüências de consultas e informações de desempenho dos nós sejam captadas durante a execução do sistema.

Capítulo IV

DISTRIBUIÇÃO E ALOCAÇÃO DINÂMICA DE DADOS EM SGBD PARALELOS

Um sistema que tenha a capacidade de obter informações de desempenho serve, no mínimo, como auxiliar para o projetista planejar uma nova distribuição dos dados. Uma redistribuição dos dados faz-se necessária quando há uma queda de desempenho no sistema devido à modificação dos valores dos dados ou à modificação da forma de acesso à base através das consultas. A utilização das estratégias de distribuição de dados de maneira dinâmica gera vários fatores que devem ser analisados. Serão apresentados neste texto os seguintes aspectos: as informações que devem ser captadas pelo sistema e a forma de execução da distribuição e alocação dos dados (com o sistema parado ou em funcionamento).

IV.1 INFORMAÇÕES CAPTADAS PELO SISTEMA

O primeiro aspecto mencionado está associado às estratégias que baseiam-se em algum tipo de informação relativa ao uso do sistema. Na maioria dos casos, estão presentes as informações das consultas e de suas freqüências. Este tipo de informação pode ser facilmente captada. Em um sistema paralelo de arquitetura de memória distribuída, o nó mestre fica encarregado de captar esta informação. Esta tarefa poderia ser feita antes ou depois de executada a consulta. No primeiro caso, ao receber uma solicitação, o nó mestre faria a atualização da freqüência (incrementando seu valor de um), assim como do tipo de consulta que está sendo realizada (caso a consulta não tivesse aparecido previamente). No segundo caso, o nó mestre computaria apenas as consultas realmente executadas pelo sistema, evitando calcular consultas que não foram bem sucedidas devido a algum erro.

Além das informações relativas à freqüência e aos tipos de consultas realizadas, outras informações poderiam ser aproveitadas pelas estratégias de informação. Destacam-se entre elas: a capacidade de processamento de cada nó, os tempos médios de resposta em cada nó, o tempo de processamento, o tempo de entrada e saída e a comunicação para cada consulta. Os tempos médios e a capacidade de processsamento podem ser úteis em algoritmos que utilizam o conceito de nó lógico, mencionado no

capítulo II. A ordenação dos nós segundo o critério de desempenho pode utilizar também estas informações. Os tempos médios de alocação de recursos de processamento, entrada e saída e comunicação de cada consulta, são valores geralmente estimados e utilizados por simulações. Uma substituição possível é utilizar o tempo de resposta médio para cada consulta. Neste tempo estarão representados os tempos de processamento, entrada e saída e comunicação. Esta alternativa é uma medida real, calculada pelo nó mestre através do armazenamento dos tempos de respostas das consultas.

Com base na explanação anterior, é possível concluir que o sistema deve possuir nos seus nós um módulo de informações. Este módulo está associado com o tipo de estratégia utilizada. A utilização de algum tipo de protocolo ou linguagem ajuda na manutenção do sistema. Com esta medida, novas informações podem ser requeridas sem que haja uma mudança no módulo, isto é, este deve estar preparado para fornecer outros tipos de informação e não somente as implementadas para suportar a estratégia utilizada no momento. Outra alternativa é o aproveitamento do conceito de agentes de softwares (GENESERETH, KETCHPEL, 1994) que possam fornecer informações relevantes ao projetista, alimentando simultaneamente a estratégia utilizada.

IV.2 DISTRIBUIÇÃO DINÂMICA – COM O SISTEMA PARADO OU EM EXECUÇÃO

IV.2.1 DISTRIBUIÇÃO COM O SISTEMA PARADO

A estratégia de distribuição pode ser executada de duas maneiras diferentes. Uma forma é a aplicação da estratégia com o sistema parado, ou seja, sem receber consultas. Desta forma, após a determinação dos fragmentos pela estratégia, é feita a associação de cada fragmento a um nó do sistema. Feita esta associação, a próxima fase é a alocação dos dados em cada nó. Com o sistema parado, as duas primeiras fases são de simples realização. A fase de alocação é feita com trocas de objetos entre os nós do sistema. A troca de dados pode ser feita em blocos de objetos, isto é, cada nó envia todos os objetos que outro nó precisa de uma só vez. Esta técnica evita muitas trocas de mensagens na rede. A utilização do mestre como controlador da distribuição também evita a troca de mensagens. O nó mestre é o encarregado de fornecer a identificação dos objetos que cada nó deve armazenar, além de controlar as trocas de dados entre eles.

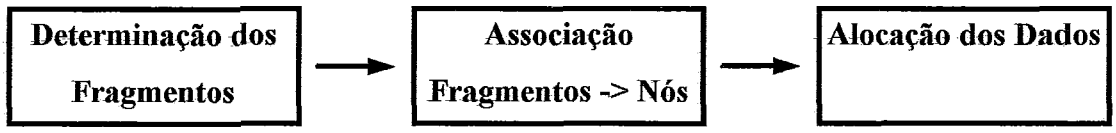


Figura IV.1 - Fases de execução da estratégia de distribuição

IV.2.2 DISTRIBUIÇÃO EM EXECUÇÃO

A distribuição dos dados pode ser feita enquanto o sistema está em funcionamento. Este tipo de distribuição é chamado de distribuição dinâmica Em Execução ou On-Line. Mais complexa do que a estática, a distribuição on-line possui outros fatores complicadores descritos nas subseções seguintes.

IV.2.2.1 A EXECUÇÃO DA ESTRATÉGIA

Para não afetar o funcionamento do sistema, a execução da estratégia pode ser feita por um nó externo. Este nó não armazena nenhuma parte da base de dados, mas tem conhecimento da base, de seu esquema e de como está distribuída. As informações captadas pelo sistema são enviadas para este nó. A aplicação da estratégia é executada em paralelo. Durante esta fase, acessos à base de dados são necessários para a formação dos fragmentos. Isto é possível através da inclusão de consultas feitas pelo nó de distribuição na carga de consultas. O nó de distribuição neste momento participa como um usuário comum do sistema, enviando uma consulta ao nó mestre, e recebendo os objetos selecionados. É importante ressaltar que a inclusão destas consultas não afeta as informações de desempenho, pois estas informações já foram enviadas anteriormente. Além disso, estas consultas não são consideradas no cálculo de tempos e medidas de desempenho. O sistema apenas executa a consulta sem armazenar as informações relativas àquela operação. O problema causado por esta interferência na carga de consultas, está na mudança dos buffers de páginas ou de objetos utilizados pelos sistemas de banco de dados em cada nó. Porém, para um grande número de consultas esta interferência é amenizada.

IV.2.2.2 A Alocação de Objetos em tempo de Execução

A realocação dos objetos da base de dados deve ser feita simultaneamente com a execução das consultas. Como já foi dito anteriormente, a utilização de identificadores físicos acarreta para cada mudança de localização de um grupo de objetos a atualização

de todos os outros que os referenciam. Logo, o identificador físico impede que as consultas continuem sendo executadas, pois o sistema permanece parado realizando as modificações necessárias relativas ao deslocamento de dados de um nó para outro. O identificador lógico apresenta-se como melhor alternativa, já que as modificações ficariam restritas às tabelas de identificadores residentes em cada nó. Porém, a existência de réplicas das tabelas de identificadores de cada nó implicaria trocas de mensagens entre os nós do sistema. A existência de apenas uma tabela de identificação em uma área em disco compartilhada é uma mudança benéfica, pois evita a troca de mensagens. Além disso, mantém a localização dos objetos atualizada durante a execução das consultas. Porém o acesso compartilhado da mesma área em disco pode ser um ponto de estrangulamento de desempenho. O custo deste compartilhamento vai depender muito do sistema de arquivos do sistema operacional utilizado e da velocidade da rede. Outro aspecto a ser analisado é a utilização deste recurso compartilhado quando a máquina paralela é formada por muitos nós. Os acessos serão muitos, gerando esperas indesejáveis pelos nós que querem pesquisar a tabela de identificadores. Uma solução para este problema seria criar ou aumentar a área de buffer de identificadores, evitando a leitura repetida de um mesmo dado. Porém, quando a tabela for modificada os buffers dos sistemas devem ser verificados para espelharem a realidade dentro do sistema.

IV.3 A FASE DE ALOCAÇÃO

Uma característica que afeta a fase de alocação de dados é a forma de implementação do identificador do objeto. Basicamente existem dois tipos de implementação: identificador físico e identificador lógico.

O identificador físico caracteriza-se pela representação da posição no disco onde o objeto está representado. O mesmo pode ser um endereço da página onde o objeto está armazenado, seguido de um deslocamento. Em sistemas distribuídos, este identificador deve informar em qual base ou volume o objeto está situado. Em sistemas paralelos, a informação desejada é a localização do nó do objeto. A vantagem do identificador físico é que, através do seu valor, é possível acessar diretamente o objeto. Porém, quando um objeto é movido para uma outra posição, é necessária uma atualização de todos os objetos que o referenciam. Para evitar esta atualização, utiliza-se a gravação de um sinal no lugar do identificador físico do objeto, indicando que ele foi deslocado para outra página. O novo endereço estaria no endereço seguinte ao do identificador (*forward*

address). No entanto, quando ocorrem várias re-alocações, a utilização desta técnica poderia acarretar a existência de uma base degenerada com várias ligações de um nó para outro. O identificador físico, desta forma, caracteriza-se como um fator de complexidade na fase de realocação dos dados.

O identificador lógico é formado por um único valor que não sofrerá modificações. Esta característica representa uma vantagem em sistemas onde o movimento de objetos é freqüente, pois evita o gasto de recursos na atualização das referências de outros objetos da base. A posição onde o objeto está armazenado é conhecida através de uma tabela de indexação entre identificador lógico e a posição atual do objeto. Em sistemas de banco de dados distribuídos, a tabela de associação deverá possuir a informação de localização da base. Em sistemas de banco de dados paralelos, esta tabela conterá o endereço do nó que armazena o objeto. Em ambientes dinâmicos cujo armazenamento dos objetos da base sofre muitas alterações, o identificador lógico tem demonstrado ser o mais apropriado. A responsabilidade de atualizar as mudanças ocorridas na base limita-se apenas à atualização da tabela de identificadores. Em sistemas de banco de dados distribuídos e paralelos, o identificador lógico não somente facilita que estratégias de agrupamento *(clustering)* (GHANDEHARIZADEH, DEWITT, 1994) (TSANGARIS, NAUGHTON, 1991) sejam implementadas em cada volume ou nó respectivamente, como também as alterações de configuração de disco.

IV.4 Estratégia de Distribuição de Múltiplos Atributos

Esta seção tem como objetivo apresentar uma estratégia de distribuição de dados que será utilizada pelo sistema descrito no próximo capítulo. Esta estratégia tem como principais características a utilização de informações relativas às consultas executadas pelo sistema e a possibilidade de haver fragmentação em relação a dois ou mais atributos de uma coleção de objetos. A estratégia de distribuição de múltiplos atributos (EDMA) foi usada em banco de dados relacionais. Desenvolvida pela Universidade de Wisconsin (GHANDEHARIZADEH, 1990), sua eficiência foi testada no sistema GAMMA (DEWITT *et al.*, 1988), obtendo resultados de desempenho superiores às estratégias de faixa de valores ou *hashing*. EDMA também forma fragmentos relacionados a um atributo somente. As próximas subseções respondem as seguintes questões:

- como é a estratégia de distribuição;
- como a estratégia foi adaptada para uma base orientada a objetos;

- como foi adaptada para ser utilizada dinamicamente.

IV.4.1 A ESTRATÉGIA DE DISTRIBUIÇÃO

A estratégia de distribuição de múltiplos atributos utiliza como informação básica para a formação dos fragmentos o conhecimento dos tipos de consultas, a freqüência de ocorrência destas consultas e o gasto de recursos (E/S, CPU, comunicação) em cada consulta.

IV.4.1.1 TIPOS DE CONSULTAS

Os tipos de consultas são avaliados segundo seus predicados. Cada predicado que tenha um atributo diferente é considerado um novo tipo. Consultas que utilizam nos seus predicados um mesmo atributo não são consideradas iguais mas são do mesmo tipo. Consultas que possuem o mesmo predicado são iguais. Por exemplo:

Select x from x in Documentos where x.Data = '01/01/97';

Select x from x in Documentos where x.Data = '01/06/96';

As consultas apresentadas acima não são iguais, mas são do mesmo tipo.

IV.4.1.2 A FREQÜÊNCIA DE OCORRÊNCIA DAS CONSULTAS

Das freqüências de ocorrência de cada consulta são determinadas: as freqüências por tipo de consulta e as freqüências relativas de cada uma. Abaixo é mostrado como são calculadas as freqüências a partir da freqüência simples de ocorrência de consultas.

$FreqC_i$ - quantidade de ocorrência da consulta i.

$FreqTipo_j$ - freqüência de ocorrência de consultas do tipo j, isto é, somatório das ocorrências de consultas que têm o mesmo tipo de predicado. A fórmula é apresentada abaixo, sendo ***s*** o número de consultas de mesmo tipo, ou seja, que contêm em seu predicado o mesmo atributo.

$$FreqTipo_j = \sum_{r=1}^{s} FreqC_r$$

Ocorrências de consultas do tipo j.

$Freq\,\mathrm{Re}lC_i$ - número de ocorrências relativas da consulta **i**.

$$Freq\ \mathrm{Re}\ lC_i = \frac{FreqC_i}{FreqTipo_j} \quad ou \quad \frac{FreqC_i}{\sum_{r=1}^{s} FreqC_r}$$

IV.4.1.3 O TEMPO DE RESPOSTA DO SISTEMA

O tempo de resposta do sistema é calculado utilizando a equação 4.1:

$$TR(M) = \frac{\overline{CPU} + \overline{ES} + \overline{COM}}{M} + M * \alpha \qquad (4.1)$$

Tempo de resposta do sistema

Onde $\overline{CPU}$, $\overline{ES}$ e $\overline{COM}$ são os tempos médios de processamento seqüencial, entrada e saída e comunicação gastos pelas consultas do sistema. α representa o tempo gasto para inicializar e sincronizar uma consulta em cada processador. M representa o número de processadores que participam da execução da consulta.

A estratégia fragmenta uma relação segundo um ou mais atributos. Assim, é aplicada a equação acima para o grupo de consultas que utiliza o atributo que fará parte da distribuição. Caso a distribuição esteja baseada em dois ou mais atributos, a fórmula é aplicada para cada um. Como na seção III.2, calcula-se o número ótimo de processadores para atingir um tempo de resposta mínimo. Para cada atributo é calculado um número de processadores. Estes números, junto com as freqüências, formam os requisitos para calcular a percentagem dos nós disponíveis pelo sistema que serão alocados para cada atributo. Para cada consulta que tenha um atributo envolvido, o número mínimo de processadores associados será o calculado por esta percentagem. As próximas fórmulas ajudam a esclarecer a estratégia de cálculo do número de processadores.

$$M = \sqrt{\frac{\left(\overline{CPU} + \overline{ES} + \overline{COM}\right)}{\alpha}}$$

A equação acima é a mesma apresentada pela seção III.2, isto é, o número de processadores ótimos para um sistema de distribuição uniforme dos dados. As variáveis cQ foram substituídas pelo somatório dos tempos médios de utilização de recursos na execução das consultas do sistema. O número de processadores ótimos para executar consultas que tenham em seu predicado um determinado atributo θ é calculado

utilizando a equação acima. Somente serão consideradas no cálculo as consultas do mesmo tipo. Além disso, para cada consulta é aplicada a sua freqüência relativa. S representa o número de consultas que utilizam o atributo θ.

$$M_\theta = \sqrt{\frac{\sum_{r=1}^{s} \left(\overline{CPU_r} + \overline{ES_r} + \overline{COM_r}\right) * FreqRelC_r}{\alpha}}$$

Número ótimo de processadores para consultas que utilizam o atributo i

O próximo passo é o cálculo da percentagem dos discos disponíveis pelo sistema que será alocada para cada atributo. Esta percentagem também pode ser denominada como política de espalhamento da *Grid-File*. *Grid-File* é uma estrutura dinâmica de armazenamento projetado para acesso através de múltiplos índices. Esta estrutura dá suporte para consultas do tipo faixa de valores e consultas com predicados parcialmente especificados. A *Grid-File* armazena um disco com um tamanho fixo de armazenamento. Esta unidade é chamada de blocos. (NIEVERGELT, HINTERBERGER, 1984) utiliza a terminologia de *bucket* para definir uma unidade de armazenamento. Cada *bucket* tem uma capacidade de c objetos. A *Grid* mantém uma correspondência entre os blocos, isto é, seus elementos e os buckets.

A criação de uma *Grid-File* é melhor explicada com um exemplo. Para simplificar é mostrada a criação para apenas duas dimensões. A *Grid-File* é criada para armazenar objetos com relação aos atributos a e b. Cada atributo vai representar um eixo (dimensão) da *Grid-File*. Supondo-se que cada *bucket* suporta três objetos no máximo, a inserção dos três primeiros objetos é feita sem problemas (figura IV.2).

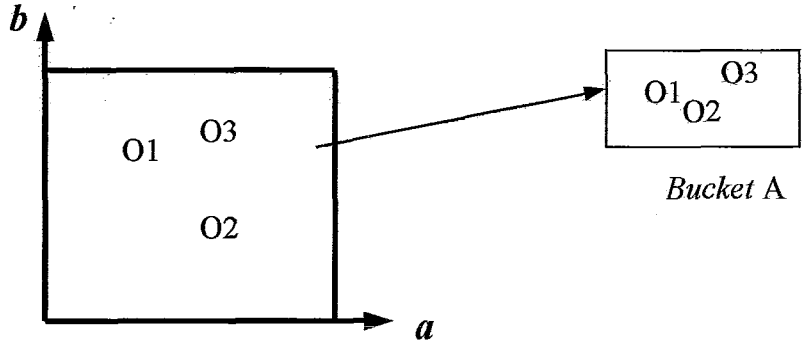


Figura IV.2 - Inclusão dos primeiros três objetos

Quando há inclusão de mais um objeto, o *bucket* A está completamente cheio. Logo, a *Grid-File* divide-se no eixo do atributo ***a***. Outro *bucket* é criado e os objetos são distribuídos para cada bucket em relação ao valor do atributo (no caso a). A figura IV.3 representa esta operação. A figura IV.4 mostra a *Grid-File* depois da inclusão de mais um quinto objeto. É importante observar que o *bucket* B está sendo apontado por duas áreas da *Grid-File*, isto é, o *bucket* B representa dois blocos de disco.

