PARALELISMO EM BANCO DE DADOS

Luiz Antônio Vivacqua Corrêa Meyer
Marta lima de Queirós Mattoso

Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ
Caixa Postal 68511, Rio de Janeiro, RJ, 21945-970
Fax: 55 21 290-6626
e-eletrônico: marta@cos.ufrj.br

XII Simpósio Brasileiro de Banco de Dados

FORTALEZA, CE - BRASIL
OUTUBRO DE 1997

ÍNDICE

I. INTRODUÇÃO

A necessidade de atendimento a um número cada vez maior de aplicações de suporte a decisão que envolvem consultas complexas sobre um volume gigantesco de dados em tempo hábil, bem como o de novas classes de aplicações não convencionais motivaram a necessidade de se explorar técnicas de processamento paralelo em Sistemas Gerenciadores de Banco de Dados (SGBDs). Um Sistema Paralelo de Banco de Dados (SPBD) pode ser definido como um SGBD implementado em uma máquina com multiprocessadores, ou seja, um computador com capacidade de processamento paralelo.

No início da década de 80, a maioria das pesquisas sobre a utilização de processamento paralelo em banco de dados abordavam o desenvolvimento de hardware específico voltado para as "máquinas de bd" ("hardware" com capacidade de processamento paralelo mais um sistema operacional projetado para suportar as operações de um sistema de banco de dados).

As previsões daquela época diziam que enquanto a velocidade dos processadores aumentaria significativamente a dos discos apenas dobraria. Desta forma, a menos que fosse encontrada uma solução para o estrangulamento de E/S, os SGBDs paralelos estariam fadados ao fracasso, diziam os críticos (DEWITT, GRAY, 1990a).

Se por um lado as previsões com respeito a evolução do hardware foram corretas, o mesmo não pode ser dito com relação aos SPBD. Diversas empresas desenvolveram e comercializaram ao longo da década produtos com sucesso, entres as quais Teradata Corporation e Tandem Computers , além de protótipos desenvolvidos em universidades como Gamma (DEWITT, GRAY, 1990b) e Bubba (BORAL et al, 1990). Este sucesso é explicado em (DEWITT , GRAY, 1992) como conseqüência de três fatores:

1. Aparecimento e adoção pelo mercado dos sistemas relacionais onde as consultas consistindo de operações uniformes aplicadas a um conjunto uniforme de dados, permitem a exploração de diversas formas de paralelismo.

2. Incapacidade dos projetistas de conceberem máquinas de grande porte("*mainframes*") com capacidade de atender a demanda de múltiplos usuários tendo acesso simultâneo à bancos de dados com volumes da ordem de terabytes.

3. Multiprocessadores baseados em microprocessadores rápidos e baratos se tornaram disponíveis por uma série de fabricantes. Estas máquinas além de oferecerem um poder de computação superior aos de grande porte, são economicamente mais viáveis e apresentam uma arquitetura modular que permite uma configuração incremental de acordo com as necessidades de processamento.

Na realidade, a idéia de SPBDs baseados em "máquinas de bd" foi descartada e atualmente a tendência se volta para arquiteturas paralelas que explorem a utilização de componentes (processadores, memória) comuns, ou seja, os SPBD não são hoje em dia dependentes de hardware específico. Entretanto o modelo de memória da máquina hospedeira tem influência na arquitetura do SPBD.

O objetivo deste trabalho é fazer uma revisão dos principais conceitos relativos aos SPBDs. Na seção II são apresentadas as principais características dos sistemas paralelos. Na seção III é feita uma análise de alguns sistemas comercializados atualmente, e finalizando o trabalho são feitas considerações sobre o uso de paralelismo em banco de dados orientados a objetos.

II. CARACTERÍSTICAS PRINCIPAIS DOS SPBDs

II.1 Métricas

Como mostrado na Figura 1, um sistema paralelo ideal deve apresentar duas propriedades (DEWITT, GRAY, 1992):

1. aceleração linear(*linear speedup*) - não leva em consideração o tamanho do problema e sim o tamanho do sistema. Se o hardware for dobrado, a tarefa deverá ser executada na metade do tempo. A aceleração linear é medida pelo ganho que é definido como a razão entre o tempo de execução com ***1*** processador e o tempo de execução com ***n*** processadores.