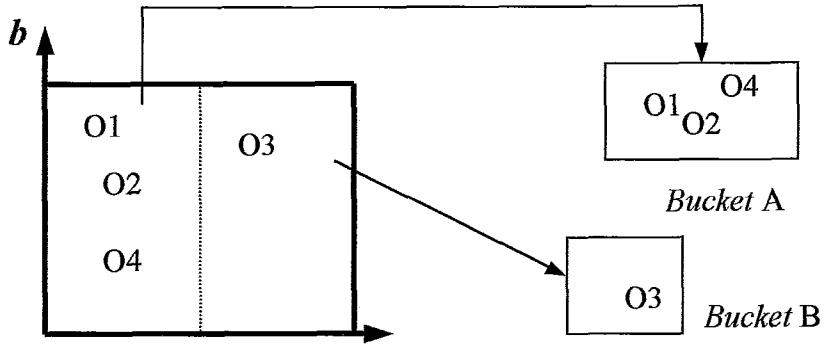


Figura IV.3 - Inclusão do quarto objeto

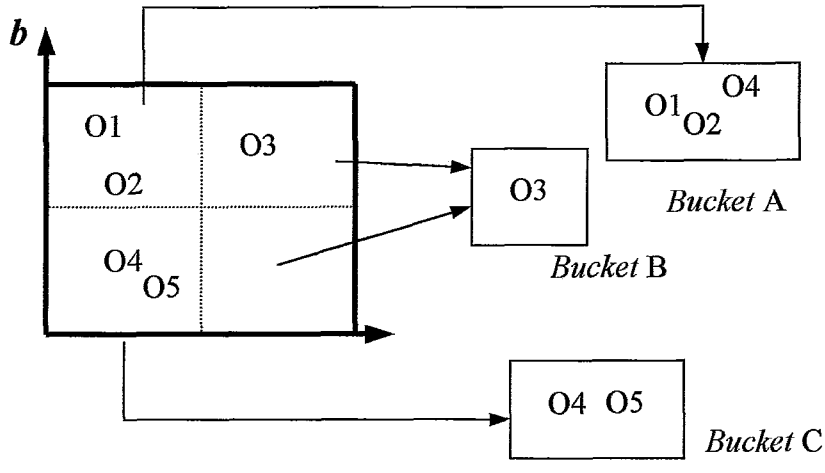


Figura IV.4 - inclusão do quinto objeto

A política de divisão da *Grid-File* adotada no exemplo acima é denominada cíclica, isto é, o mesmo número de divisões para cada dimensão ($N_i = N_j$). Esta estratégia é utilizada somente quando o número de processadores calculado para cada dimensão for em igual. Quando estes valores são diferentes, é calculada uma proporção através da equação abaixo.

$$\%Espalhamento_i = \frac{FreqCTipo_i}{Nro.Total\ Cons.} * \left(\frac{\sum_{j=1}^{K} M_j - M_i}{\sum_{j=1}^{K} M_j} \right) \quad (4.3)$$

Política de espalhamento da Grid-File

Esta proporção define a política de espalhamento da *Grid-File*. A *Grid* é dividida ao longo dos eixos de maneira que respeite a proporção calculada acima. Uma coleção de objetos que é fragmentada segundo dois atributos ***a*** e ***b*** tem como número de processadores $M_a = 3$ e $M_b = 1$. Assumindo que noventa porcento das consultas realizadas utilizam o atributo ***a*** e dez porcento o atributo ***b***. Utilizando a equação 4.3 a porcentagem de espalhamento para os atributos ***a*** e ***b*** são, 22.5 e 7.5 respectivamente. Destes valores conclui-se que ao final da fase de criação da Grid-File o número de divisões da dimensão correspondente ao atributo ***a*** é três vezes maior que a do atributo ***b***.

O próximo passo após a criação da *Grid-File* é a associação de seus elementos aos processadores do sistema. Para cada elemento ou bloco da *Grid-File* está associado um conjunto de objetos, que serão armazenados em um único nó. Esta fase tem de satisfazer dois fatores conflitantes. O primeiro caracteriza-se pelo aproveitamento de alguns processadores para cada faixa de valor de um atributo, permitindo que consultas com faixas de valores diferentes sejam executadas simultaneamente pelo sistema, aumentando o *throughput ou vazão*. O segundo é o aproveitamento total da capacidade de processamento do sistema, fazendo com que uma consulta seja executada por todos os processadores, diminuindo o tempo de resposta. Ghandeharizadeh (GHANDEHARIZADEH, 1990) criou uma heurística que concilia estes dois fatores conflitantes, possibilitando melhores tempos de respostas e maiores *throughput* do que outras estratégias como faixas de valores e *hashing*. Esta heurística é apresentada no Anexo I.

IV.4.1.4 QUANDO SOMENTE UM ATRIBUTO PARTICIPA DA DISTRIBUIÇÃO

Quando a fragmentação da base é aplicada somente em um atributo, a estratégia não adota o comportamento explicado acima. A estratégia como descrita anteriormente, calcula o número ótimo de processadores para aquele atributo. Após este cálculo, é

utilizada uma outra informação: o número de objetos selecionados pelas consultas do sistema. Com esta informação é possível calcular o número de fragmentos existentes através da seguinte equação:

$$NF = \frac{\overline{ObjPorCons}}{M} \tag{4.4}$$

Números de fragmentos

Onde $\overline{ObjPorCons}$ é o número médio de objetos selecionados pelas consultas do sistema. Em (GHANDEHARIZADEH, 1990), este número é estimado já que existe o conhecimento da base. Sabendo-se, por exemplo, que a base tem uma distribuição de valores uniformes em relação ao atributo, é possível calcular este número médio de objetos selecionados pelas consultas. Desta forma, a base será fragmentada em ***NF*** partes como é feito na estratégia de faixa de valores. A associação dos fragmentos aos nós é feita de maneira circular. A distribuição dos fragmentos de forma circular possibilita que uma consulta possa ser executada por mais de um processador, melhorando o tempo de resposta do sistema. Na faixa de valores a mesma consulta tem mais probabilidade de ser executada em número menor de nós. A figura IV.5 mostra como é feita a associação dos nós com 100 fragmentos de uma coleção de 10000 objetos. O número *M* representa o número ótimo de processadores. O número *P* representa o número de processadores existentes.

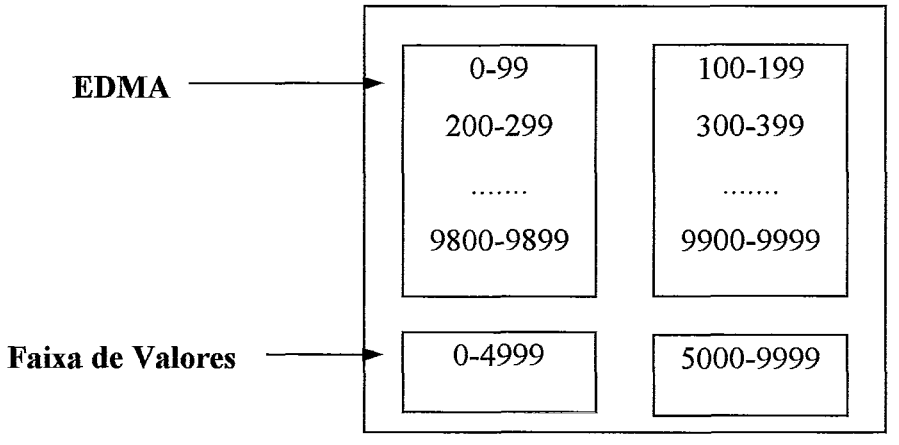
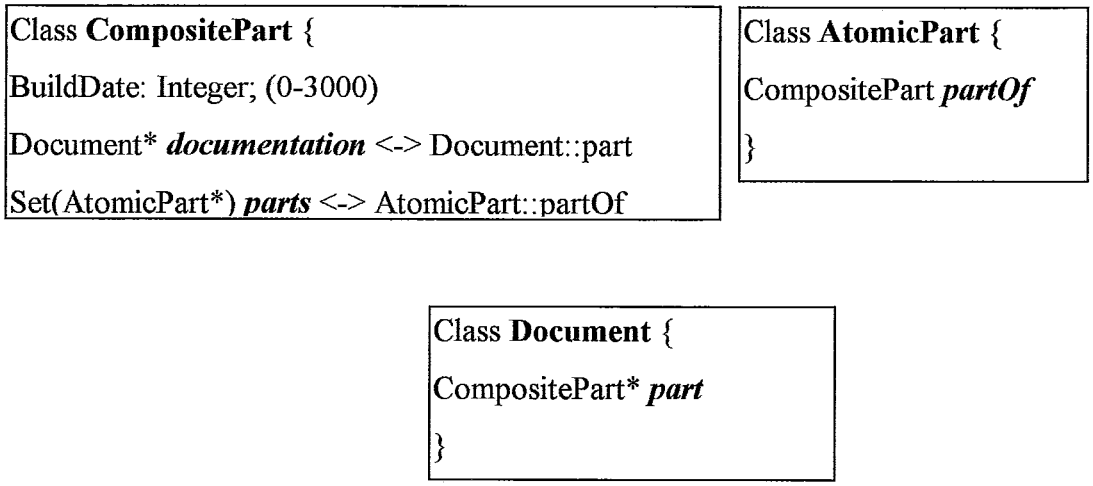


Figura IV.5 - fragmentação baseada em um atributo (*M=5 e P=2*)

IV.5 ADAPTAÇÃO DA ESTRATÉGIA PARA A ORIENTAÇÃO A OBJETOS

Para utilização da estratégia no modelo orientado a objetos, decisões tiveram que ser tomadas e implementadas nesta tese. Estas decisões estão classificadas segundo a cardinalidade dos relacionamentos existentes entre objetos da coleção alvo da fragmentação e os diversos objetos da base. Um objeto pode conter vários atributos que são atômicos ou são referências a outros objetos existentes na base. Estas referências são feitas através dos identificadores dos objetos relacionados. Como exemplo, são apresentadas na figura IV.6 três classes tiradas do benchmark OO7 (CAREY *et al.*,



1994). Foram excluídos os atributos não utilizados. Na figura é possível observar dois tipos de relacionamentos com cardinalidades diferentes: 1-1, e 1-M. Além destes dois relacionamentos, esta seção explica também o caso de relacionamentos com cardinalidade N-M.

Figura IV.6 – Classes do benchmark OO7

IV.5.1 RELACIONAMENTOS 1-1

Neste caso, os objetos têm uma referência exclusiva um do outro. Este relacionamento está representado na figura IV.6, nos atributos "documentation" e "part" das classes "CompositePart" e "Document" respectivamente. Este tipo de relacionamento pode fazer parte da estratégia de fragmentação como sendo um atributo simples. Como exemplo é proposta a seguinte situação: o sistema executa em grandes quantidades consultas relacionadas à coleção de "CompositePart". Estas consultas têm em seus predicados os seguintes tipos de comparações: "BuildDate > 1950" e "documentation->id = '100'. A primeira comparação é feita com um atributo do objeto "CompositePart"; já a segunda é feita com um atributo do objeto "Document"

relacionado com "CompositePart" através do atributo "documentation". A estratégia utiliza estes dois atributos fragmentando primariamente as "CompositeParts" segundo o atributo "BuildDate". A outra dimensão é composta do atributo "documentation". Os objetos "CompositePart" são fragmentados derivadamente em relação aos objetos "documentation". Sendo assim, a *Grid-File* teria a seguinte forma:

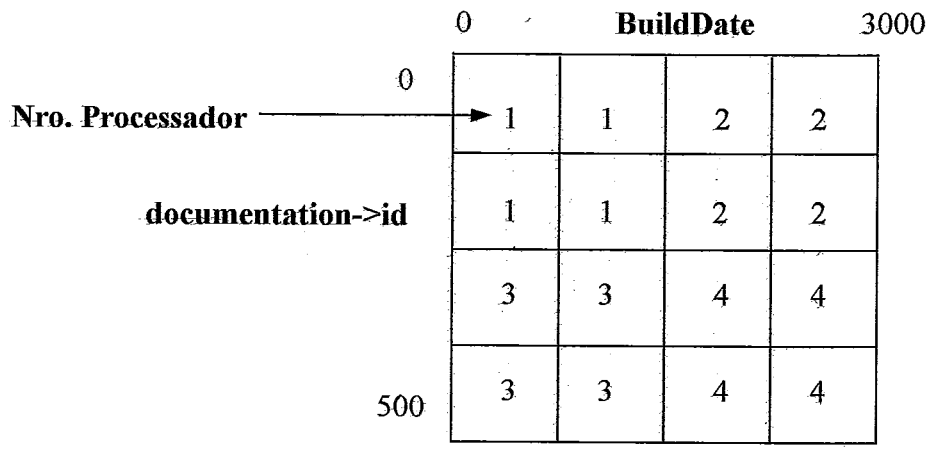


Figura IV.7 – Grid-File

IV.5.2 RELACIONAMENTOS 1-M

Neste tipo de relacionamento um objeto pode fazer referências a um ou a mais objetos de outra classe. Este caso está representado pelos atributos "partof" e "parts" das classes "AtomicPart" e "CompositePart" respectivamente. No exemplo descrito anteriormente, um objeto "AtomicPart" é armazenado no processador onde reside a maioria dos objetos "CompositePart" que o referenciam. A coleção de "AtomicParts" sofre uma fragmentação derivada.

IV.5.3 RELACIONAMENTOS M-N

Assim como no caso anterior, as coleções que não participam diretamente da estratégia de fragmentação são distribuídas derivadamente pelos nós. Porém, neste tipo de fragmentação, normalmente os objetos não conseguem ficar agrupados em um mesmo fragmento, aumentando a necessidade de comunicação entre os nós. Uma distribuição uniforme destes objetos é importante para não degradar muito o desempenho do sistema.

IV.5.4 EDMA DINÂMICA

Os valores CPU, E/S, e COMM são estimados pelo projetista, pois há o conhecimento da Base de Dados, das consultas, de suas freqüências, assim como do número de tuplas selecionadas nesta consulta. Estes valores representam o tempo médio de execução de uma consulta no sistema. Para utilizar esta estratégia em tempo de execução, é observado que estes valores médios podem ser substituídos pelos tempos de execução das consultas do sistema. Entende-se como tempo inicial de execução de uma consulta o momento onde o nó escravo, após comunicar-se com o nó mestre para receber a assinatura da consulta e outros dados de protocolo, inicia efetivamente a execução da consulta. Após esta execução, é obtido um conjunto de objetos na forma de seus identificadores. O nó mestre recebe este conjunto do nó escravo e repassa-o para o usuário. O tempo de execução da consulta é marcado pelo começo efetivo da consulta no nó escravo e o recebimento da lista de objetos selecionados pelo nó mestre. Este tempo vai incluir o tempo de processamento da consulta, o tempo de entrada e saída relativo à busca dos objetos no disco, além de captar também o tempo gasto em troca de dados e mensagens como outros nós relativos ao acesso a dados remotos (situados em outros processadores). O tempo de recebimento da assinatura da consulta e a troca de mensagens iniciais entre escravos e mestre ocorrida antes do início da operação também são captados pelo sistema. Estes tempos são somados e atribuídos à variável α que aparece nas equações 4.1 e 4.2. Abaixo são mostradas as equações 4.1 e 4.2 com os termos $\overline{CPU}$, $\overline{ES}$ e $\overline{COM}$ substituídos por $\overline{TempoExecucao}$.

$$TR(M) = \frac{\overline{TempoExecucao}}{M} + M * \alpha$$

$$M_i = \sqrt{\frac{\sum_{r=1}^{s} \left(\overline{TempoExecucao}_r\right) * Freq\,\mathrm{Re}lC_r}{\alpha}} \qquad (5.1)$$

Equações 4.1 e 4.2 modificadas

Em um sistema multi-usuário, várias consultas são executadas ao mesmo tempo. Logo, existe um tempo onde uma consulta fica esperando um determinado processador terminar a execução de outra operação já em andamento. Este tempo de espera não é considerado pela estratégia, pois esta se preocupa apenas em melhorar o desempenho da

execução das consultas. Com o ganho de desempenho, este tempo de espera também diminui. Porém, este tempo é considerado quando se quer medir a vazão do sistema. A próxima seção aborda a avaliação de sistemas paralelos de banco de dados, explicando como é a avaliação das estratégias que podem mudar a distribuição dos dados dinamicamente.

IV.6 AVALIAÇÃO DE ESTRATÉGIAS DE DISTRIBUIÇÃO

Vários são os trabalhos que mostram ganho de desempenho de sistemas paralelos de banco de dados através da distribuição dos dados. Porém, estes trabalhos diferem na forma de avaliação. (MEYER, 1997) faz uma avaliação mostrando ganhos na execução de determinadas consultas, com fragmentações de dados específicos. Para cada tipo de consulta é gerada uma ou mais fragmentações, comparando os dois resultados. Além da comparação dos resultados entre as diversas fragmentações, há também a confrontação com o tempo de execução da consulta em um único nó. Neste tipo de avaliação, a consulta é executada um certo número de vezes, garantindo que o sistema terá em seus *buffers* de páginas ou objetos alguns dados relativos à consulta. Porém, deve-se garantir através da diminuição do tamanho do *buffer*, que cada nó não contém toda a informação necessária para o processamento da consulta. Deste tipo de experimento extraem-se dois valores: o tempo de resposta da primeira execução e o tempo de resposta médio da consulta, isto é, a soma de todos os tempos dividido por seu número de execuções.

Trabalhos como (GHANDEHARIZADEH, DEWITT, 1989) tem como objetivo analisar estratégias e fragmentação de dados para sistemas multi-usuários. Ao contrário da avaliação descrita anteriormente, esta apresenta uma distribuição de dados que será utilizada para um conjunto de consultas. Estes sistemas objetivam medir o tempo de resposta do sistema, assim como o ganho no número de operações executadas durante um determinado intervalo de tempo (*throughput ou vazão*). As próximas seções descrevem as características que influenciam os testes de desempenho deste gênero, além de mostrar o procedimento a ser adotado quando estas estratégias são utilizadas dinamicamente.

IV.7 AVALIAÇÃO DE ESTRATÉGIAS DE DISTRIBUIÇÃO PARA SISTEMAS MULTI-USUÁRIOS

Segundo GHANDEHARIZADEH e DeWitt (GHANDEHARIZADEH, DEWITT, 1989), existem três fatores principais que afetam o desempenho de consultas executadas em sistemas que adotaram estratégias de distribuição de dados. Estes fatores são o número de consultas que são executadas simultaneamente, o grau de utilização de recursos da consulta, e como as consultas são misturadas dentro da carga de trabalho.

IV.7.1 NÚMERO DE CONSULTAS EXECUTADAS SIMULTANEAMENTE

A estratégia pode ter um comportamento diferente à medida que cresce o número de consultas lançadas ao sistema pelos terminais. Desta forma, são geradas análises do tempo de resposta com o crescimento do número de consultas lançadas ao sistema. Esta informação também pode ser utilizada na avaliação da *vazão*. Para simular o envio paralelo das operações, a carga de consultas ao qual o sistema será submetido é armazenada em disco. Processos são executados para ler pedaços iguais da carga enviando-os ao nó mestre. Como conseqüência, temos o lançamento de várias consultas ao mesmo tempo. Se estes processos forem executados em máquinas diferentes, esta situação é ainda mais real. Imagine um sistema onde 2 consultas são solicitadas concomitantemente. Para simular esta situação teríamos um arquivo com um número "N" de consultas. Dois processos seriam criados, cada um sendo responsável pela metade das N consultas executadas. Cada processo desta forma agiria separadamente como um terminal, solicitando a execução de uma consulta de cada vez.

IV.7.2 GRAU DE UTILIZAÇÃO DE RECURSOS DE UMA CONSULTA

Em uma base de dados fragmentada, uma consulta pode ser executada em um ou mais processadores do sistema. Portanto, esta pode ser classificada quanto ao grau de recursos utilizados no sistema. É considerada consulta de alta utilização de recursos aquela onde todos os nós estarão envolvidos na sua execução. Por outro lado, é considerada consulta de baixa utilização, aquela executada em apenas um nó. Em uma avaliação onde se pretende medir o aumento de *vazão*, é interessante utilizar consultas de utilização de recursos nos níveis baixo e moderado. Em um sistema de oito processadores, uma carga apropriada para uma avaliação é composta de consultas que

envolvem em sua execução de 1 a 6 processadores. Esta conclusão deve-se ao fato de que consultas de alta utilização de recursos utilizam quase todos os processadores do sistema acarretando em um *vazão* baixo, independente da estratégia adotada. Em uma base de dados que contenha valores uniformemente distribuídos, o grau de utilização de recursos é controlado através da percentagem de objetos selecionados na coleção. Consultas que selecionam 80% da coleção fragmentada são classificadas como de alto grau de utilização.