2. crescimento linear(*linear scaleup*) - mede a habilidade de crescimento do sistema e do problema. Se o hardware for duplicado, o sistema será capaz de executar uma tarefa duas vezes maior no mesmo espaço de tempo.

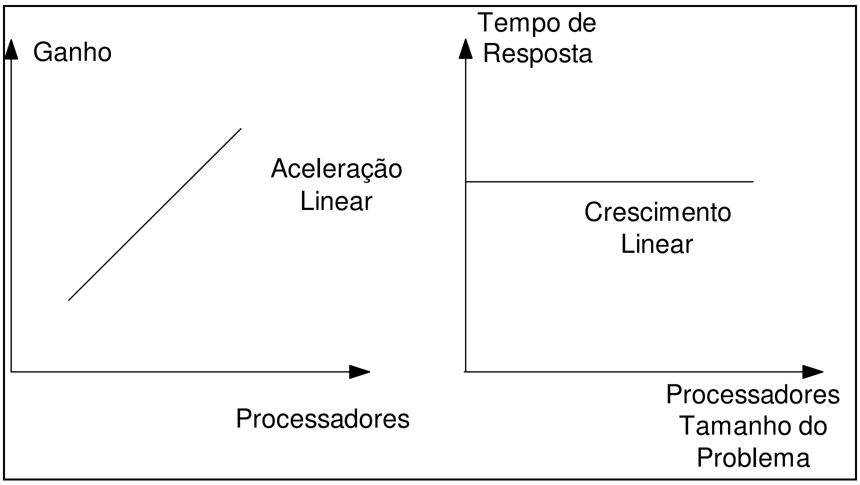

Figura 1 - Propriedades de um sistema paralelo

As principais barreiras à obtenção de aceleração e crescimento lineares são:

1. iniciação - tempo necessário para iniciar uma operação paralela. Se milhares de processadores tiverem que ser ativados, tal procedimento pode comprometer o tempo de execução.

2. interferência - aumento no tempo de resposta a cada vez que um processador tem acesso a recursos compartilhados.

3. inclinação *(skew)* - o tempo de execução de uma tarefa corresponde ao tempo de execução do processador mais lento. Quando existe um desbalanceamento de carga, o paralelismo fica comprometido.

Entretanto ganhos superlineares também podem ser alcançados, em particular nas aplicações de banco de dados, quando se consegue tirar proveito de um bom balanceamento de carga e da disponibilização dos dados em memória e *cache*; o ganho superlinear resulta do acréscimo não só de processadores mas também de memória RAM e unidades de disco.

II.2 Arquiteturas de Máquinas Paralelas

A organização da memória (principal e secundária) é um dos parâmetros da arquitetura paralela hospedeira que tem influência no projeto de SPBDs (MATTOSO, 1994). Três diferentes abordagens tem sido utilizadas: memória compartilhada (*shared memory*), memória distribuída (*shared nothing*) e disco compartilhado (*shared disk*). Algumas variações nessas três categorias clássicas também vem sendo propostas.

II.2.1 Memória Compartilhada

Neste tipo de arquitetura (Figura 2), todos os processadores tem acesso a qualquer módulo de memória ou disco através de uma rápida rede de interconexão. XPRS (STONEBRAKER et al, 1988a) e DBS3 (BOUGANIM et al, 1996) são exemplos de sistemas que utilizam tal arquitetura. Este modelo de memória apresenta duas grandes vantagens

(VALDURIEZ, 1993a): simplicidade e balanceamento de carga. Uma vez que o catálogo e as informações de controle podem ser recuperadas por todos os processadores, os procedimentos para gerenciamento do banco são similares aos empregados em sistemas com mono-processamento. O balanceamento de carga é simples já que o sistema associa as tarefas aos processadores em tempo de execução baseado na carga atual dos mesmos.