IV.7.3 MISTURA DAS CONSULTAS

As consultas formadas para serem executadas pelo sistema devem ser distribuídas aleatoriamente, evitando uma distribuição tendenciosa que favoreça a estratégia adotada. Para estratégias que se baseiam em mais de um atributo (como a EDMA - seção 5), os tipos de consultas devem ser misturados. Além disso, para cada tipo de consulta existe um grau de utilização de recursos. Para dois atributos participantes na estratégia EDMA é aplicada a seguinte combinação de utilização de recursos:

Consultas que utilizam atributo 1	Consultas que utilizam atributo 2
Baixa utilização de recursos	Baixa utilização de recursos
Baixa utilização de recursos	Média utilização de recursos
Média utilização de recursos	Baixa utilização de recursos
Média utilização de recursos	Média utilização de recursos

IV.7.4 AVALIAÇÃO DE ESTRATÉGIAS UTILIZADAS DINAMICAMENTE

A avaliação das estratégias de distribuição está associada às informações utilizadas por cada estratégia na formação e distribuição dos fragmentos. Quando estas estratégias são executadas dinamicamente, a avaliação deve simular uma mudança de situação do sistema que gere queda de desempenho e incentive a aplicação da estratégia. Este incentivo caracteriza-se pela mudança das informações utilizadas para a formação dos fragmentos. No caso da estratégia EDMA, estes dados são os tempos, os tipos e as freqüências de consultas. Uma avaliação possível é modificar as freqüências de

ocorrências de consultas, provocando uma reorganização da base através da aplicação da estratégia. Os tempos obtidos das consultas executadas sobre a nova distribuição são comparados com os tempos provenientes da situação anterior. A figura seguinte determina o procedimento adotado para estratégia EDMA. Este procedimento é utilizado na avaliação feita no capítulo VI.

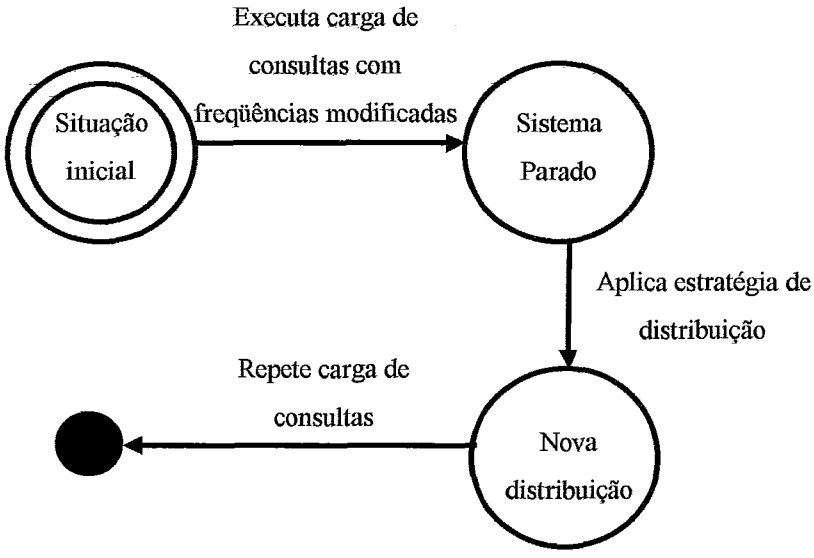


Figura IV.8 - avaliação da estratégia EDMA

Capítulo V

O GERENTE DE DISTRIBUIÇÃO DINÂMICA

V.1 INTRODUÇÃO

Este capítulo busca descrever um sistema paralelo de banco de dados que contém um gerente de distribuição capaz de adquirir informações de acesso e desempenho e, com base nestas informações, modificar a alocação da base de dados, melhorando o desempenho.

Um gerente de distribuição dinâmica em um banco de dados paralelo tem duas características principais. A primeira, comum a qualquer gerente de distribuição, é a de indicar quais os processadores que irão participar de uma consulta. A segunda é a de intervir no sistema, alterando a alocação dos objetos entre os processadores e melhorando o desempenho do sistema. Um gerente de distribuição dinâmico foi implementado neste trabalho. Este gerente foi inserido em um banco de dados paralelo com arquitetura de memória distribuída, anteriormente implementado por (MEYER, 1997) e chamado de PARGOA-MD. O PARGOA-MD é uma evolução de um gerente de objetos armazenados (GOA) implementado nos laboratórios da Coppe Sistemas. As alterações feitas no PARGOA-MD não implicaram mudanças radicais na sua arquitetura e funcionamento. Porém, incluem a característica de realocação de uma base de dados dinamicamente, para melhorar o seu desempenho.

A seção 2 deste capítulo mostra as versões existentes até a implementação do PARGOA-MD. A seção 3 apresenta as características do gerente de distribuição, assim como a arquitetura do sistema paralelo onde ele foi implementado. A quarta seção trata das mudanças que foram necessárias para adaptar o PARGOA-MD e embutir o novo gerente de distribuição. A Quinta seção apresenta o ambiente computacional onde foram realizados os testes de desempenho.

V.2 SISTEMAS ANTERIORES

V.2.1 GOA

O Gerente de Objetos Armazenados é um gerente de objetos que pode ser acoplado a outros sistemas de bancos de dados orientados a objetos. Sua arquitetura é cliente-servidor e apresenta dois módulos principais: o Gerente de Meio de Armazenamento (GMA) e o Gerente de Memória de Objetos (GMO).

O primeiro módulo é responsável pela organização e pelo acesso ao meio de armazenamento do banco de dados (Memória secundária). O segundo módulo é responsável pela gerência da memória primária do sistema. O GOA contém um outro módulo, o PC, responsável pelo processamento das consultas à base de dados.

O identificador de objetos do GOA é um endereço formado por dois campos: o primeiro indica o número da página de disco onde o objeto está armazenado; o segundo representa o deslocamento dentro da página onde o objeto está localizado.

O GOA armazena também a representação da estrutura que compõe um objeto. Para cada objeto são armazenados: os nomes, o tamanho e o valor dos atributos. Estes podem ser simples ou apenas de referência, representando um identificador para outro objeto.

O GOA contém construtores do tipo lista e coleção. O primeiro suporta o armazenamento do tipo lista, possibilitando a representação de atributos multi-valorados. O segundo é utilizado para armazenar identificadores de objetos de uma mesma classe. As consultas são feitas através de procuras em uma coleção de objetos.

Para manipular os objetos, o GOA contém uma série de operações que são disponibilizadas em forma de funções, como por exemplo: buscaobjeto(), insereobjeto(), etc.

V.2.2 PARGOA

O servidor paralelo de objetos ParGOA (TAVAREZ *et al.*, 1996) é um protótipo implementado na máquina paralela desenvolvida pela COPPE, o NCPI, formada por oito nós, cada qual com um processador i860 e 2Mbytes de memória. Suas principais características são:

- *Utiliza o modelo de memória de disco compartilhado onde todos os nós do sistema acessam uma mesma unidade de memória secundária. Cada*

processador gerencia os objetos lidos do disco através de um buffer na memória principal.

- *Executa uma fragmentação circular distribuindo a base de dados uniformemente, sem distinção dos valores dos objetos armazenados.*
- *Faz uma distribuição dinâmica dos segmentos dos objetos no momento da execução da consulta.*
- *Apresenta desagrupamento dos segmentos que armazenam os objetos, podendo ser total ou parcial, dependendo do número de processadores do sistema. Se o número de processadores for menor ou igual a quantidade de segmentos existentes, o desagrupamento é total.*

O PARGOA possui um gerente de coleções responsável por fornecer as informações relativas às coleções alvos referenciadas por uma consulta à base. Com estas informações, o gerente de distribuição determina o número de processadores que irão executar a consulta, enviando para cada um os predicados da consulta e a lista de objetos a serem processados. Estas listas são representadas por uma cadeia de identificadores de objetos formada por uma fragmentação circular, como dito anteriormente. Através destas listas, cada nó será o responsável pela leitura dos segmentos equivalentes. Uma boa política de agrupamento pode influenciar no desempenho das consultas.

Este sistema tem como desvantagem o não aproveitamento das características dos dados dos objetos. Outro ponto que pode ser um "gargalo" no sistema é o acesso compartilhado ao disco. Porém, como mencionado em capítulos anteriores, este tipo de arquitetura facilita a distribuição dinâmica dos objetos em tempo de execução de uma consulta. A figura V.1 mostra a arquitetura do sistema PARGOA.

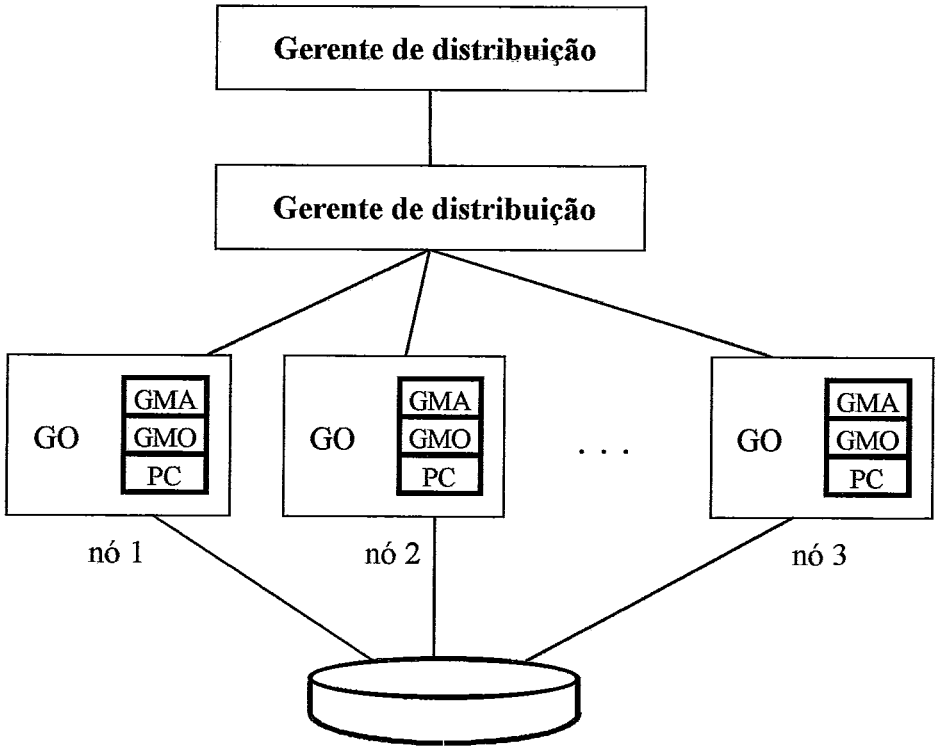


Figura V.1 - Arquitetura do PARGOA

V.2.3 PARGOA-V

Este sistema é uma adaptação do PARGOA para ambientes com paralelismo virtual. A máquina paralela é simulada através do software PVM (GEIST *et al.*, 1994), que possibilita a configuração de uma máquina paralela de várias estações de trabalhos de diversas configurações.

O PARGOA-V mantém as características do seu antecessor: arquitetura de disco compartilhado, utilizando fragmentação dinâmica horizontal e circular na execução da consulta. O gerente de coleções e o gerente de distribuição foram substituídos pelo gerente de tarefas, que recebe a consulta e a repassa para os demais módulos do sistema juntamente com a lista de identificadores necessários para a execução de consultas.

A figura 2 mostra a arquitetura do sistema desenvolvido no laboratório do programa de sistemas da COPPE-UFRJ.

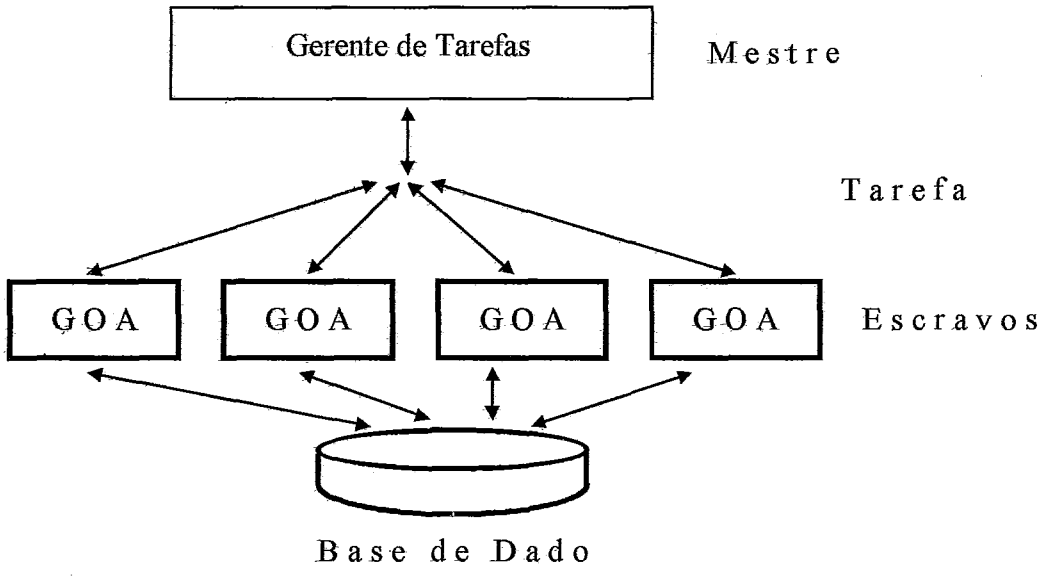


Figura V.2 - Arquitetura do PARGOA-V

V.2.4 PARGOA -MD

Este sistema foi desenvolvido por (MEYER, 1997) com o objetivo de testar diversas estratégias de fragmentação em um sistema paralelo de banco de dados com arquitetura de memória distribuída. Como características principais este sistema apresenta:

- ***Modelo de memória distribuída**, onde cada nó dispõe de um processador, memória principal e um fragmento da base de dados associada.*
- ***Fragmentação horizontal de forma primária**, por faixa de valores, ou de forma derivada, permitindo que classes que se interelacionarem tenham seus fragmentos residindo em um mesmo nó, evitando a necessidade de comunicação entre processadores.*
- ***Transparência de fragmentação e localização**. O nome do fragmento de cada classe é o mesmo em todos os nós e também igual ao seu nome global. Foi modificado o identificador físico do GOA para permitir o encapsulamento do identificador do nó processador. Para isso utilizou-se o recurso de endereçamento estruturado, já disponibilizado pelo GOA. No PARGOA-MD um objeto residente em outro nó era representado em um nó remoto por dois endereços. O primeiro indica o número do nó e um endereço apontando para outra posição. Nesta posição está representado o*

identificador físico do GOA no nó remoto. O PARGOA-MD lê este endereço e se comunica com o nó remoto, solicitando o objeto não residente.

V.3 CARACTERÍSTICAS DO SISTEMA DE ALOCAÇÃO DINÂMICA

Com a utilização dos conceitos apresentados no capítulo 4, foi construído um sistema capaz de realocar a base de dados dinamicamente. Esta realocação é feita de maneira automática, analisando informações de acesso e desempenho do sistema.

A seguir são apresentadas as principais características do sistema:

- *O uso de identificador lógico na representação dos dados;*
- *Arquitetura de memória distribuída;*
- *Fragmentação horizontal por faixa de valores para um ou mais atributos de uma coleção. Esta fragmentação é determinada automaticamente pelo sistema, com base em informações de tempo de consultas, tipos de consultas, freqüência etc (Vide capítulo IV);*
- *Alocação dinâmica do sistema, realizada com o sistema parado ou em execução. A alocação dinâmica é feita de acordo com a fragmentação determinada pelo sistema.*

V.4 IMPLEMENTAÇÃO DO SISTEMA DE ALOCAÇÃO DINÂMICA

Na construção do sistema foram necessárias algumas mudanças na biblioteca do GOA, o modelo de memória foi o mesmo utilizado pelo PARGOA-MD (Memória distribuída). As principais características do sistema apresentadas na seção V.3 são descritas nas próximas subseções.

V.4.1 IDENTIFICADOR LÓGICO

Como já foi visto no capítulo 4, quando os objetos são transportados de um nó para outro, o seu endereço dentro do disco se altera. Esta mudança deve ser considerada em toda base, forçando uma atualização nos objetos que fazem referência ao objeto realocado. Para evitar este problema, foi implementado no GOA o identificador lógico, representado por um número sequencial. Desta forma, o primeiro objeto criado em uma

base de dados terá o número *um* como identificador lógico. Para transferir este conceito no GOA, foi construída uma camada que fica responsável pela criação do identificador lógico e pela associação deste ao endereço físico do GOA. Esta característica permite manter a mesma estrutura do GOA. As operações de manipulação de objetos que utilizam identificador físico não foram modificadas; apenas houve a criação de outras rotinas que fazem as mesmas funções de manipulação, porém com identificador lógico. Estas rotinas, na realidade, fazem a associação do identificador lógico com o nó e o endereço físico do objeto. Se o objeto estiver em outro nó, uma comunicação será criada solicitando o dado. Se o objeto não for remoto, a camada utilizará as funções normais do GOA, passando como parâmetro o endereço físico.

Para que haja a gerência entre o identificador lógico e o endereço de disco do objeto, foi criada uma tabela de identificadores que guarda a seguinte informação:

- identificador físico do objeto;
- volume ou nó onde o objeto está armazenado.

É importante notar que, como o identificador lógico é um número sequencial, não há a necessidade de procura para localizar a sua posição dentro da tabela. Este acesso é feito diretamente. O objeto com identificador de número 1 será representado na primeira posição da tabela.

Cada nó processador contém uma tabela de identificadores com a descrição completa de todos os objetos da base. Quando um objeto é transferido de um nó para outro, há atualização de todas as tabelas de usuários. Esta autlização é uma desvantagem em relação ao modelo com apenas uma tabela de identificadores. Além do mais, a replicação da tabela de identificadores implica desperdício de espaço em disco e existência de controles de integridade para garantir a igualdade de tabela entre as diversas tabelas do sistema.

V.4.2 ARQUITETURA DE MEMÓRIA DISTRIBUÍDA

O sistema apresenta a arquitetura de memória distribuída como o PARGOA-MD. Ele foi implementado em linguagem C e utilizou o pacote PVM (Parallel Virtual Machine). O sistema completo é caracterizado por cinco processos apresentados a seguir:

1. **Processo de Consultas**

Este processo é responsável por controlar a execução da consulta entre os processadores. Ele simula a interação do sistema com o usuário, recebendo uma solicitação de execução de consulta. Também analisa o predicado da consulta e, com base em um catálogo de distribuição, determina os processadores a serem utilizados no processamento da consulta.

2. **Gerente de distribuição**

É responsável pela captação das informações de acesso à base, assim como de desempenho. O gerente de distribuição também determina qual o melhor tipo de alocação da base para se conseguir um aumento de desempenho, executando a etapa de realocação. Este processo será explicado mais detalhadamente na seção V.4.2.1.

3. **O Executor de consultas**

Recebe do gerente e executa a consulta como se fosse localmente. Caso sejam necessários acessos a objetos remotos, ele fica responsável pela comunicação com os outros nós.

4. **O fornecedor de atributos e objetos**

Este processo é responsável por atender uma solicitação de dados vinda de um outro nó. Ele pode fornecer um objeto completo ou simplesmente uma cadeia de valores representando uma série de atributos. Este mecanismo de fornecimento de atributos diminui a troca de mensagens na rede, melhorando o desempenho.

5. **O Trocador de objetos**

É o responsável pela troca de objetos entre um nó e outro na etapa de realocação. Ele terá seu comportamento regulado pelo gerente de distribuição.

V.4.2.1 GERENTE DE DISTRIBUIÇÃO

O gerente de distribuição apresenta como principais funções: observar o sistema, realocar a base de dados e criar um novo catálogo de distribuição para o processo responsável pela execução das consultas, melhorando o desempenho. A seguir são listadas as principais funções deste módulo:

1. *Inicia todos os processos em todos os nós;*
2. *Observa o sistema recebendo as informações de acesso e desempenho;*

3. *Aplica uma estratégia de distribuição com base nas informações que recebeu;*
4. *Pára o sistema e executa a realoção da base de acordo com o resultado da estratégia executada;*
5. *Reativa o sistema possibilitando que os usuários possam solicitar novamente as consultas;*

A estratégia utilizada foi a descrita no capítulo 4. As informações de acesso são fornecidas pelo módulo de processo de consultas. Estas informações são:

1. *A assinatura da consulta;*
2. *O tempo de execução da consulta;*
3. *Os nós que participaram da consulta;*

Com estas informações, o gerente pode classificar as consultas de acordo com o seu tipo de predicado, como foi descrito no capítulo 4. A partir deste momento, após várias execuções, é aplicada a estratégia, formando uma Grid-File de objetos da coleção a ser fragmentada primariamente. Com a Grid-File pronta, é executado o algoritmo descrito no anexo I, criando o roteiro necessário para alocação da base de dados. Na implementação realizada a alocação é feita com o sistema parando de executar qualquer consulta. Nesta etapa, os processos de trocas de objetos passam a executar a alocação controlada pelo gerente de distribuição. A figura V.3 mostra este fenômeno.