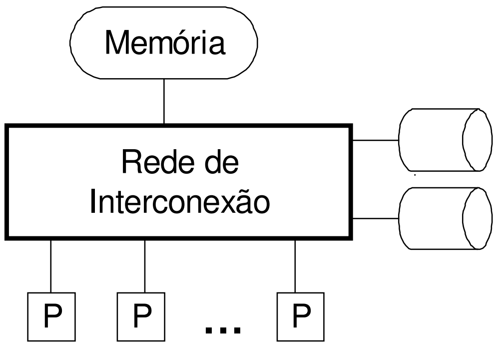

Figura 2 - Arquitetura de memória compartilhada

Como desvantagens podem ser citadas o custo, a expansibilidade limitada e a baixa disponibilidade. Os custos são altos devido a complexidade da rede de interconexão, pois é necessário que se ligue cada processador a cada módulo de memória e a cada disco do sistema. Acessos conflitantes a memória compartilhada podem causar interferência entre os processadores limitando o crescimento do sistema. Segundo DEWITT E GRAY (1992), a rede de interconexão deve ter a largura de banda (quantidade de mensagens que a rede pode suportar) da soma dos processadores e discos sendo portanto difícil construir tais redes que apresentem um crescimento linear alto. Uma vez que o espaço de memória é dividido por todos os processadores, uma falha neste componente pode afetar todos os processadores.

II.2.2 Disco Compartilhado

No modelo de arquitetura de disco compartilhado, cada processador possui uma memória exclusiva e tem acesso a qualquer unidade de disco através da rede de interconexão (Figura 3). Segundo VALDURIEZ (1993a), as principais vantagens desta arquitetura são: custo, expansibilidade, balanceamento de carga, disponibilidade e facilidade para migrar de um sistema monoprocessável. O custo da rede de interconexão é bem inferior ao modelo de memória compartilhada já que a tecnologia de *bus* padrão pode ser utilizada. Como cada processador tem a si associado um módulo de memória o problema de interferência pode ser minimizado permitindo uma expansibilidade maior (da ordem de centenas de processadores). O balanceamento de carga pode ser tão bom quanto na memória compartilhada e falhas nos módulos de memória podem ser isoladas dos outros nós possibilitando assim uma maior disponibilidade do sistema. IMS/VS Data Sharing(IBM), VAX DBMS(DEC) são exemplos de sistemas que utilizam esta arquitetura.

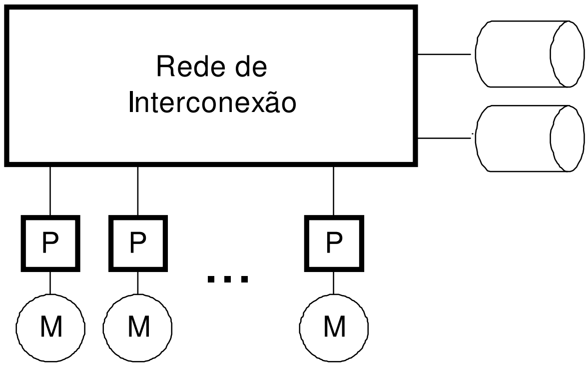

Figura 3 - Arquitetura de disco compartilhado

As principais desvantagens segundo o mesmo autor são, uma maior complexidade e potenciais problemas de desempenho. O gerenciamento de transações é mais complexo, pois não existe uma memória global para o controle de bloqueios o que aumenta a troca de mensagens entre os processadores acarretando em interferência e atrasos. O acesso aos discos compartilhados representa também, potenciais problemas de estrangulamento na E/S. Desta forma, segundo DEWITT e GRAY (1992), este tipo de arquitetura é mais adequado para aplicações "*read only*".

II.2.3 Memória Distribuída

Na arquitetura de memória distribuída, cada processador tem exclusivo acesso a sua memória e a(s) sua(s) unidade(s) de disco (Figura 4). Cada nó (processador + memória + disco) pode ser visto como um SGBD local da mesma forma que os bancos de dados distribuídos. Devido a esta similaridade, muitas das soluções projetadas para SGBDs distribuídos tais como, fragmentação, gerenciamento distribuído de transações e processamento de consultas distribuído podem ser reutilizadas neste modelo (VALDURIEZ, 1993a,OZSU, VALDURIEZ, 1996).