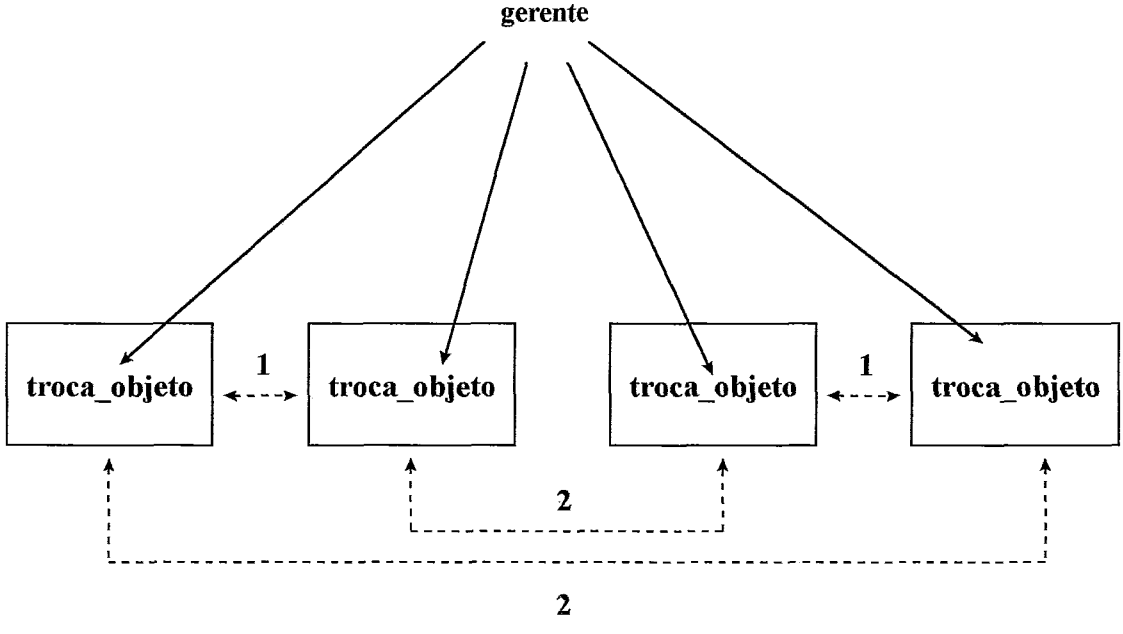


Figura V.3 - Alocação Off-Line

O gerente de distribuição inicia os processos de troca de objetos em cada nó. Após a inicialização, o gerente de distribuição ordena a troca de dados entre os nós em grupos de 2. Cada processo responsável pela transferência de dados recebe os identificadores dos objetos residentes em sua base local. Além dos identificadores, ele recebe o endereço do nó remoto para onde será transferido o nó. Após a transferência do primeiro nó, o gerente de distribuição envia novamente as mensagens para transferir os objetos do segundo para o primeiro nó.

Outra forma de realocação foi implementada (figura V.4). O gerente de distribuição armazena o número dos identificadores a serem transferidos, assim como os endereços dos nós destinos e a ordem das transferências. Após esta tarefa, o gerente de distribuição ordena os processos de trocas a acessarem esta base e executarem a troca de objetos. Esta alternativa de implementação oferece como vantagem a economia na passagem de mensagens com os processos de transferência de objetos. A desvantagem está na necessidade de existir uma área em disco compartilhada por todos os nós.

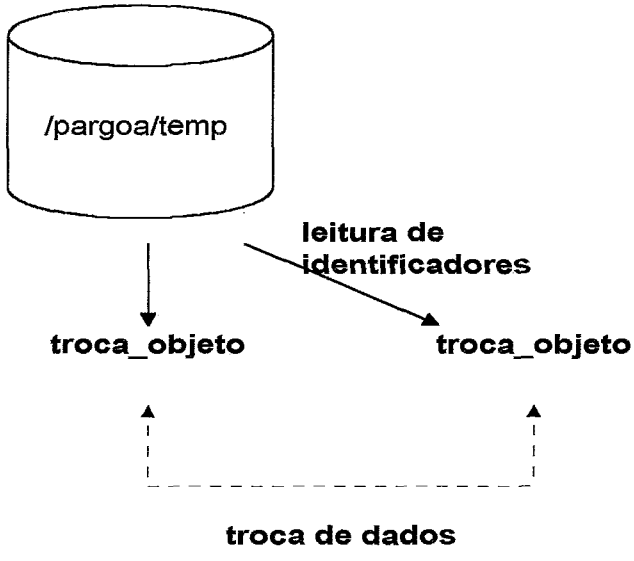


Figura V.4 – Alocação Off-line

Feita toda a transferência, o gerente de distribuição finaliza a execução dos processos de troca de objetos e reinicia os processos de execução de consultas e envio de atributos.

V.4.2.2 SIMULANDO A EXECUÇÃO DE CONSULTAS POR VÁRIOS USUÁRIOS

Para simular a solicitação de consultas por vários usuários, foi introduzido na arquitetura do sistema outro processo responsável por autorizar o envio de consultas,

além de receber seus resultados. A implementação deste processo é necessária, caso contrário, o processo de consultas ficaria parado, esperando pelo recebimento do resultado da consulta enviada.

Com a inclusão deste novo processo, a arquitetura do sistema fica representada na figura abaixo.

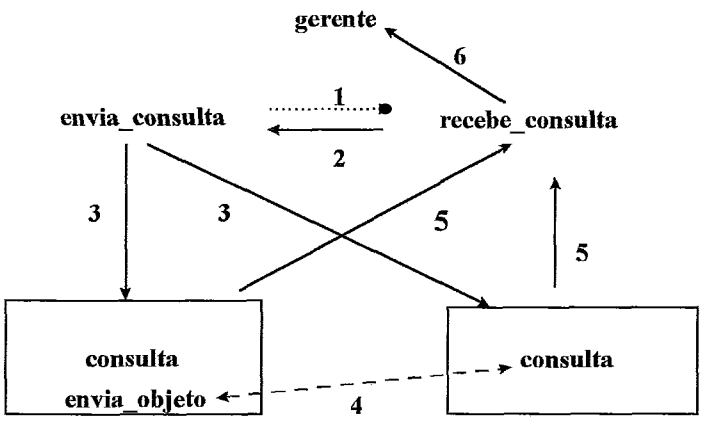


Figura V.4 - Arquitetura do Sistema

O gerente inicia todos os processos do sistema e fica aguardando. O processo de consultas pede autorização de envio de consulta para o mestre recebedor. Este autoriza o envio e marca o tempo de início de execução. O processo de consultas envia a consulta para os nós apropriados. Quando todos os nós enviarem o resultado da consulta, o processo recebedor marca o tempo final de execução de uma consulta. O processo recebedor também envia os tempos e as informações relativas às consultas para o nó gerente, que fará a realocação dos objetos, se necessário. O processo de recebimento de consultas só autoriza a execução de uma operação de seleção caso todos os processos envolvidos estejam sem atividade.

V.5 O AMBIENTE DE DESENVOLVIMENTO

O Gerente de Distribuição Dinâmica foi desenvolvido no laboratório da linha de Banco de Dados da COPPE. Na configuração da máquina paralela utilizou-se o pacote PVM versão 3.3, de domínio público. Os testes basearam-se numa fragmentação apresentada no trabalho de (MEYER, 1997), que foi baseado no benchmark OO7. Este foi desenvolvido na universidade de Wisconsin (CAREY *et al.*, 1994). A fragmentação mencionada acima é adotada neste trabalho como comparação para tipos de alocações diferentes. Além disso, este trabalho também promove outra fragmentação horizontal de

uma coleção utilizando dois atributos. As próximas subseções explicam mais detalhadamente o ambiente de desenvolvimento.

V.5.1 O AMBIENTE COMPUTACIONAL

O laboratório da COPPE/SISTEMAS é composto de 6 estações de trabalho com processadores POWER-PC, rodando sistema operacional AIX versão 4.1. Adicionalmente, foi utilizada uma outra estação de trabalho com sistema operacional AIX versão 3.1. A figura abaixo mostra a arquitetura das máquinas, apontando a capacidade de memória de cada estação. As seis estações mencionadas acima foram utilizadas para executar as consultas do teste de desempenho. A estação de trabalho adicional foi utilizada para executar os processos de gerente de distribuição, de envio de consultas, assim como o de recebimento de consultas. Todas as seis estações de trabalho contêm discos locais.

Uma área em disco, visualizada por todos os nós, foi utilizada pelo gerente de distribuição e pelos processos de troca de objetos na realocação da base de dados. O ambiente computacional utilizado é apresentado na figura abaixo:

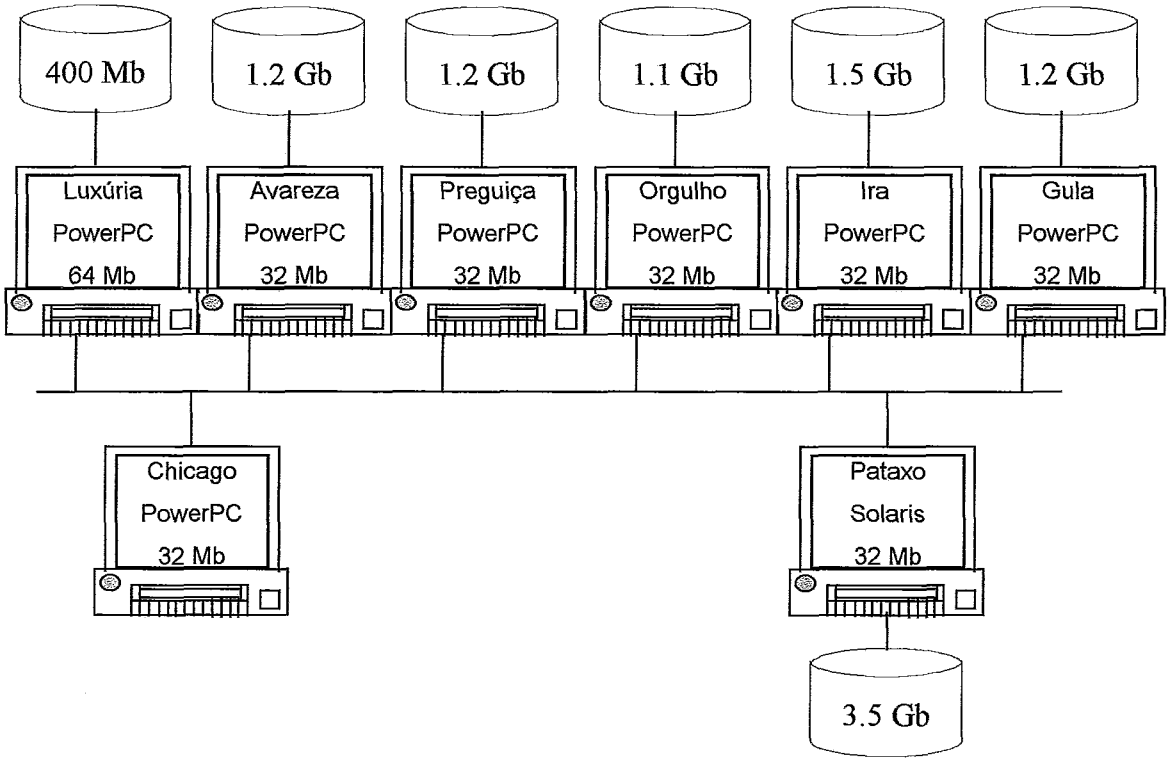


Figura V.5 – Estações de trabalho

V.5.2 PVM

O *Paralell Virtual Machine* é um pacote de bibliotecas que possibilita a configuração de uma máquina paralela através da comunicação entre estações de trabalho interligadas por uma rede de computadores. A primeira versão do PVM foi desenvolvida em 1989 no OAK *Ridge National Laboratory*. A versão 2.0 foi desenvolvida na universidade do Tennessee para ser utilizada em aplicações científicas. A partir da versão 3.0, o PVM tornou-se de domínio público, sendo disponibilizado na Internet, instalação, manual, guia de instalação, FAQs, etc. A versão utilizada neste trabalho foi a 3.3. Novas versões surgiram durante a implementação desta tese. Porém, resolveu-se por usar a versão do PVM já utilizada para manter o grau de comparação desejado com o trabalho de (MEYER, 1997).

O PVM possibilita a criação de aplicações paralelas, utilizando, como máquina paralela, o hardware disponível no laboratório. O PVM permite que um grupo de computadores com características diferentes seja visto como uma máquina paralela virtual. Para isso, o PVM controla de forma transparente a passagem de mensagens, conversão de dados e gerência de tarefas através da rede de computadores do laboratório.

As principais características do PVM foram retiradas de (GEIST *et al.*, 1994) e são listadas a seguir:

Computação baseada em processos: A unidade de paralelismo no PVM é uma tarefa, que pode ser um processo rodando ou um conjunto de instruções executadas seqüencialmente. O PVM permite que várias tarefas sejam executadas numa mesma máquina.

Transmissão de mensagens: O PVM permite, de forma fácil e transparente, através de funções implementadas, a passagem de mensagens entre as tarefas. Assim, o programador preocupa-se somente com o protocolo entre as diferentes tarefas existentes na aplicação. O tamanho da mensagem não recebe uma limitação inicial, sendo restringida pela quantidade de memória disponível em cada máquina.

Diferentes tipos de hardware: O PVM permite a interligação de diferentes tipos de computadores rodando diferentes tipos de sistemas operacionais. Para isso, uma versão do PVM deve ser instalada em cada máquina. O PVM resolve problemas de

conversão de dados, através da passagem de mensagens contendo diferentes tipos de dados definidos em sua biblioteca.

V.5.3 O BENCHMARK OO7

O Benchmark OO7 (CAREY, *et al.*, 1994) foi desenvolvido pela Universidade de Wisconsin e tem sido adotado como referência de teste de desempenho em Sistemas Gerenciadores de Banco de Dados Orientado a Objetos. Vários trabalhos já desenvolvidos na COPPE/SISTEMAS (MEYER, 1997) (LIMA, MATTOSO, 1996) já utilizaram o OO7 como teste de desempenho. O modelo adotado neste trabalho foi o mesmo adotado por (MEYER, 1997) e (LIMA, MATTOSO, 1996) e é apresentado na figura V.6.

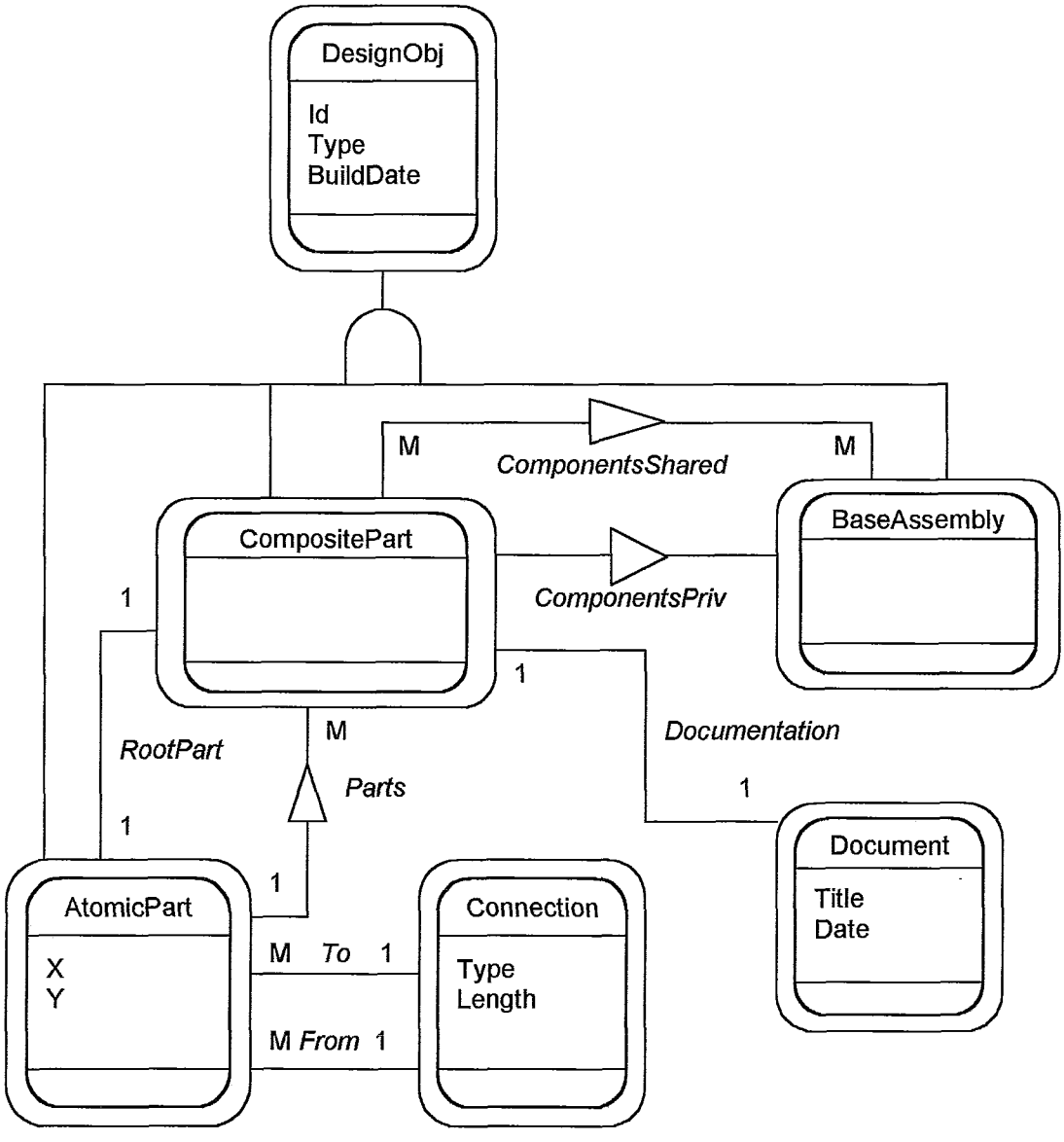


Figura V.6 – Modelo OO7 Adaptado

O Benchmark OO7 apresenta três tamanhos de base de dados: pequeno, médio e grande. Neste trabalho foi utilizada a base de dados de tamanho médio. A tabela abaixo mostra o número de objetos por classe.

	População		
Classe	**Pequeno**	**Médio**	**Grande**
BaseAssemblies	20	200	200
CompositeParts	500	500	500
Documents	500	500	500
AtomicParts	10000	100.000	100.000
Connections	30000	300.000	900.000

Segundo (CAREY *et al.*, 1994), a chave do modelo está na classe *CompositePart*. Cada *CompositePart* tem um número de atributos: *id* e *builDate* do tipo inteiro e um tipo alfanumérico representando o atributo tipo. Associado com a *CompositePart* está um objeto do tipo Documento (um manual por exemplo). É importante ressaltar que o tamanho de cada documento varia de acordo com o tamanho da base. Além disso, uma *CompositePart* é composta por objetos da classe *AtomicPart*. Na base média e na base grande, o número de *AtomicParts* por *CompositePart* é fixado em 200. Cada *AtomicaPart* se liga a outra *AtomicaPart* através do objeto *Connection*. O número de *Connections* dentro do benchmark OO7 pode variar entre 3, 6, e 9. Neste trabalho foi adotado o número de três *Connections* por *AtomicPart*. Um objeto da classe *BaseAssemblie* é formado por várias *CompositeParts* divididas em *CompsonestShared* e *ComponentsPriv*.