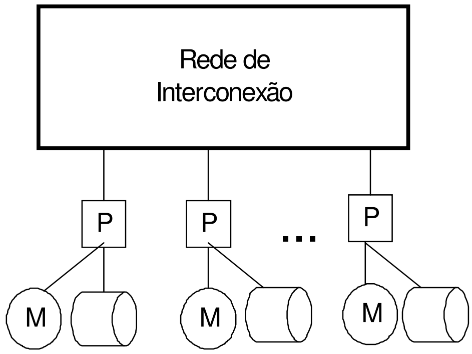

Figura 4 - Arquitetura memória distribuída

Os produtos Sybase MPP (SYBASE INC, 1996a), NonStopSQL/MP (TANDEM COMPUTERS INC, 1996a), DB2/PE (BARU et al, 1995) além dos protótipos GAMMA e BUBBA são exemplos de sistemas que utilizam este modelo de arquitetura.

As principais vantagens da memória distribuída são o baixo custo aliado a uma alta expansibilidade e disponibilidade. Permitindo um crescimento incremental, consegue suportar um número enorme de processadores minimizando inteferências através da redução de recursos compartilhados. A replicação de dados entre os nós permite que seja obtido um alto grau de disponibilidade do sistema (DEWITT, GRAY, 1992).

O maior problema desta arquitetura é o balanceamento de carga que pode degradar consideravelmente o desempenho do sistema. Isto pode ocorrer porque diferentemente das arquiteturas citadas anteriormente, o balanceamento de carga dos dados é estático (dependente da alocação inicial) e decidido baseado na localização dos dados e não na carga corrente do sistema. Além disso, o acréscimo de novos nós ao sistema requer uma reorganização dos dados (VALDURIEZ, 1993b, NORMAM et al, 1996).

Segundo VALDURIEZ (1993b), para pequenas configurações (menos que 20 processadores) a memória compartilhada pode oferecer o melhor desempenho em virtude de um melhor balanceamento de carga. Entretanto as arquiteturas de disco compartilhado e memória distribuída superam o modelo de memória compartilhada tanto em expansibilidade como em disponibilidade sendo que a memória distribuída consegue um grau de expansão muito superior as outras duas arquiteturas.

Apesar de ser uma tendência atual no mercado e nos trabalhos da literatura, (DEWIT e GRAY (1992) chegam a afirmar a existência de um consenso em relação a utilização da memória distribuída para SPBDs) alguns autores sugerem a adoção de uma arquitetura híbrida de forma a conciliar as melhores características destas arquiteturas básicas.

II.2.4 Outras Alternativas

Dada a expansibilidade limitada da arquitetura *shared-memory* e o problema de se obter um bom balanceamento de carga na *shared-nothing*, VALDURIEZ (1993b) sugere em seu trabalho uma nova arquitetura, denominada *shared-something*, constituída de um sistema shared-disk tendo em cada nó um multiprocessador shared-memory (Figura 5). Desta forma, acredita o autor, a presença de discos compartilhados facilita a migração de um sistema centralizado e o balanceamento de carga, ao mesmo tempo em que a existência de poucos mas poderosos nós simplifica a partição dos dados, a administração do banco e os ajustes de desempenho.

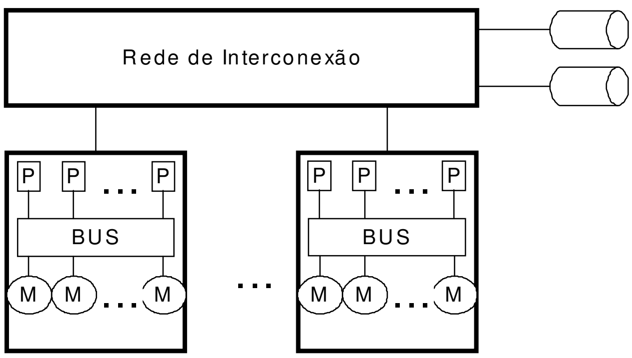

Figura 5 - Arquitetura *shared-something*

NORMAN et al (1996) em seu trabalho também citam um outro tipo de arquitetura híbrida(Figura 6), como forma de se obter a melhor configuração de hardware para um SPBD. Este modelo híbrido consiste de uma arquitetura *shared-nothing* onde cada nó é formado no nível interno como uma arquitetura *shared-memory*. Na realidade, novas versões de alguns produtos desenvolvidos para o modelo de memória distribuída começam a tirar proveito destas novas modalidades de arquitetura com modelo de memória híbrida.

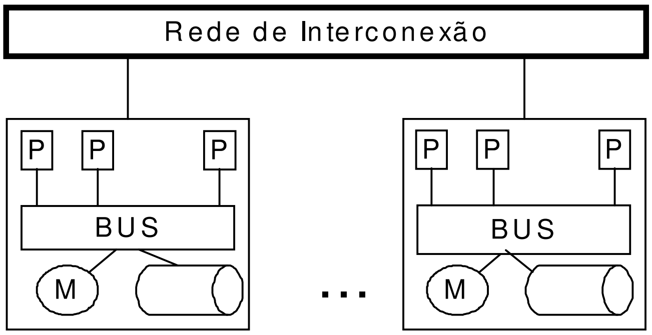

Figura 6 - Arquitetura híbrida

II.3 Distribuição dos Dados

A distribuição dos dados em SPBDs pode ser feita de duas formas: agrupada ou desagregada. No primeiro caso, cada relação ou classe é alocada como um todo a um dos nós do sistema, enquanto que no segundo, é feito um projeto de distribuição que envolve duas fases: Fragmentação e Alocação.

No modelo relacional, a fragmentação consiste da divisão de cada relação global em fragmentos através de um processo de seleção de tuplas (fragmentação horizontal), projeção de colunas (fragmentação vertical) ou uma combinação de ambas operações (fragmentação híbrida). Analogamente, no modelo orientado a objetos, a geração de fragmentos pode ser obtida através da seleção de objetos, da seleção de atributos ou da combinação de ambas.

A alocação é o processo de distribuição dos fragmentos pelos nós/processadores do sistema. Ela pode ser realizada com ou sem replicação dos fragmentos.

II.3.1 Fragmentação Horizontal

A fragmentação horizontal pode ser feita de forma primária ou derivada. Neste caso, a fragmentação de uma classe ou relação não é baseada nas propriedades dos seus próprios atributos mas em função da fragmentação de outra classe ou relação. Ela é usada para facilitar as operações de junção e navegação entre fragmentos. Já a fragmentação primária, utiliza três técnicas para a geração de fragmentos: faixa de valores, "hashing" e circular.

II.3.1.1 Fragmentação por Faixa de Valores

Neste tipo de fragmentação, os fragmentos são obtidos em função dos valores dos atributos da classe/relação levando-se em consideração os predicados existentes nas aplicações dos usuários. Em geral, escolhe-se um ou mais atributos baseado nas cláusulas dos predicados, sendo a faixa de valores obtida tomando como base o domínio dos atributos escolhidos.

II.3.1.2 Fragmentação por Função de Hashing

A relação/classe é fragmentada de acordo com o valor de uma função de *hashing* aplicada sobre um atributo, de forma a relacionar cada tupla/objeto da relação/classe a um fragmento.

II.3.1.3 Fragmentação Circular

A fragmentação circular consiste da associação seqüencial de cada tupla/objeto da relação/classe a um fragmento sem levar em consideração o valor de nenhum atributo. Desta forma, se a relação/classe contiver *n* tuplas/objetos e o número de fragmentos a serem gerados for *m*, cada fragmento receberá *n/m* tuplas/objetos.

II.3.2 Fragmentação Vertical

A fragmentação vertical é o processo que divide a relação/classe, agrupando em cada fragmento uma parte dos atributos existentes na relação/classe original e replicando-se em cada fragmento a chave primária/identificação do objeto da relação/classe de forma a permitir a sua reconstrução a partir da junção dos fragmentos.

O seu objetivo é identificar fragmentos de tal forma que várias aplicações possam ser executadas com acesso à somente uma partição da relação/classe global. Entretanto caso uma aplicação necessite de acesso à dados presentes em mais de um fragmento, a fragmentação vertical não é benéfica visto que será necessária a realização de junções entre os fragmentos para a obtenção do resultado (CERI, PELAGATTI, 1984).

II.3.3 Considerações quanto a Distribuição

Segundo DEWITT e GRAY (1992), a distribuição circular é excelente se todas as aplicações tem acesso as relações/classes varrendo-as completamente e de forma seqüencial em cada consulta. A distribuição por função de "hashing" é adequada às aplicações que necessitam tanto de acesso seqüencial como de acesso associativo. Neste último caso, as tuplas/objetos cujo um determinado atributo apresentem um valor específico são direcionadas para um determinado nó evitando o trabalho desnecessário de se iniciar a consulta em múltiplos processadores. Na fragmentação por faixa de valores as tuplas/objetos são agrupadas em partições segundo os valores de um ou mais atributos. Ela é boa para acessos seqüenciais e associativos permitindo que consultas cuja a cláusula condicional especifique os atributos de fragmentação sejam direcionadas aos nós apropriados. Porém pode apresentar problemas em função do comportamento(skew) dos dados, ou seja, todos os dados residirem em uma partição ou todas as execuções ocorrem em um único nó.