O trabalho desenvolvido por (MEYER, 1997) mostrou como é possível obter ganhos de desempenho utilizando uma fragmentação horizontal primária. Em várias consultas foram obtidos ganhos super lineares. O ganho é medido através da comparação entre as configurações com várias máquinas e os tempos de execução medidos em uma única máquina com uma única base de dados. A expressão abaixo define o ganho de desempenho medido nos testes realizados.

$$Ganho = {Tempo(1\,nó)}/{Tempo(N\,nós)}$$

$Ganho > N$ – *Ganho Superlinear*
$Ganho \cong N$ – *Ganho Linear*
$Ganho < N$ – *Ganho Sublinear*
$Ganho < 1$ – *Perda*

Se a relação Tempo 1 nó / Tempo N Nos for maior que N, obtem-se um ganho super-linear, sendo aproximadamente igual temos um ganho linear, uma medição perto de 1 caracteriza um ganho linear.

Esta tese teve como parâmetro de comparação a fragmentação adotada por (MEYER, 1997). Assim, foram construídas diversas variações desta fragmentação, buscando aumentar o número de fragmentos e distribuindo-os de forma circular entre os nós do sistema. Com isso, busca-se um ganho nas consultas através da verificação de uma percentagem menor da base de dados na execução da consulta. Após esta análise, foi feita uma outra fragmentação baseada em dois atributos. Os tempos médios das consultas realizadas como teste foram comparadas para cada fragmentação adotada. As próximas duas seções explicam com mais detalhes as fragmentações mencionadas acima.

V.5.3.1 FRAGMENTAÇÃO PRIMÁRIA DA CLASSE DOCUMENT

(MEYER, 1997) obteve resultados significativos de ganho de desempenho com uma fragmentação horizontal primária feita na Classe de *Document*. A coleção de *Documents* foi fragmentada por faixa de valores pelo atributo date, formando 6 fragmentos distribuídos de forma circular para cada nó do sistema. Cada base tem uma percentagem aproximadamente igual da base de dados.

Esta fragmentação tem como característica a não comunicação entre os nós do sistema durante a execução da consulta. Não existem objetos que referenciam objetos alocados em outro nó. Esta característica permite a execução total da consulta localmente, sem a necessidade de comunicações com nós remotos.

Esta tese recriou a fragmentação descrita acima. Sua identificação é (6 ca 100). A partir de consultas executadas nesta configurações o gerente de distribuição, segundo a estratégia EDMA, sugeriu uma nova alocação de objetos, chamada de (16 ca 50). Esta arrumação consiste em 16 fragmentos criados através de uma fragmentação por faixa de valores feita na classe *Document*. Estes 16 fragmentos foram resultantes da inserção dos 500 *Documents* dentro da Grid-File, cuja capacidade de cada Bucket de 48 objetos foi calculada pelo Gerente de distribuição. O sistema executou um conjunto de consultas

que tem como característica o aproveitamento do paralelismo do sistema. Isto é, a consulta será executada somente pelos nós responsáveis, deixando os outros livres para executarem outras consultas.

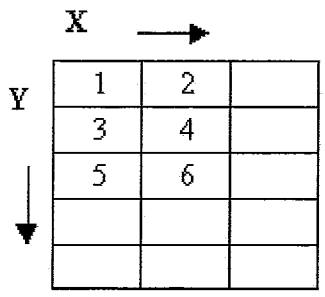
Outra arrumação foi criada, agora configurando-se manualmente a capacidade do bucket da Grid-File, garantindo que em cada máquina estejam 10 fragmentos diferentes. Esta configuração tem como objetivo testar se uma divisão da base em um número maior de fragmentos não possibilitaria um ganho de desempenho através da redução da atividade de procura dos dados. A capacidade do bucket foi configurada para 12 objetos e foram criados 61 fragmentos diferentes. A configuração chamada de (61 ca 10) representa esta realocação de objetos.

O gerente de distribuição realocou objetos e reconfigurou a máquina paralela virtual. Para cada fragmento, foi criado um processo de execução de consulta. Desta forma, na realocação proposta pelo gerente de distribuição, (16 ca 50), foram criados 16 processos de execução de consultas. Na realocação feita manualmente, (61 ca 10), foram criados 61 processos diferentes.

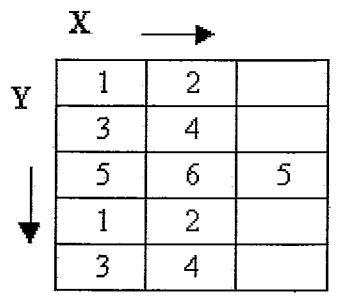
V.5.3.2 FRAGMENTAÇÃO BASEADA EM DOIS ATRIBUTOS

Utilizando o algoritmo descrito no capítulo IV, foi feita uma fragmentação horizontal da coleção de *Documents,* agora levando em consideração dois atributos. O primeiro atributo constituindo a primeira dimensão da Grid-File é o campo date. O segundo atributo foi retirado do relacionamento 1-1 que *Document* tem com *CompositePart,* pegando o atributo date. Desta forma, a Grid-File foi montada utilizando o atributo Date da classe *Document* e o atributo Date da Classe *CompositePart.* A política de espalhamento e capacidade do bucket também foram calculados pelo sistema. As figuras abaixo mostram como foi o processo de associação de cada fragmento a um nó do sistema.

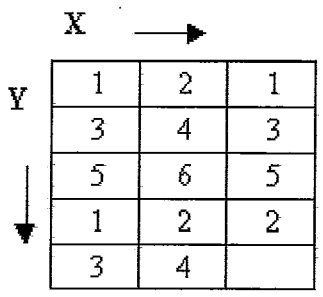
Dados ($N_1 = 3, N_2 = 5, M_1 = 15, M_2 = 15, P = 6, Capacidade\ Bucket = 139$)



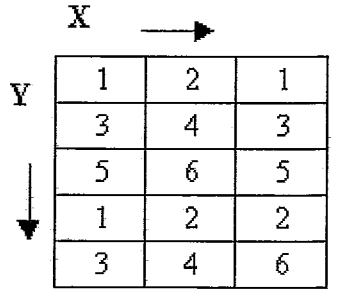
Passo 3 – Fase 1



Passo 3 – Fase 2



Passo 3 – Fase 3



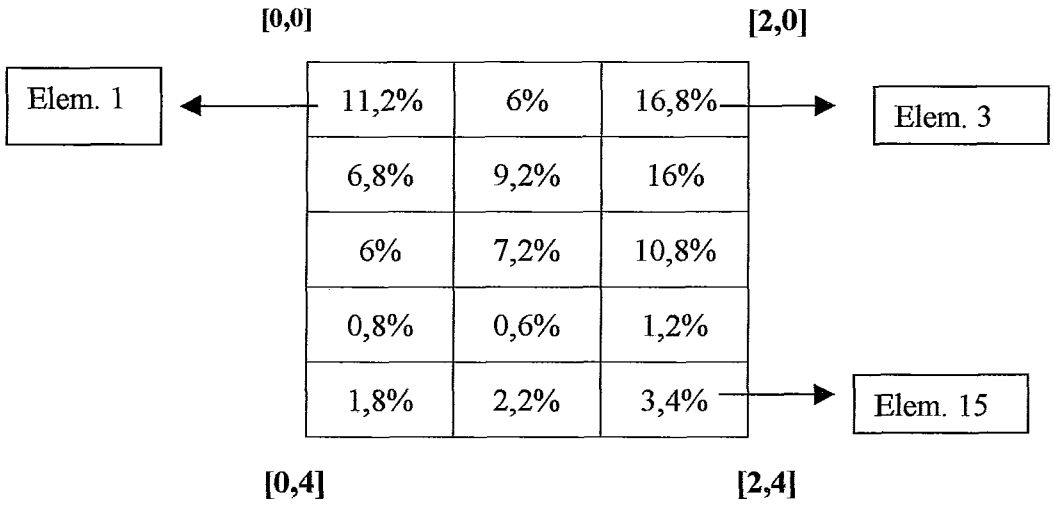
Passo 3 – Fase 4

Eixo dos X ⇒ Document.date

Eixo dos Y ⇒ Document->CompositePart.date

Figura V.7 – Associação Fragmentos-Nós

No final do processo de fragmentação, constatou-se que a Grid-File formada não ficou com uma distribuição uniforme, isto é, os elementos da Grid-File continham percentagens da base de dados muito diferentes entre si (vide figura V.8). Isto se deve ao fato de que um dos atributos escolhidos para a fragmentação não é um campo da classe. Ele é um campo de uma classe referenciada.



Nó	Elementos	Percentagem da Base de Dados
1	1, 3 e 10	28,8%
2	2, 11 e 12	7,8%
3	4,6 e 13	24,6%
4	5 e 14	11,4%
5	7 e 9	16,8%
6	8 e 15	10,6%

Figura V.8 – Percentagem de dados de cada elemento

Com isto, foi criado um novo algoritmo de associação dos fragmentos aos nós, para garantir uma melhor distribuição de carga dentro de cada máquina. O algoritmo é descrito abaixo em português estruturado:

```
Ordene em ordem decrescente todos os elementos da Grid-File
segundo a percentagem da base de dados;
Para todos os nos faça
      Qtd_Dados_No[i] = 0;
Para cada elemento não associado faça
      Ordene os No[i] segundo a sua percentagem de dados
      (Ordem decrescente)
      Para cada No[i] faca
            Se não Existe algum Nó cuja Qtd_Dados_No[i] é
            menor então
                  Associe o elemento para o No[i]
                  Qtd_Dados_No[i]    =    Qtd_Dados_No[i]    +
                  Percentagem da base do elemento associado;

Inicie A como sendo o conjunto de máquinas do laboratório
Para i = do nó mais carregado para o menos carregado faça
      O No[i] será representado pela máquina mais veloz
disponível;
      Retire a máquina selecionada de A
```

Figura V.9 – Algoritmo de distribuição de carga entre os nós

Com a execução deste algoritmo, foi feita uma nova realocação da base. A figura V.10 apresenta a nova distribuição da carga após a execução do algoritmo descrito acima.

Nó	Elementos	Percentagem da Base de Dados
1	5, 7 e 13	17%
2	2 e 9	16,8%
3	6 e 10	16,8%
4	3	16,8%
5	1, 3, 6 e 15	16,4%
6	4, 8 e 14	16,2%

Figura V.10 – Percentagem de dados dos elementos após algoritmo

A tabela abaixo mostra as realocações executadas e que participaram da análise de desempenho mostrada no capítulo VI.

Id.	Tipo fragmentação	Tipo Conf.	# de Fragmentos	# Processos de exec. de cons.	# de processos/Nó
6 (ca 100)	Frag. Horiz. Prim. (1 atrib)	Manual	6	6	1
16 (ca 50)	Frag. Horiz. Prim. (1 atrib) Grid-File	**Sug. pelo Sistema**	16	16	2,66
61 (ca 10)	Frag. Horiz. Prim. (1 atrib) Grid-File	Manual	61	61	10,1
Frag15NU (*)	Frag. Horiz. Prim. (2 atrib) Grid-File	**Sug. pelo Sistema**	15	15	2,5
Frag15U (**)	Frag. Horiz. Prim. (2 atrib) Grid-File	**Sug. pelo Sistema**	15	15	2,5

(*) $\Rightarrow$ base criada com associação fragmentos-nós não uniforme.

(**) $\Rightarrow$ base criada com associação fragmentos-nós feita pelo algoritmo de distribuição descrito na figura V.10.

V.5.4 DESCRIÇÃO DOS TESTES DE DESEMPENHO

Esta subseção apresenta as consultas que foram executadas e tiveram os tempos analisados na próxima seção.

V.5.4.1 Vazão

Para medir a *vazão ou o throughput* do sistema foram criadas três cargas de consultas. A primeira e a segunda utilizam consultas que contêm em seu predicado o atributo date da classe document, variando somente na quantidade de consultas

executadas. A terceira carga de consultas contém consultas que utilizam o atributo date da classe Document ou atributo date da classe CompositePart. A tabela abaixo mostra os tipos de consultas utilizadas nas três cargas de consulta.

Descrição da Consulta	Cargas participantes	Qtd de consultas exec. / Carga		
		C 1	C 2	C 3
Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data [OPER] [VALOR]	C 1, C 2, C 3	50	200	24
Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data [OPER] [VALOR]	C3			24

Os operadores e os valores das consultas foram escolhidos de forma aleatória. Nas cargas 1 e 2 foram utilizados os operadores: =, >, <. Na carga 3 foram utilizados os operadores >, < somente. As cargas de consultas foram armazenadas em arquivos. Estes arquivos são lidos pelo processo de envio de consultas na inicialização do sistema. Com todas as consultas lidas, o processo de envio de consultas pode simular a chamada de consultas, como descrito na seção V.4.2.2.

V.4.2 Descrição de Consultas

Esta tese também procurou medir o desempenho das várias alocações descritas em relação à execução de diferentes tipos de consultas. As consultas utilizadas no teste de desempenho foram as mesmas utilizadas no trabalho de (MEYER, 1997), adicionando-se mais duas consultas que utilizam o atributo *date* da classe *CompositePart*. A descrição das consultas, assim como suas principais características, são mostradas abaixo:

Consulta	Descrição
Consulta 1	Obtenha as AtomicParts que tenham data < 1011
Consulta 2	Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data < 1200
Consulta 3	Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data < 1010
Consulta 4	Obtenha as CompositeParts que tenham uma RootPart com data < 1011
Consulta 5	Obtenha as Connections cuja AtomicPart To tenha data < 1011
Consulta 6	Obtenha as Connections com lenght < 10
Consulta 7	Obtenha os Documents das CompositeParts com data > 1600
Consulta 8	Obtenha as CompositeParts cujos Documents tenham data > 1660
Consulta 9	Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data > 1000
Consulta 10	Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data > 100

Principais Características:

Consultas 1 e 6: predicado simples e coleção alvo com grande população.

Consultas 2 e 3: predicado com navegação percorrendo 3 coleções. A diferença entre elas se dá na cláusula condicional.

Consulta 4: predicado com navegação percorrendo duas coleções, sendo que a população da coleção alvo é pequena, enquanto que a da coleção fim é grande.

Consulta 5: predicado com navegação percorrendo duas coleções, onde tanto a coleção alvo como a coleção fim têm grande população de objetos.

Consultas 7 e 8: predicado com navegação percorrendo duas coleções, onde as populações das coleções alvo e fim são pequenas.

Consultas 9 e 10: predicado com navegação percorrendo 2 coleções, sendo a população da coleção alvo grande, enquanto a população da coleção fim é pequena.

Capítulo VI

ANÁLISE DE DESEMPENHO

VI.1 INTRODUÇÃO

O objetivo deste capítulo é medir os tempos dos testes descritos no final do capítulo V em relação às várias estratégias de alocação adotadas pelo sistema de distribuição construído.

O tempo aferido pelo sistema foi o de "relógio de parede" (CROWL, 1994), isto é, o tempo que o usuário ou aplicação espera pelo resultado da consulta. Este tipo de medição inclui tanto o tempo de processamento da consulta, quanto o tempo de E/S, comunicação na rede, bloqueios, etc.

Procurou-se medir os tempos de execuções de consultas separadamente. Para isso, cada consulta era executada separadamente dentro do sistema. O sistema media o tempo de execução da primeira consulta, chamada execução a Frio, e o tempo médio das próximas 20 execuções, execução a quente. Na execução a frio, o sistema deve ser iniciado, isto é, o buffer de páginas do sistema deve estar limpo. Na execução a quente, foi observado um aumento de desempenho devido à utilização do cache de páginas.

Também procurou-se medir o grau de paralelismo entre consultas do sistema em cada tipo de alocação. Para isso foi medido a vazão, isto é, o número de consultas executadas por unidade de tempo adotada. A vazão medida nesta tese é representada em Consultas/Segundo. A medição foi feita calculando o tempo de resposta total do sistema, isto é, a diferença entre tempo final de execução da última consulta e o tempo inicial da primeira consulta.

Foram utilizadas seqüências especiais de consultas neste experimento. As seqüências contêm consultas onde os predicados adotados fazem referência aos atributos utilizados nas fragmentações das bases.

Segundo (MEYER, 1997), a análise dos resultados explorando paralelismo em aplicações de banco de dados é complexa devido ao grande número de variáveis que influem no resultado, entre as quais estão:

- *Comunicação na rede;*
- *Utilização do buffer do sistema;*

- *Comunicação de Entrada e saída dos processadores com as unidades de disco;*
- *Agrupamentos da base no disco;*
- *Algoritmos para paralelização;*
- *Fragmentação das classes;*
- *Balanceamento de carga entre os nós.*

Na análise não foi possível separar cada variável listada acima. Em uma das fragmentações feitas pelo sistema de distribuição foi constatada uma desproporção da carga de dados associada a cada nó. As fragmentações geradas por fragmentação horizontal primária baseada em um atributo não geraram diferenças significativas no balanceamento de cargas. Todos os fragmentos tinham quase a mesma proporção da base de dados. Como os fragmentos são alocados de forma circular entre os processadores, o balanceamento de carga não foi afetado. No caso da fragmentação baseada em dois atributos, foi constatada uma desproporção entre as máquinas do sistema. Desta forma, uma nova realocação foi feita de maneira a garantir um bom balanceamento de carga. Estas duas alocações fizeram parte da análise de desempenho.

O buffer de páginas do sistema também foi abordado na análise de desempenho. O tamanho do buffer foi ajustado para ter o melhor desempenho possível. Neste processo, ficou constatado que para tamanhos maiores, o sistema apresenta queda no desempenho devido ao aumento de "swaps" feitos no disco durante a alocação de memória. Desta forma, a base dividida em 6 fragmentos, (6 ca 100) teve seu cache configurado em um buffer de 100 páginas. O cache do GOA consiste em um buffer onde as páginas mais utilizadas são guardadas na memória. O GOA mantém uma tabela de envelhecimento, fazendo a política de reposição das páginas dentro do buffer.

Um página do GOA tem 512 inteiros longos. Nas máquinas PowerPC cada inteiro longo representa 4 bytes, logo, cada página tem 2Kbytes de tamanho.

Os tamanhos do cache do sistema para as outras configurações foram também configurados, procurando manter uma proporcionalidade com a base de 6 fragmentos. Somente a configuração (6 ca 100) teve seu tamanho de cache configurado para garantir o melhor desempenho. Nas outras configurações o buffer de páginas foi configurado de maneira proporcinal ao número de processos por máquina.

Na alocação feita com 61 fragmentos foi utilizado um buffer de 10 páginas para cada processo. Nas configurações com 16 e 15 fragmentos foram utilizados um buffer de 50 páginas. A tabela abaixo apresenta os tamanhos para o buffer do sistema de acordo com cada configuração:

Configuração	Número de Páginas	Total do tamanho do Buffer
6(ca 100)	100	200 Kb
16(ca 50)	50	100 Kb
61(ca 10)	10	20 Kb
Frag15NU	50	100 Kb
Frag15U	50	100 Kb

Tabela VI.1 – Tamanho do Buffer

A seção 2 deste capítulo aborda a medição da *vazão* do sistema em seqüências de consultas que usam atributos utilizados nas fragmentações executadas pelo sistema de distribuição. A seção 3 mostra os tempos de vários tipos de consultas executadas seqüencialmente a frio e a quente. Finalmente, a seção 4 faz comentários finais a respeito dos resultados encontrados nas duas seções anteriores.

VI.2 VAZÃO DO SISTEMA

Seqüências de consultas foram criadas com o intuito de medir o aproveitamento do paralelismo em cada configuração diferente de alocação. As duas primeiras seqüências examinadas apresentam consultas do tipo 2, que foi apresentada no capítulo V. Esta consulta tem como característica a navegação entre classes e a utilização do atributo *date* da classe *Document*. Os tipos de operadores variaram aleatoriamente, assim como o valor da condição de seleção. A seguir é descrita a consulta utilizada nas seqüências 1 e 2, e são apresentados os resultados de vazão apurados pelo sistema em consultas/segundo.