A desagregação pode ser total, onde todos os nós do sistema recebem um fragmento da relação, ou parcial. O primeiro caso pode ser prejudicial na presença de relações com volume de dados pequeno ou de sistemas com um número grande de nós. Neste caso, a desagregação parcial , onde os fragmentos são armazenados em apenas alguns dos nós, pode ser benéfica permitindo que um grupo de processadores fiquem livres para atuar em outras operações. Em trabalho recente, MEHTA e DEWITT (1997) fazem uma análise de desempenho mostrando que excetuando-se os casos de relações muito pequenas, a desagregação total deve ser sempre usada em arquiteturas de memória distribuída (*shared-nothing*).

Como conseqüência das operações inclusão, exclusão e alteração, o banco de dados pode vir a apresentar problemas com relação a má distribuição de dados entre os diversos nós ocasionando um mal balanceamento de carga. Além disto, um acréscimo no volume de dados pode implicar na necessidade de expansão do sistema através do acréscimo de mais processadores e discos. Surge então a necessidade de se fazer uma reorganização dos dados. Segundo VALDURIEZ (1993a), esta operação deveria ser realizada on-line e de forma paralela e transparente, o que não ocorre nos sistemas atuais. As aplicações não deveriam necessitar de recompilações após uma reorganização permanecendo independentes da localização dos dados.

Outro aspecto ainda em aberto é a questão da replicação dos dados. A solução mais simplista é manter duas cópias dos mesmos dados em nós separados. Entretanto em caso de falha do nó, a carga sobre o nó que contém a cópia irá dobrar afetando portanto o balanceamento de carga. Uma alternativa a esta solução é distribuir a cópia de segurança entre um conjunto de nós. Em caso de falha, a carga referente ao nó afetado é distribuída entre os outros nós equilibrando o balanceamento de carga. Porém a reconstrução dos dados afetados através de suas cópias é mais onerosa. Outra alternativa é armazenar a cópia em um nó contíguo e em caso de falha distribuir pelos nós restantes os dados do nó afetado o que torna a manutenção da consistência mais barata.

II.4 Paralelismo nas Consultas em SPBD

Existem três níveis de paralelismo que podem ser obtidos no processamento de consultas em SPBDs: paralelismo entre consultas(*interquery*), dentro da consulta (*intraquery*, *interoperator*, *pipelined*) e dentro da operação (*intraoperator*, *partitioned*).

O primeiro caso caracteriza-se pela execução concorrente de múltiplas consultas de diversos usuários. O paralelismo dentro da consulta consiste da execução paralela de diferentes operadores dentro de uma única consulta. Se um operador envia o resultado obtido para o seguinte, ambos podem executar em paralelo, diminuindo o tempo de resposta da consulta. Os benefícios a serem obtidos com o paralelismo dentro da consulta é limitado por três fatores (DEWITT, GRAY, 1992):

1. O encadeamento entre operações geralmente não é muito longo.

2. Alguns operadores relacionais não emitem sua saída até que tenham processado toda a sua entrada.

3. Muitas vezes o custo de execução de um operador é muito maior que o de outros.

O paralelismo dentro da operação é alcançado através da distribuição dos dados de entrada pelos diversos nós da máquina paralela e da replicação da operação que é executada simultaneamente sobre os fragmentos. Nesse nível de paralelismo encontram-se os principais algoritmos de consultas em Banco de Dados .

O processamento de consultas em um ambiente convencional consiste na geração de planos de tal forma que as consultas são executadas seqüencialmente. O problema de otimização nestes casos , geralmente é resolvido via busca exaustiva dentre os possíveis planos, escolhendo-se aquele que apresenta os menores custos.