Consultas utilizadas na seqüência 1 e 2:

> *Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data [OPER] [VALOR]*

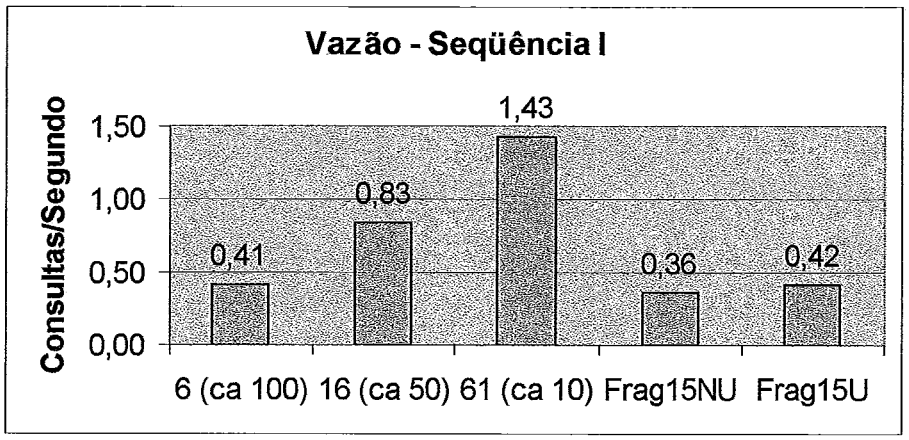


Figura VI.1 – Vazão da Seqüência I

61 (ca 10) levou vantagem

Na seqüência I foram executadas 50 consultas. A configuração que levou vantagem neste teste foi a formada por 61 fragmentos. Segundo (DEWITT, GRAY, 1990), a medida que o tamanho do fragmento no disco diminui, o tempo gasto por uma consulta para procura nos dados é reduzido de forma significativa. Como o número de fragmentos em 61(ca 10) é muito maior que nas outras configurações, ela apresenta um tempo de resposta mais rápido por fragmento. 16(ca 50) também apresentou uma *vazão* maior do que as outras três configurações.

Além disso, esta configuração leva vantagem na autorização de execução de consulta. Uma consulta só é liberada para executar quando nenhum processo designado para executá-la está ocupado. Com 61 processos, a probabilidade de ter um processo livre para execução aumenta em relação às outras configurações.

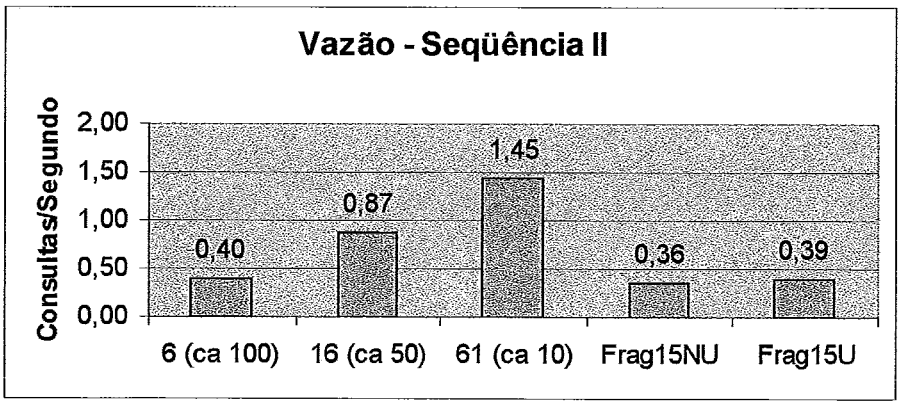


Figura VI.2 – Vazão da Seqüência II

O aumento do número de consultas não alterou os resultados

Na seqüência II são executadas 200 consultas. No gráfico acima é possível perceber a tendência de ganho de desempenho das configurações 16(ca 50) e 61(ca 10). Estas aumentaram a *vazão* em relação à seqüência I. Em contrapartida, as outras configurações sofreram uma pequena perda.

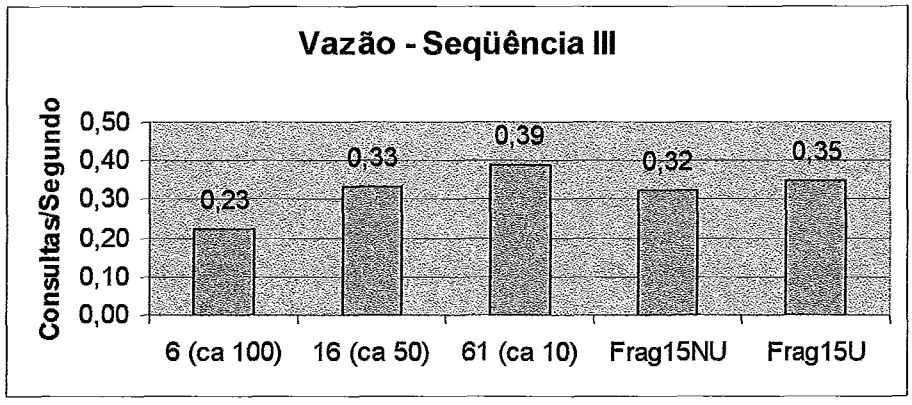


Figura VI.3 – Vazão da Seqüência III

Com pequenos fragmentos 61 (ca 10) leva vantagem seguida de perto por Frag15U

A seqüência III tem uma nova característica: aparece em sua formação consultas que utilizam o atributo *date* da classe *CompositePart*. Metade das consultas foram compostas com o tipo utilizado nas seqüências I e II. A outra metade é formada pela consulta descrita abaixo:

Consulta que participa na seqüência III:

> *Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data [OPER] [VALOR]*

Desta forma, as configurações FragNU15 e Frag15U se beneficiam, mas por terem número de fragmentos menor, a configuração 61(ca 10) continua a ter uma pequena vantagem. Além disso, os operadores participantes da consulta adicionada na seqüência III foram ">" e "<". Quando o sistema executar uma consulta com operador de igualdade na fragmentação 61(ca 10), envolverá apenas um processo, enquanto que as configurações bidimensionais envolverão mais processos. Logo, a existência de operadores de igualdade na seqüência III pode ter aumentado a *vazão* para a configuração 61(ca 10).

VI.3 CONSULTAS EXECUTADAS EM SEQÜÊNCIA

Os tempos de resposta das execuções a frio e a quente da análise são fornecidos na tabela abaixo:

	1 Nó		6(ca 100)		16 (ca 50)		61 (ca 10)		Frag15NU		Farg15U	
	F	Q	F	Q	F	Q	F	Q	F	Q	F	Q
Cons. 1	77,46	52,43	4,83	3,36	4,10	2,80	3,45	2,25	5,10	3,41	3,56	2,39
Cons. 2	110,04	73,65	6,99	5,11	2,29	1,71	1,54	1,15	2,91	2,06	3,61	2,68
Cons. 3	102,59	72,72	5,39	4,07	1,63	1,16	0,37	0,29	2,04	1,93	3,24	2,40
Cons. 4	0,93	0,88	0,19	0,13	0,27	0,18	0,62	0,35	0,16	0,12	0,21	0,18
Cons. 5	258,23	178,03	16,89	9,57	14,50	8,41	17,80	11,11	24,88	11,15	13,73	7,4
Cons. 6	141,80	86,97	10,82	5,92	9,37	4,98	10,22	4,40	13,39	6,39	8,67	4,44
Cons. 7	-	0,502	-	0,127	-	0,092	-	0,580	-	0,045	-	0,046
Cons. 8	-	0,488	-	0,95	-	0,055	-	0,140	-	0,056	-	0,061
Cons. 9	96,57	67,48	6,01	4,35	5,18	3,72	4,26	3,09	4,33	3,07	2,77	1,97
Cons. 10	96,66	66,56	6,27	4,60	5,54	3,98	4,46	3,29	7,25	5,09	4,84	3,52

VI.1.3.1 Consultas com predicado simples

Consulta 1:

Obtenha as AtomicParts que tenham data < 1011.

Consulta 6:

Obtenha as Connections com tamanho < 10.

Nas consultas 1 e 6, todos os processos participam da execução da consulta. As duas consultas trabalham com coleções de populações grandes, 100.000 objetos para AtomicParts e 300.000 para Connections.

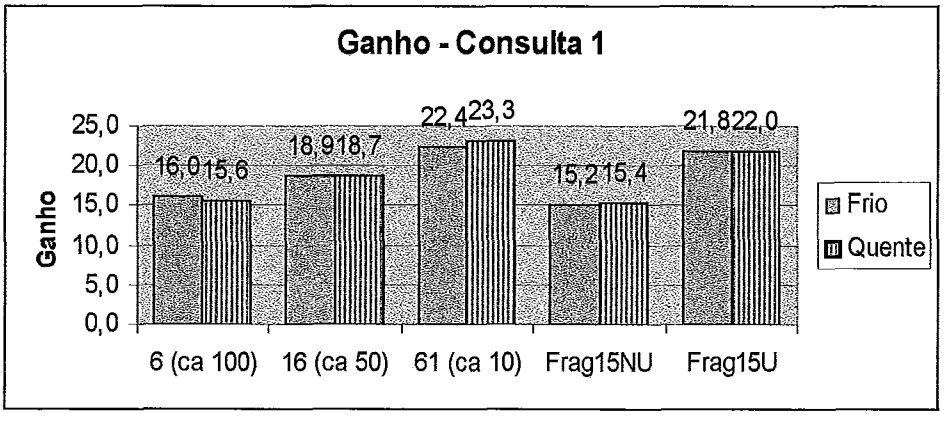


Figura VI.4 – Consulta 1

61 (ca 10) e Frag15U tiveram melhor desempenho

A figura VI.4 mostra uma pequena vantagem de desempenho para a configuração formada por 61 fragmentos, seguida de perto pela fragmentação feita com dois atributos e distribuição uniforme de carga entre os nós.

A configuração Frag15NU foi a única que ficou abaixo de 6(ca 100), mas mantendo o ganho super-linear, tanto na execução a frio quanto na execução a quente.

A figura VI.5 mostra o ganho super-linear obtido em todas as configurações. A mesma tendência é mantida com Frag15NU, tendo um desempenho pouco inferior às demais. Nas execuções a frio, Frag15U obteve melhor desempenho.

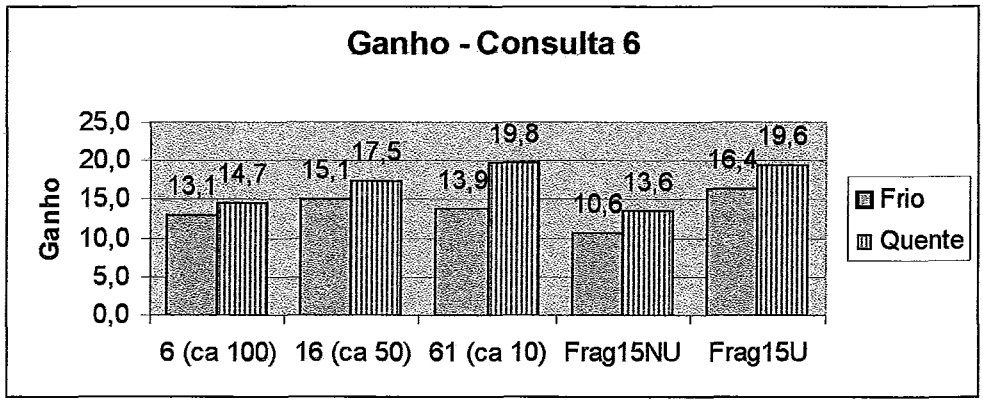


Figura VI.5 – Consulta 6

Frag15U e 61 (ca 10) tiveram melhor desempenho

VI.1.3.2 Consultas com predicado de navegação percorrendo três classes

Consulta 2:

Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data < 1200.

Consulta 3:

Obtenha as AtomicParts que façam parte de uma CompositePart cujo Document tenha data < 1010.

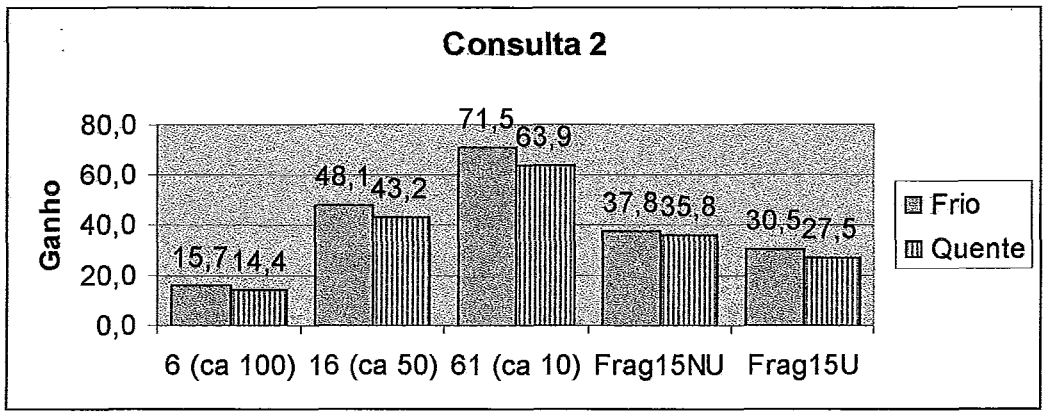


Figura VI.6 – Consulta 2

Pequenos fragmentos acarretam ganho na procura dos dados

As consultas 2 e 3 são praticamente iguais, sendo diferenciadas somente pela cláusula condicional. Outra diferença está no número de objetos retornados pela consulta, 21000 objetos para consulta 2 e 800 objetos para a consulta 3.

As duas consultas têm na cláusula condicional o atributo *Document.date* utilizado em todas as fragmentações testadas. Com tamanho de fragmentos reduzidos, a configuração de 61 fragmentos mostrou um desempenho muito maior que as demais configurações. Este desempenho é maior na consulta 3, onde existem menos objetos selecionados pela consulta.

O número de processos executantes e número de nós que participaram das consultas 2 e 3 são mostrados na tabela abaixo:

Configuração	Consulta 2		Consulta 3	
	No de Processos Executantes	No de Nós participantes	No de Processos Executantes	No de Nós participantes
6(ca 100)	2	2	1	1
16(ca 50)	3	3	1	1
61(ca 10)	13	6	1	1
Frag15NU	5	3	5	3
Frag15U	5	4	5	4

Figura VI.7 – Número de processo por consulta

É importante notar que em todas as configurações feitas com fragmentação unidimesional, a consulta 3 teve sua execução direcionada para 1 nó. Em Frag15U, há a participação de mais nós nas consultas do que em Frag15NU. Porém, Frag15NU

consegue um desempenho superior ao Frag15U. A diferença de carga entre o nó mais carregado de Frag15NU e de Frag15U é pequena, cerca de 3% do total dos objetos procurados nas duas consultas.

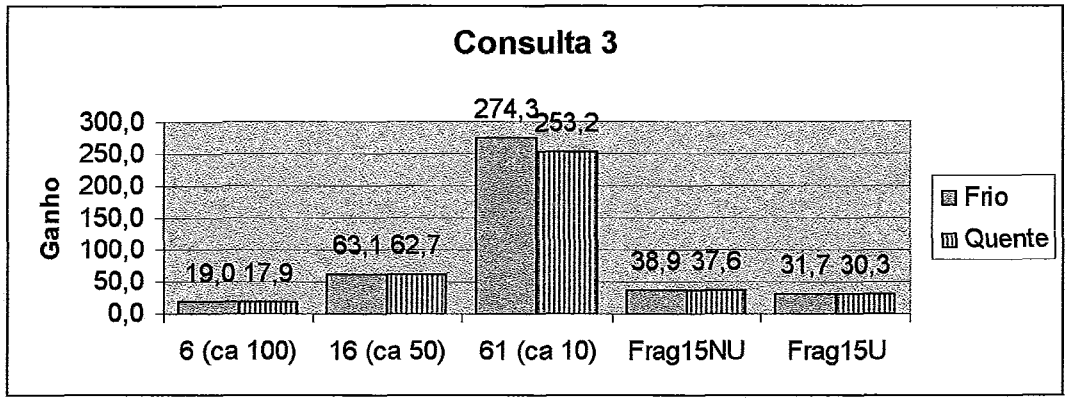


Figura VI.8 – Consulta 3

Com pequena número de objetos selecionados 61 (ca 10) se destaca

VI.1.3.3 Consultas com predicado de navegação percorrendo duas classes

Consulta 4:

Obtenha as CompositeParts que tenham uma RootPart com data < 1011.

Consulta 5:

Obtenha as Connections cuja AtomicPart To tenha data < 1011.

Consulta 7:

Obtenha os Documents das CompositeParts com data > 1600.

Consulta 8:

Obtenha as CompositeParts cujos Documents tenham data > 1660

Consulta 9:

Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data > 1000

Consulta 10:

Obtenha as AtomicParts que façam parte de uma CompositePart que tenha data > 100

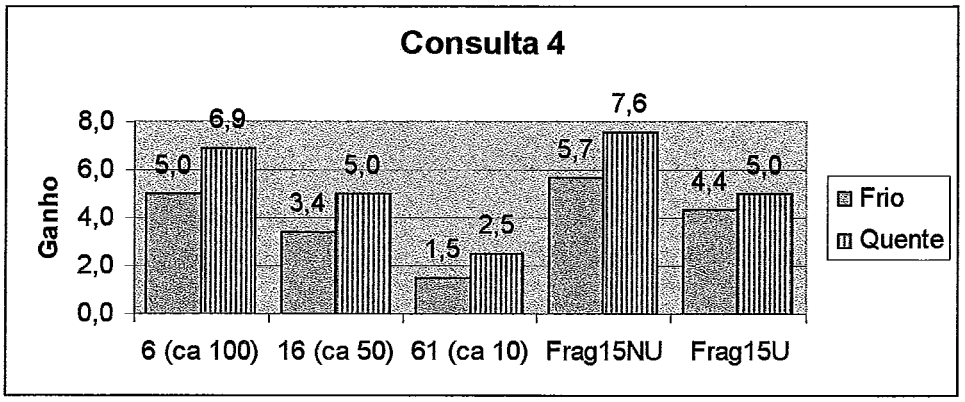


Figura VI.9 – Consulta 4

Frag15NU foi a única configuração que conseguiu ganho super-linear

Na consulta 4 quase todas as configurações exceto a Frag15NU tiveram desempenho inferior à fragmentação adotada por (MEYER, 1997). Nesta consulta, não houve um ganho super-linear em nenhuma das execuções a frio. Na execução a quente pode-se observar um ganho na configuração Frag15NU. A configuração formada por 61 fragmentos, que até então tinha os melhores resultados, ficou muito abaixo das outras configurações, obtendo um ganho sublinear. Na consulta 4, todos os processos participam da execução.

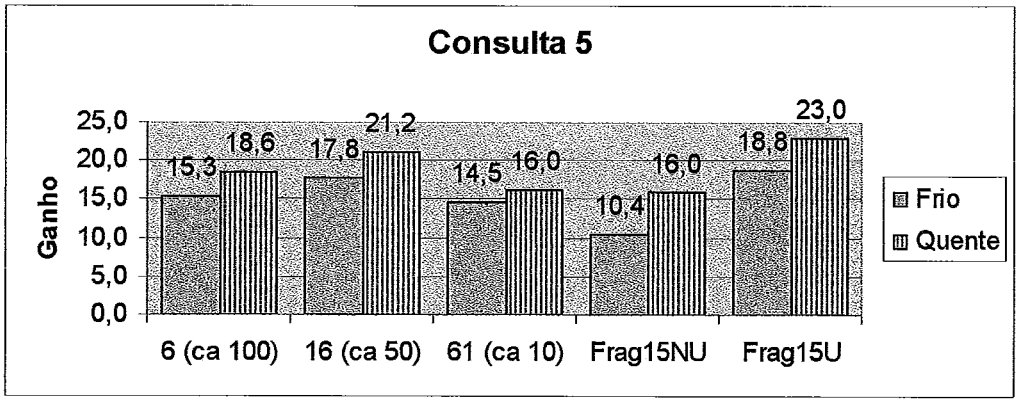


Figura VI.10 – Consulta 5

Frag15U obtém o melhor desempenho devido ao bom balanceamento de carga

Nesta consulta, também há a participação de todos os processos do sistema. Esta consulta trabalha com duas coleções de grande população. Nesta medição, os piores desempenhos foram das configurações 61(ca 10) e Frag15NU. O melhor desempenho foi obervado em Frag15U, um pouco acima de 16(ca 50). Todas as configurações obtiveram ganho super-linear.

Assim com em (MEYER, 1997), os tempos medidos nas execuções a frio das consultas 7 e 8 contiveram variações muito grandes obtendo um coeficiente de variação acima de 20%. Por isso, foi descartado este dado nas análises das duas consultas.

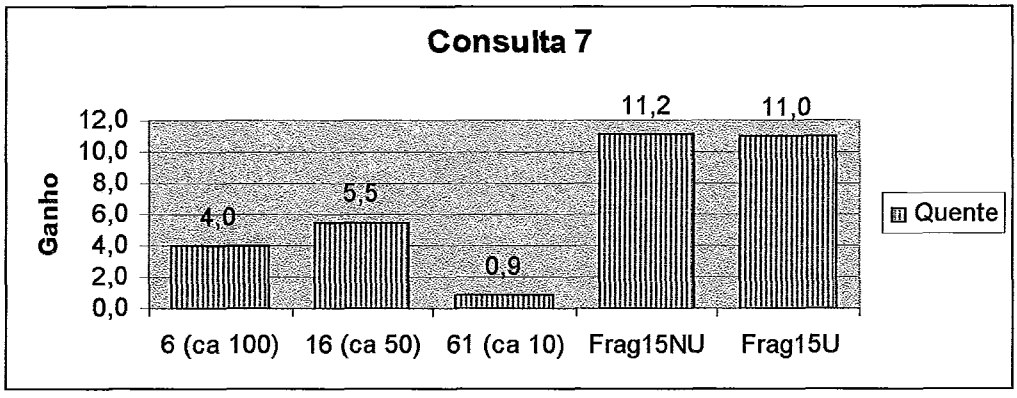


Figura VI.11 – Consulta 7

As configuração baseadas em dois atributos foram superiores

Na consulta 7, houve utilização do atributo *CompositePart.date*, beneficiando diretamente as duas últimas configurações da figura VI.11. Enquanto as configurações 6(ca 100), 16(ca 50), 61(ca 10) enviaram a consulta para ser executada por todos os processos, as configurações Frag15NU e Frag15U enviaram a consulta para execução em 6 processos. O número de nós executantes foram 5 e 4 nós respectivamente.

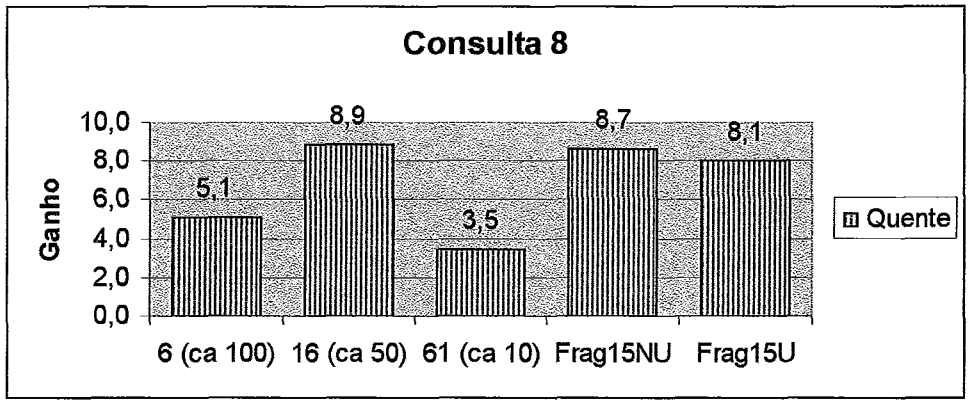


Figura VI.12 – Consulta 11

6 (ca 10) e 61 (ca 10) apresentaram o pior desempenho

Nesta consulta, tanto a coleção alvo como a coleção fim têm populações pequenas. Nesta corrida houve um desempenho inferior na configuração 61(ca 10). As configurações 16(ca 50), Frag15Nu e Frag15U apresentaram ganho super-linear acima da apresentada por 6(ca 100), com ganho linear. Em (MEYER, 1997), os testes

atingiram um ganho linear para 6 nós, mesmo com o sistema tirando proveito de direcionar a consulta para os nós apropriados. O fato de (MEYER, 1997) ter utilizado discos remotos não prejudicou o resultado nesta medição.

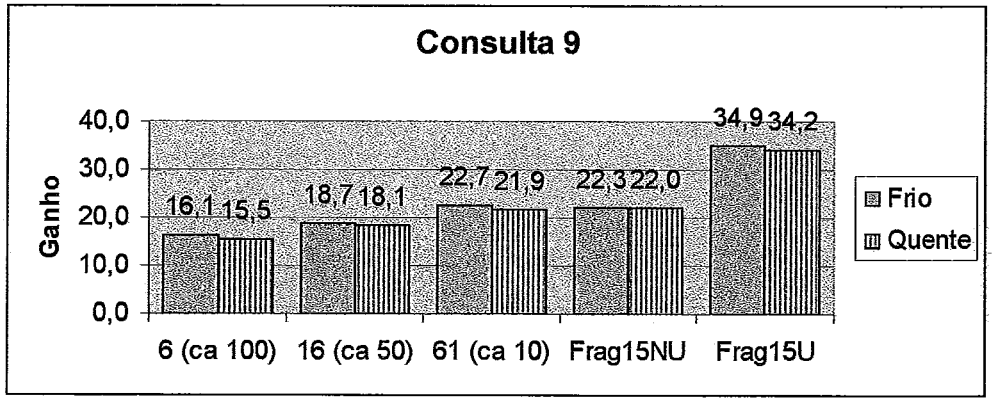


Figura VI.13 – Consulta 9

Frag15U destaca-se em relação Frag15NU

As consultas 9 e 10 apresentam população grande na coleção alvo e população pequena na coleção fim. A consulta 9 destaca o ganho obtido em Frag15U. As outras medições também apresentaram ganhos super-lineares, mas a Frag15U se beneficiou por que pôde direcionar a consulta para os nós apropriados. Frag15NU também continha esta vantagem. Porém, seu desempenho foi inferior a Frag15U devido ao desbalanceamento de carga entre os nós . Frag15U utiliza os 6 nós do sistema na execução da consulta 9, enquanto Frag15NU utiliza somente 5.

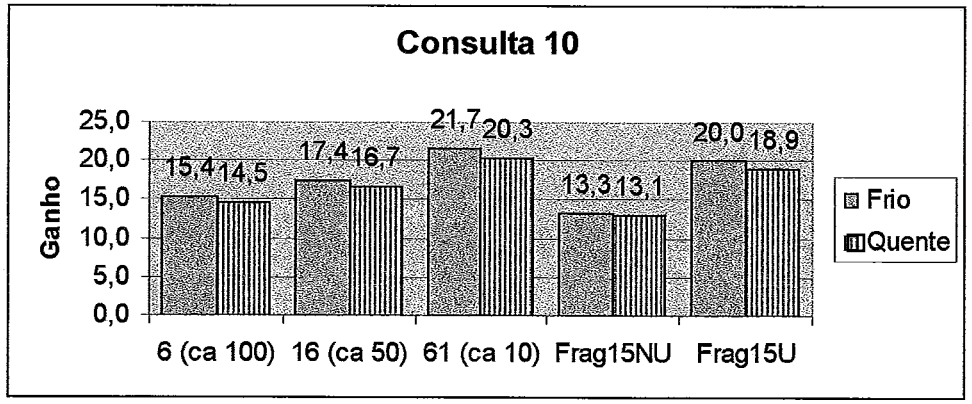


Figura VI.14 – Consulta 10

Todos os processos participam da consulta. 61 (ca 10) e Frag15U são superiores

Na consulta 10, todos os processos participam da sua execução. Frag15NU obteve ganho inferior a 6(ca 100). Todas as configurações obtiveram ganho super-linear devido à redução na procura dos dados. Os resultados se equivaleram por causa da

proporcionalidade do tamanho do buffer em cada configuração. Na consulta 10 uma grande quantidade de objetos é selecionada (94000 objetos). Na consulta 9 este número é reduzido 10000 objetos.

VI.4 Considerações Finais

Em relação a vazão, a melhor estratégia foi a fragmentação feita manualmente, forçando um número de fragmentos grande. Porém, a política de liberação de execução da consulta beneficiou muito este tipo de desempenho. Na seqüência III, observa-se um aumento de desempenho em relação às fragmentações bidimensionais.

Das várias configurações testadas, as que mais obtiveram ganho em relação à fragmentação de 6 nós com cache de 100 páginas (6 ca 100) foram Frag15U e 16(ca 50). Estas duas configurações conseguiram desempenho superior em relação à 6(ca 10) em 9 das 10 consultas apresentadas, não obtendo êxito somente na consulta 4.

Além disso, separando os dados da configuração 6(ca 100) e fazendo uma comparação com o trabalho desenvolvido por (MEYER, 1997), foi possível observar um ganho significativo devido ao uso de disco local nos testes atuais. Nas execuções a quente das consultas 4 e 8 os ganhos foram lineares segundo o trabalho anterior. Com discos locais, estes resultados passaram para ganhos super-lineares. Ao mesmo tempo, ganhos super-lineares na execução a frio das consultas 1, 6, 2 e 3 foram conseguidos neste trabalho. As tabelas abaixo mostram quais configuração tiveram melhor desempenho nas execuções a frio e a quente por consulta:

Configuração	Consultas									
	C1		C2		C3		C4		C5	
	F	Q	F	Q	F	Q	F	Q	F	Q
6 (ca 100)	4°	4°	5°	5°	5°	5°	2°	2°	3°	3°
6 (ca 50)	3°	3°	2°	2°	2°	2°	4°	4°	2°	2°
61 (ca 10)	1°	1°	1°	1°	1°	1°	5°	5°	4°	5°
Frag15NU	5°	5°	3°	3°	3°	3°	1°	1°	5°	4°
Frag15U	2°	2°	4°	4°	4°	4°	3°	3°	1°	1°

Tabela VI.2 – Posição de desempenho das diversas configurações

Consultas de 1 a 5

Configuração	Consultas									
	C6		C7		C8		C9		C10	
	F	Q	F	Q	F	Q	F	Q	F	Q
6 (ca 100)	5°	4°	-	4°	-	4°	5°	5°	4°	5°
6 (ca 50)	2°	3°	-	3°	-	1°	4°	4°	3°	3°
61 (ca 10)	3°	1°	-	5°	-	5°	2°	3°	1°	1°
Frag15NU	4°	5°	-	1°	-	2°	3°	2°	5°	4°
Frag15U	1°	2°	-	2°	-	3°	1°	1°	2°	2°

Tabela VI.3 – Posição de desempenho das diversas configurações

Consultas de 6 a 10

Nas tabelas é possível observar que a configuração manual (61 ca 10) apareceu com o melhor desempenho em várias consultas. Porém, nas consultas onde esta configuração não aparece bem colocada, há perda de desempenho. Em alguns casos, há até perda de desempenho em relação à consulta sendo executada por um nó (Consulta 7), em outros o ganho é apenas sublinear (Consulta 8 e Consulta 4).

A seguir é apresentada outra tabela com as descrições das consultas em relação aos seguintes fatores: Comunicação, E/S, Estratégia com melhor desempenho, estratégia de pior desempenho. As configurações que não apresentaram ganho super-linear estão em negrito.

Consulta	Tipo	Com.	E/S	Frio		Quente	
				Melhor	Pior	Melhor	Pior
C1	Ext.	Média	Média	61(ca 10) (SupLin)	Frag15NU (SupLin)	61(ca 10) (SupLin)	Frag15NU (SupLin)
C2, 3	Nav.	Baixa	Alta	61(ca 10) (SupLin)	6(ca 100) (SupLin)	61(ca 10) (SupLin)	6(ca 100) (SupLin)
C4	Nav.	Média	Baixa	FragNU15 (SupLin)	**61(ca 100) (SubLin)**	FragNU15 (SupLin)	**61(ca 100) (SubLin)**
C5	Nav.	Média	Alta	Frag15U (SupLin)	Frag15NU (SupLin)	Frag15U (SupLin)	61(ca 100) (SupLin)
C6	Ext.	Média	Alta	Frag15U (SupLin)	Frag15NU (SupLin)	Frag15U (SupLin)	Frag15NU (SupLin)
C7	Nav.	Média	Baixa	-	-	Frag15NU (SupLin)	**61(ca 100) (Perda)**
C8	Nav.	Baixa	Baixa	-	-	16 (ca 50) (SupLin)	**61(ca 100) (SubLin)**
C9	Nav.	Baixa	Média	Frag15U (SupLin)	6 (ca 10) (SupLin)	Frag15U (Sup Lin)	6 (ca 100) (SupLin)
C10	Nav.	Baixa	Média	61 (ca 10) (SupLin)	Frag15NU (SupLin)	61 (ca 10) (SupLin)	6 (ca 100) (SupLin)

Tabela VI.4 – Desempenho das estratégias por consulta

Capítulo VII

CONCLUSÕES

VII.1 CONTRIBUIÇÕES

O ganho de desempenho conseqüente da aplicação de modelos de distribuição em base de dados é relatado em vários trabalhos da literatura. Alguns deles (BAIÃO, 1997) (GHANDEHARIZADEH, 1990) (EZEIFE, BARKER, 1995), ressaltam a importância do conhecimento de informações de acesso à base para implementação das estratégias de fragmentação. Porém, nem sempre o projetista tem o grau de certeza apropriado em relação a sua base de dados para alimentar estas estratégias.

Esta tese desenvolveu uma ferramenta de distribuição que determina novas configurações de distribuição da base visando aumentar o desempenho. Nesta ferramenta foi implementado um módulo capaz de absorver várias informações de acesso à base para auxiliar o projetista de banco de dados no modelo de distribuição. Adicionalmente, foi implementado no sistema um módulo de fragmentação horizontal. Este módulo incorporou a estratégia definida em (GHANDEHARIZADEH, 1990). Modificações foram implementadas para adaptar a estratégia ao modelo Orientado a Objetos. Além disso, as informações de desempenho foram captadas pelo gerente de distribuição, já no trabalho de (GHANDEHARIZADEH, 1990) estes dados foram estimados por um simulador.

O módulo de informações e o módulo de integração foram integrados dando inteligência ao sistema. O sistema com base em informações de acesso sugere uma fragmentação horizontal da base, baseado em 1 ou 2 atributos de uma coleção de objetos. A análise de desempenho da fragmentação baseada em dois atributos é uma contribuição importante desta tese, pois o trabalho onde foi descrita a estratégia utilizada para esta fragmentação (GHANDEHARIZADEH, 1990) não realizou testes de desempenho de configurações baseadas em dois atributos, permanecendo apenas no campo unidimensional.

Várias configurações foram testadas dentro do ambiente computacional composto de uma rede com 6 estações de trabalho POWERPC. A capacidade de

paralelismo entre consultas foi relatada medindo-se *o Throughput ou a Vazão do sistema*. As várias alocações foram testadas em várias situações. Para isso cada configuração teve seu desempenho medido em relação às várias consultas diferentes.

Foram criadas 5 configurações diferentes, (16 ca 50), (Frag15U) (Frag15NU) foram sugeridas pelo sistema de distribuição com base em informações de acesso. A configuração (6 ca 100) foi realizada conforme descrito no trabalho de (MEYER, 1997). A configuração (61 ca 10) foi configurada manualmente.

Os testes realizados mostraram que as estratégias sugeridas pelo sistema conseguiram ganho super-linear na maioria das situações medidas, não provocando queda de rendimento em relação a fragmentação estática desenvolvida por (MEYER, 1997). Adicionalmente, esta tese mostrou que a divisão da base em muitos fragmentos pode gerar um ganho de desempenho muito grande. A configuração (61 ca 10) mostrou ganhos muito acima das outras configurações nas consultas 2 e 3. Porém, este tipo de configuração apresentou perda de desempenho na consulta 7 além de ganhos sublineares nas consulta 4 e 8.

Fazendo um sistema de pontuação simples, variando de 5 a 1 pontos para 1° e 5° lugares respectivamente, e utilizando a tabela de classificação apresentada na seção VI.4, é possível calcular o ranking das configurações testadas em relação a todas as consultas executadas. As tabelas a seguir mostram esta classificação.

Classificação Execução a Quente	Configuração	Pontos
1°	Frag15U	36
2°	16 (ca 50)	33
3°	61 (ca 10)	32
4°	Frag15NU	30
5°	6 (ca 100)	19

Tabela VII.1 Classificação das configurações – Execução a Quente

Classificação Execução a Frio	Configuração	Pontos
1°	Frag15U	30
2°	61 (ca 10)	30
3°	16 (ca 50)	26
4°	Frag15NU	19
5°	6 (ca 100)	15

Tabela VII.2 Classificação das configurações – Execução a Frio

Observa-se que nas execuções a frio 61 (ca 10) obteve melhor classificação que 16(ca 50). A estratégia adaptada na implementação deste trabalho incorpora outros valores além do tempo médio de processamento, entrada e saída e comunicação. Com a utilização dos tempos médios de repostas das consultas executadas o tempo que o sistema operacional está realizando "swaps" também foi incorporado. Desta forma, não é possível afirmar que 16 fragmentos (configuração 16ca50) é um número ótimo.

Além disso, o número de fragmentos calculados pelo sistema deve variar em relação às configurações das máquinas que compõem a máquina paralela. A capacidade de multi-tarefa do sistema operacional é outro fator que modifica o resultado da estratégia, se aplicada em outro ambiente computacional.

61 (ca 10) obteve ganhos expressivos nas consultas 2 e 3. Se todas as transações de uma aplicação são compostas na sua maioria por consultas semelhantes, o gerente de distribuição pode sugerir uma super fragmentação dos dados, aumentando o desempenho. Uma facilidade possível de ser implementada é o projetista configurar a capacidade do bucket da Grid-File e em seguida fazer testes de desempenho utilizando uma seqüência de consultas.

Esta tese também abordou o assunto da alocação de objetos em arquitetura de memória distribuída, apontando aspectos e comparando a alocação de objetos em outras arquiteturas paralelas. Outra contribuição importante foi a constatação de que identificadores lógicos de objetos simplificaram a fase de realocação da base de dados. A fase de realocação com o sistema parado foi especificada e totalmente desenvolvida por este trabalho.

VII.2 TRABALHOS FUTUROS

Nos trabalhos de paralelismo desenvolvidos pela linha de Banco de Dados que utilizaram o Gerente de Objetos Armazenados (TAVARES *et al.*, 1996), (MEYER, 1997), no qual este se inclui, não houve uma padronização de implementação. Este trabalho utilizou o mesmo modelo de memória usado em (MEYER, 1997). Porém, o sistema construído nesta tese não partiu da mesma implementação do PARGOA-MD.

Durante o desenvolvimento deste trabalho, uma nova versão do GOA foi desenvolvida, colocando novas funcionalidades e reformulando o modelo de gerência de objetos persistentes. Esta versão, chamada de GOA++ (MAURO, 1997), não é apenas uma biblioteca com funções de gerência de objetos; já incorpora características de um servidor de objetos. Com esta versão, é possível partir para uma padronização dos sistemas paralelos baseados no GOA. Nesta padronização, poderiam ser incorporados os módulos de informação e alocação automática implementados neste trabalho.

Além disso, o GOA++ possui uma ferramenta gráfica que permite visualizar todo o esquema da base de dados e manipular objetos residentes no servidor (MAURO, MATTOSO, 1998). Nesta ferramenta, poderia ser incluído um assistente de distribuição, auxiliando o projetista no modelo de distribuição da base de dados.

Este trabalho observa, ainda, que a transferência de objetos durante a execução do sistema é um assunto em aberto. A replicação da base de dados e a utilização de outras técnicas de fragmentação podem ser incluídas dentro do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

BAIAO, F. 1997, *Uma Estratégia para o Projeto de Distribuição de Bases de Dados Orientados a Objetos* COPPE/UFRJ, Rio de Janeiro, Brasil.

BAIAO, F., MATTOSO, M. 1998, "A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases" *Special Issue of the Journal of Computing and Information* -, Winnipeg, Canadá, junho 1998, pp.141-148.

BAIAO, F., MATTOSO, M. 1996, "Questões em Projeto de Distribuição de Bases de Dados Orientadas a Objetos", In: *WorkShop de Orientação a Objetos,* COPPE/UFRJ, Rio de Janeiro, Brasil.