Em SPBDs, embora o objetivo continue sendo o mesmo, ou seja, a escolha do plano menos oneroso, novas complexidades são adicionadas devido ao espaço de soluções ser consideravelmente maior. A otimização de uma consulta pode ser feita de forma estática ou dinâmica. No primeiro caso, ela é realizada em tempo de compilação tendo como argumentos informações tais como número de tuplas, fragmentação, índices etc. Entretanto a utilização desta forma de otimização não leva em consideração parâmetros como o número de processadores livres ou a área de buffer disponível no momento da execução da consulta e que podem afetar os custos dos planos gerados (HONG, 1992).

A fragmentação é a base para a exploração do paralelismo nas consultas em banco de dados. O princípio básico empregado em arquiteturas com memória distribuída é o de execução onde estiver localizado o dado, explorando desta forma a distribuição e enviando as operações aos dados.

II.4.1 Paralelismo Dentro da Operação

A paralelização de uma operação de consulta é baseada em algoritmos globais e locais. O algoritmo global decompõe a operação para vários locais onde outros algoritmos são aplicados (VALDURIEZ, 1993a). No caso da operação unária de **seleção**, supondo que a relação esteja fragmentada em *n* partições, inicialmente os fragmentos são alocados aos respectivos processadores que em seguida recebem o predicado da seleção e avaliam paralelamente os seus fragmentos utilizando para tal um algoritmo seqüencial de seleção. Contudo, caso o predicado especificado envolva os atributos utilizados para a fragmentação, somente alguns processadores serão invocados. Essa operação é dita naturalmente paralelizável pois o custo de comunicação é mínimo.

Segundo GRAEFE (1993), existem duas técnicas para a paralelização de operações binárias: distribuição simétrica e fragmentação com replicação sendo que em ambos os casos o resultado final é a união dos resultados locais. No método de distribuição simétrica, as duas entradas são distribuídas de acordo com os atributos relevantes para a operação (atributo de junção no caso desta operação) para então serem processadas em cada nó. O método para fragmentar as entradas geralmente é uma função de *hashing* enquanto que os algoritmos locais adotam estratégias às vezes diferentes de acordo com o sistema. Na fragmentação com replicação, uma entrada é fragmentada enquanto que a outra é enviada integralmente(“*broadcast*”) para todos os processadores. Em geral a maior relação permanece nos locais em que se encontra fragmentada sendo transmitida a menor.

Embora existam algoritmos paralelos para todas as operações da álgebra relacional, a maioria dos trabalhos se concentram na operação de **junção**. De uma forma geral os SPBDs implementam um ou mais tipos de algoritmos de junção seqüencial e que são estendidos para explorar o paralelismo do sistema.

SCHNEIDER e DEWITT (1989) em seu trabalho fazem uma avaliação de desempenho de quatro algoritmos de junção implementados no sistema GAMMA. A estratégia utilizada neste SPBD para alguns dos algoritmos, consiste em utilizar processadores sem disco para a execução das junções. Segundo os autores, esta estratégia permite a obtenção de um melhor desempenho pois reduz a carga dos processadores com discos associados. LERNER et al (1996) estudam de forma analítica e prática o desempenho de algoritmos paralelos de junção em arquiteturas com memória distribuída onde é proposto e implementado uma estratégia de paralelização com equilíbrio dinâmico de carga. A seguir serão descritos três algoritmos muito utilizados pelos SPBDs na operação de junção paralela.

II.4.2 Junção por iterações aninhadas (nested-loops)

É o mais simples e mais genérico algoritmo de junção. Consiste da composição do produto cartesiano das duas relações em paralelo. Na forma seqüencial, para cada tupla de uma entrada (entrada de fora) varre seqüencialmente toda a outra entrada (entrada de dentro) descobrindo as tuplas que satisfazem a condição de junção (GRAEFE, 1993). Na forma paralela a idéia é fragmentar uma das relações pelos processadores disponíveis e realizar as junções em paralelo entre os fragmentos e a outra relação completa.

O algoritmo pode ser dividido em duas fases. Supondo a junção entre as relações R e S, no caso de arquiteturas com memória distribuída, na primeira fase uma das relações tem seus fragmentos(R_i) enviados e replicados paralelamente em cada nó onde existe um fragmento(S_j) da outra relação. Em seguida, cada nó processa a junção de R com o fragmento(S_j) em paralelo (OZSU, VALDURIEZ, 1991). No caso de uma arquitetura com memória compartilhada, não existe o aproveitamento da localização física dos dados junto aos processadores. Assim, os fragmentos da relação maior são alocados aos processadores disponíveis enquanto que a relação de menor cardinalidade é enviada aos nós de forma integral.