BARBOSA, V., 1990, "Strategies for the Prevention of Communication Deadlocks in Dsitributed Databases Parallel Programs", *IEEE Transactions on Software Engineering,* v. 16, n. 11, pp. 1311-1316.

BASSILIADES, N., VLAHAVAS, I. 1995, "A Non Uniform Data Fragmentation Strategy for Parallel Main-Memory Database Systems", *Proc.of th 21st VLDB Conference,* pp. 370-381.

BATINI, C., LENZERINI, M., NAVATHE, S. 1986, "A Comparative Analisys of Methodologies for Database Schema Integration", *ACM Computing Surveys,* v. 18, n. 4, pp. 323-364.

BERNSTEIN, P.A., GOODMAN, N. 1981, "Concurrency Control in Distributed Database Systems", *ACM Computing Surveys,* v. 2, n. 13, pp. 185-222.

BORAL, H., ALEXANDER, W., CLAY, L. *et al.*, 1990, "Prototyping Bubba, a High Parallel Database System", *IEEE Transactions on Knowledge and Data Engineering,* v. 2, n. 1, pp. 2-24.

CAREY, M.J., DEWITT, D.J., FRANKLIN, M.J. *et al.*, 1994, Shoring Up Persistent Applications, *Proceedings of 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis MN.*

CAREY, M.J., DEWITT, D.J., NAUGHTON, J.F. 1994, The OO7 Benchmark, *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, n.2, Washington DC, pp. 12-21*

CATTEL, R.G.G. 1994, "Future Database Systems", In: CATTEL, R.G.G. (ED), *Object-Oriented and Extended Relational Database Systems,* cap. 6, Addison-Wesley.

CERI, S., PERNICI, B., WIEDERHOLD, G. 1987, "Distributed Database Design Methodologies", *Proc IEEE,* v. 75, n. 5, pp. 533-546.

CROWL, L., 1994, "How to Measure, Present and Compare Parallel Performance", IEEE *Paralell and Distributed Technology,* vol2, n.1, pp. 9-25.

DEWITT, D.J., GERBER, R. 1985, "Multiprocessor Join Algorithms", In: *Int Conf on VLDB,* Stokholm, Setembro.

DEWITT, D.J., GRAY, J.N. 1990, "The Parallel Database Systems: The future of Database Processing or a Passing Fad ?", In: *SIGMOD RECORD,* v. 19, pp. 104-112, No 4.

DEWITT, D.J., GHANDEHARIZADEH, S., SCHNEIDER, A.D. 1988, "A Performance Analisys of the Gamma Database Machine", In: *SIGMOD CONFERENCE,* No. 1, pp. 350-360.

ELMAGARMID, A.K., SOUNDARARAJAN, N., LIU, M.T. 1988, "A Distributed Deadlock Detection and Resolution Algorithm and Its Correctness Proof", *IEEE Transactions on Software Engineering,* v. 14, n. 10, pp. 1443-1452.

EZEIFE, C.I., BARKER, K. 1995, "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", *Distributed and Parallel Databases,* v. 3, n. 3, pp. 247-272.

GEIST, A., BEGUELIN, A., DONGARRA, J., *et al.*, 1994, PVM 3 User's Guide And Reference Manual, ORNL/TM-12187 ed. Oak Ridge National Laboratory, Tennesse, EUA.

GENESERETH, M.R., KETCHPEL, S.P. 1994, "Software Agents", *Communications of the ACM*, v. 37, n. 7, pp. 48-53.

GHANDEHARIZADEH, S. 1990, *Physical Database Design in Multiprocessor Database Systems* Univerisity of Wisconsin, Madison, EUA.

GHANDEHARIZADEH, S. , DEWITT, D.J. 1989, A Multiuser Performence Analisys of Alternatives Declustering Strategies, Tech. Report. 855, Univerisity of Wisconsin, Madison. EUA.

GHANDEHARIZADEH, S., DEWITT, D.J. 1994, "A Performance Analisys of Alternative Multi-Attribute Declustering Strategies", *ACM SIGMOD*, pp. 29-38.

GHANDEHARIZADEH, S., WILHITE, D., LIN, K. *et al.*, 1994, "Object Placement in Parallel Object-Oriented Database Systems", *Proc IEEE*, pp. 253-262.

GRAEFE, G. 1993, "Query Evaluation Techniques for Large Databases", *ACM Computing Surveys*, v. 25, n. 2, pp. 73-170.

GRAY, J.N. 1979, "Notes on Data Base Operating Systems", In: BAYER, R., GRAHAM, R.M., SEEGMÜLLER, G. (EDS), *Operating Systems: An Advanced Course*, New York, Springer-Verlag.

HAMMER, M., NIAMIR, B. 1979, "A Heuristic Approach to Attribute Partioning", In: *ACM SIGMOD Int.Conf on Management of Data*, pp. 93-101, Boston, Maio.

HOFFER, H.A., SEVERANCE, D.G. 1975, "The Use of Cluster Analisys in Phisical Database Design", In: *First .Int.conf.on Very Large Data Bases*, pp. 69-86, Framingham, Setembro.

HSIAO, D.K. , ÖZSU, M.T. 1981, A Survey of Concurrency Control Mechanisms for Centralizad Databases, Technical Report 81-1, Columbus, Ohio,

HUANG, Y., WOLFSON, O. 1994, "Object Allocation in Distributed Database and Mobile Computers", In: *10th.Int.Conf.Data Engineering*, No.1, pp. 20-29.

KARLAPALEM, K., NAVATHE, S., MORSI, M. 1994, "Issues in Distribution Design of Object Oriented Databases", In: *Distributed Object Management,* Morgan Kaufman Publishers, pp. 148-164.

KARYPIS, G. , KUMAR, V. 1996a, A Fast and High Quality Multilevel Scheme for Particioning Irregular Graphs, Tech Report 95-035, University of Minnesota , Departament of Computer Science, Minneapolis, EUA, February.

KARYPIS, G. , KUMAR, V. 1996b, Analisys of Multilevel Graph Partioning, Tech. Report 95-037, University of Minnesota , Departament of Computer Science, Minneapolis, EUA, February.

KIM, W., CHOI, I., GALA, S., *et al.*, 1995, "On Resolving Schematic Heterogeity in Multidatabase Systems", In: KIM, W. (ED), *Modern Database Systems,* cap. 26, New York, ACM Press.

KNAPP, E. 1987, "Deadlock Detection in Distributed Databases", *ACM Computing Surveys,* v. 4, n. 19, pp. 303-328.

LIMA, F., MATTOSO, M. 1995, "Performance Study of Distributed O2 Databases using 007 BenchMark", In: *Anais do Simposio Brasileiro de Banco de Dados,* pp. 117-129, Recife, Outubro.

LIMA, F., MATTOSO, M. 1996, "Performance Evaluation of Distribution in OODBMS: a Case Study with O2", *Anais da IX International Conference on Parallel & Distributed Computing Systems,* ISCA/IEEE, Dijon, França, setembro 1996, pp.720-726.

LIU, D. , SHASHI, S. 1996, Partioning Similarity Graphs: A Framework for Declustering Problems, Tech-Report, University of Minnesota , Departament of Computer Science, Minneapolis, EUA.

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

MAURO, R.C., MATTOSO, M. 1998, "GOA++ e suas ferramentas", In: *Ia.Mostra Brasileira de Software Acadêmico e Comercial do XIII Simpósio Brasileiro de Banco de Dados,* pp. 83-88, Maringá, outubro.

MAURO, R.C. 1997, "GOA++: Tecnologia, implementação e extensões aos serviços de gerência de objetos", In: *XII Simpósio Brasileiro de Banco de Dados,* pp. 272-277, Fortaleza, outubro.

MAIER, D., GRAEFE, G., SAHPIRO, L. D., DANIELS, S., KELLER, T., VANCE, B., "Issues in Distributed Object Assembly", *IWDOM,* 1992: 165-181.

MEYER, L.V. 1997, *Paralelismo em SGBDOO com memória distribuída: Uma implementação no PARGOA,* Tese de M. Sc., COPPE, Rio de Janeiro, Brasil.

MOLINA, H., HSU, M. 1995, "Distributed Databases", In: KIM, W. (ED), *Modern Database Systems,* cap. 23, New York, Addison-Wesley.

NAVATHE, S., WIEDERHOLD, G., DOU, J. 1984, "Vertical Partioning of Algotithms for Database Design", *ACM Transactions Database Systems,* v. 4, n. 9, pp. 680-710.

NIEVERGELT, J., HINTERBERGER, H. 1984, "The Grid File: An Adaptable, Symmetric Multikey File Structure", *ACM Transactions Database Systems,* v. 9, n. 1, pp. 38-71.

NYANCHAMA, G.M., OSBORN, S.L. 1994, "Database Security Issues in Distributed Object-Oriented Databases", In: *Distributed Object Management,* Morgan Kaufman, pp.92-97.

OMIECINSKI, E. 1995, "Parallel Realtional Databases Systems", In: KIM, W. (ED), *Modern Database Systems,* cap. 24, New York, ACM Press.

ÖZSU, M.T., DAYAL, U., VALDURIEZ, P. 1994, "An Introduction to Distributed Database Management", In: *Distributed Object Management,* Morgan Kaufmann, pp.1-24.

ÖZSU, M.T., VALDURIEZ, P. 1991a, Principles of Distributed Database Systems, U.S.A, Prentice Hall.

ÖZSU, M.T., VALDURIEZ, P. 1991b, "Distributed Database Systems: Where are we now?", *Computer,* v. 24, n. 8, pp. 68-78.

PIRES, P.F., MATTOSO, M. 1996, "Aspectos de Interoperabilidade na Arquitetura HIMPAR", In: *Anais do XI Simpósio Brasileiro de Banco de Dados,* pp. 43-57, Sao Carlos, Outubro.

PITOURA, E., BUKHERS, O., ELMAGARMID, A.K. 1995, "Object Orientation in Multitabase Systems", *ACM Computing Surveys,* v. 27, n. 2, pp. 141-195.

SACCA, D., WIEDERHOLD, G. 1985, "Database Partioning in a Cluster of Processors", *ACM Transactions Database Systems,* v. 4, n. 10, pp. 473-498.

SCHNEIDER, A.D., DEWITT, D.J. 1989, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Enviroment", *ACM SIGMOD Int.Conf on Management of Data,* pp. 110-121.

SCHNEIDER, A.D., DEWITT, D.J. 1990, "Tradeoffs in processing Complex Join Queries via Hasing in Multiprocessor Database Machines", In: *Proc.of the Sixteenth VLDB Conference,* pp. 469-480.

SOLEY, R., KENT, W. 1995, "The OMG Object Model", In: KIM, W. (ED), *Modern Database Systems,* cap. 2, New York, ACM Press.

SOUZA, J. M. 1986, "Software Tools for Conceptual Schema Integration" *School of Information Systems, Univesity of East Anglia.*

STONEBRAKER, M., KATZ, M., PATTERSON, D. *et al.*, 1988, "The Design of XPRS", *Proc.of the Forteenth VLDB Conference,* pp. 318-330.

TAVARES, F.O., SOARES, J., MATTOSO, M. 1996, PARGOA-V: Paralelismo no Servidor de Objetos Persistentes GOA, Relatório Técnico ES-387/96, Universidade Federal do Rio de Janeiro, COPPE/PESC, Brasil.

TSANGARIS, M., NAUGHTON, J.F. 1991, "A Stochastic Approach for Clustering in Object Bases", In: *SIGMOD CONFERENCE,* pp. 12-21.

UCHOA, M.E., SECIM, E.O. 1995, "Interoperabilidade de Objetos em Sistemas de Banco de Dados Heterogêneos", In: *II WorkShop em Banco de Dados Nao Convencionais,* pp. 8-10, Niteroi, Dezembro.

VALDURIEZ, P. 1986, "Optimization of Complex Database Queries Using Join Indices", *Database Engineering Bulletin,* 9(4), pp.10-16.

VALDURIEZ, P., 1993, "Parallel Database Systems: Open Problems and New Issues", *Journal on Dsitributed and Parallel Databases,* v. 1, n. 2, pp. 1-27.

WOLF, J.L., DIAS, D.M., YU, P.S. 1990, "An Effective Algorithm for Parallelizing sort-Merge Join in the Presence of Data Skew", In: *Proc.of the Second International Symposium on Database in Parallel and Distributed Systems,* pp. 103-115.

ANEXO I

Algoritmo de Associação de Nós aos Elementos da Grid-File

O algoritmo a seguir representa uma Heurística criada por (GHANDEHARIZADEH, 1990) com o objetivo de associar os nós existentes do sistema a cada fragmento criado pela inclusão de objetos da coleção numa Grid-File.

A associação é feita considerando-se dois cenários:

O número de fragmentos |F| é menor ou igual ao número de nós P;

O número de fragmentos é maior que o número de nós P;

Caso 1: |F| <= P

Neste caso cada fragmento é associado a um nó diferente.

Caso 2: |F| > P

Se a Grid-File contém somente uma dimensão, isto é, ela foi construída observando-se somente um atributo; os fragmentos são associados aos nós de maneira circular. Para cada fragmento será criado um processo que ficará responsável pela execução das consultas daquele fragmento.

Se a Grid-File contém mais de uma dimensão a associação é feita seguindo 4 Passos. Os quatro passos são baseados em condições de contornos. As próximas seções descreverão os passos do algoritmo assim como as condições de contorno. Antes porém, será apresentado uma descrição breve das variáveis utilizadas pelo algoritmo:

Descrição:

M_i o número de nós ótimos para a dimensão i (como foi explicado no capítulo 4);

P o número total de processadores;

K número de dimensões

N_i é o número de fatias que uma dimensão contém. Fatias são as partes que são divididas uma Grid-File quando se fixa uma dimensão. Numa Grid-File de dimensão 2x2 as fatias são representadas pelas colunas ou pelas linhas dependendo do eixo de referência.

Passo 1

Condições de contorno:

1. $\prod_{i=1}^{k} M_i = P$

2. N_i é múltiplo de M_i

3. valor de M_i é o mesmo para cada dimensão D_i $(M_i = p^{\frac{i}{K}})$

Dividindo a Grid original em M_i grupos de $\frac{N_i}{M_i}$ fatias, a estrutura original é dividida em P grids menores de dimensão KxK. A associação é feita atribuindo para cada SubGrid formada um nó diferente.

Passo 2:

Condições de contorno:

É removida a terceira condição do passo 1. Com esta condição relaxada novos valores para M_i são calculados. M_i é substituído por $\frac{\prod_{j=1}^{K} M_j}{M_i}$. A associação segue-se como no passo 1.

Exemplo:

Grid-File de 12x8;

Sistema composto de 8 nós;

N_1=12, N_2= 8, P=8;

M_1= 4 e M_2= 2

Os valores de M_i são diferentes. O algoritimo substitui os valores de M_1 para 2 e M_2 para 4. Com estres novos valores, usando o passo 1 a Grid-File resultando com os nós associados é apresentada na figura abaixo.

1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8

Figura A.1 - Grid-File Passo 2

Passo 3:

É removida a Segunda condição do passo 1.

O passo 3 é constituído de 4 fases. A fase 1 consiste em associar nós aos elementos de uma Grid-File que preenche as condições do passo 1. A fases 2, 3 e 4 associam os elementos restantes, ainda não atingidos pela primeira fase. As fases 2 e 3 analisam os elementos restantes associando nós procurando obter M_i diferentes nós para cada elemento. A fase 4 é somente um complemento e preenche os espaços vazios deixados pelas duas últimas fases.

As 4 fases são apresentadas a seguir em forma de português estruturado.

Fase 1

```
/* Para cada dimensão i calcule NovaNi para ser multiplo de M1
(Passo 2)
for i = 1 to K
{
NewN[i] = N[i] - (N[i] MOD M[i])
}
Usando o Passo 1 associe as Sub-Grids criadas a um processador
diferente.
```

Fase 2

Elementos_Restantes = $\prod_{i=1}^{K} N[i] - \prod_{i=1}^{K} NewN[i]$

For i = 1 to P

QuotaDe_P[i] = $\left\lceil \frac{Elementos\_Restantes}{P} \right\rceil$

For i = a dimesão mais frenquentemente asseçada para a menos acessada

{

Elementos = Número de elementos não associados na fatia da dimensão i

For z =para cada fatia na dimensão i da grid com NovaN[i] fatias

Inicializar conjunto A com os nós que aparecem na fatia Z

Remover de A os nós que estão a quota atingida

If ($\sum QuotaDe\_P[a] \geq Elementos, a \in A$)

{

for j = cada elemento não associado da fatia Z

{

Para cada nó em A determine se ele aparece em alguma das outras K fatias

Se um nó em A aparece em todas as K-1 fatias

marque este nó

Se não

Ache um no que apareça em pelo menos uma fatia correspondente marque este no

Se nenhum no foi achado marque o nó que tem a maior quota

Associe o nó marcada ao elemento j

Decremente QuotaDe_P[nó marcado]

Decremente Elementos_Restantes

Se (QuotaDe_P[nó maracado] = 0)

Remova o nó marcado de A

}

}

}

Fase 3:

Restantes = $\left(\prod_{i=1}^{K} N[i] - \prod_{i=1}^{K} NewN[i] \right) \ MOD \ P$

For i = a fatia com o menor número de elementos não associados

{

c = a dimensão que a fatia pertence

inicializa A com os nós que aparecem na fatia selecionada

```
      if (restantes = 0)
            remove de A os nós que têm quota zero
                  if (Somatório de Quotas de A) + mínimo(restantes,
                  numero de nos que tem quota zero) >= Elementos não
                  alocados)
      {
      for j = cada elemento não associado de i
            {
            para cada nó em A determine se algum aparece em uma das
            outras K - 1 fatias que incluem j com elemento
      Se um nó em A aparece em todas as K-1 fatias
                        marque este nó
            Se não
            Ache um no que apareça em pelo menos uma fatia
            correspondente marque este no
            Se nenhum no foi achado marque o nó que tem a maior quota
            Associe o nó marcada ao elemento j
            Decremente QuotaDe_P[nó marcado]
            Decremente Elementos_Restantes
            Se (QuotaDe_P[nó marcado] = -1)
                  {
                  decremente restantes
                  remova o nó marcado do conjunto A
                  }
      Se (restantes = 0)
            Remova todos os processadores de A que tem quota igual a
            zero
            }
      }
}
Fase 4:
Inicializa o conjunto A com os nós que têm quota diferente de -1
Se (restantes = 0)
      Remova todos os processadores de A que têm quota igual a zero
Para cada elemento não associado restante faça
      {
      para cada nó em A determine se ele aparece em alguma das K-1
      fatias que incluem j como um elemento
      Se existe um nó que parece em todas as fatias marque o nó
      Se não
      Marque o nó que aparece em uma fatia da dimensão (atributo) mais
      acessada.
```

```
Se o nó ainda não foi marcado marque o nó com maior quota
Associe o nó marcado ao elemento j
Decremente QuotaDe_P[nó marcado]
Se (QuotaDe_P[nó marcado] = -1) decremente restante
Se (restante = 0)
      Remova todos os nós de A que tem quota igual a 0
}
```

Passo 4:

É removida a primeira condição do passo 1.

Neste caso os valores originais de M_i são substituidos por novos valores que satisfaçam a primeira condição relaxada. Os novos valores de M_i devem satisfazer ainda duas novas condições:

1. $\prod_{i=1}^{K} M_i = P$
2. os novos valores devem ficar o mais perto possível dos valores originais de M_i

Para atingir esta duas condições todos os possíveis divisores de P, consistindo de K termos, $(T_1, T_2, T_3, \ldots T_k)$ são calculados. Qualquer divisor que, com um termo calculado, é maior que o número de elementos da cada fatia da dimensão ***i***, é eliminado. Dos termos que sobrarem será escolhido o que apresentar o mínimo valor da seguinte equação:

$$\sum_{i=1}^{K} (FreqD_i * | M_i - T_i |)$$

Após calculados os novos valores de M_i são aplicados os passos 1, 2 ou 3, dependendo das condições das condições de contorno existentes.