II.4.3 Junção com ordenação e fusão (sort-merge)

A versão paralela deste algoritmo (figura 8) implementada no sistema GAMMA constitui-se em uma adaptação simples da versão seqüencial (SCHNEIDER, DEWITT, 1989). Supondo R a relação de menor tamanho, esta é primeiramente redistribuída pelos nós através do uso de uma tabela de divisão que contém as entradas para cada processador com disco associado. Para tal é usada uma função de hashing que é aplicada sobre o atributo de junção de cada tupla determinando o nó apropriado. À medida em que as tuplas da relação chegam aos nós, elas são armazenadas em arquivos temporários. Quando toda a relação R acabou de ser redistribuída, cada um dos arquivos temporários é ordenado em paralelo. A outra relação operando S é então processada da mesma maneira. Uma vez que a mesma função de hashing é usada para redistribuição de ambas relações, as tuplas com possibilidade de junção residirão nos fragmentos localizados nos mesmos nós. Por fim a ligação (merge) dos fragmentos é realizado em paralelo finalizando a junção.

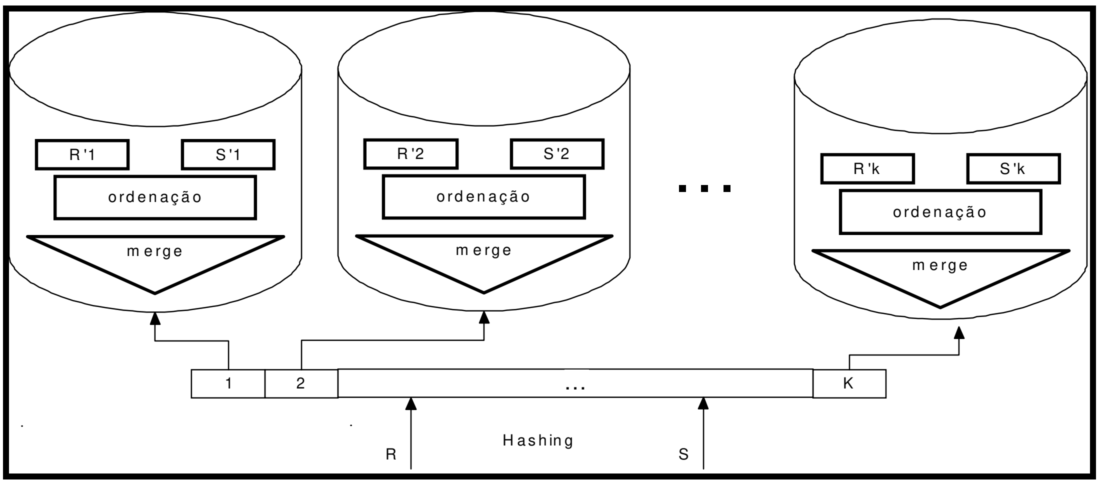

Figura 8 - *Sort-Merge* paralelo

II.4.4 **junção baseada em *hash***

Este algoritmo é muito utilizado quando não existem índices para as colunas de junção. Ele é geralmente mais eficiente que o algoritmo anterior visto que não existe necessidade de ordenação. Na forma seqüencial, a menor tabela (também chamada de entrada de construção) é lida e endereçada para uma tabela em memória através da utilização de uma função de hashing aplicada sobre o atributo de junção. Em seguida a segunda relação é varrida seqüencialmente e para cada tupla lida é feita uma verificação na tabela de hashing para estabelecimento ou não da junção. Na sua versão paralela (figura 9), SCHNEIDER e DEWITT (1989) utilizam o mesmo artifício usado no algoritmo *sort-merge*, ou seja, é inserida uma tabela de distribuição que endereça as tuplas das relações para os nós apropriados. Caso a memória disponível não seja suficiente para o armazenamento dos fragmentos da primeira relação, torna-se necessária a utilização de arquivos temporários para o armazenamento da tabela de hashing.

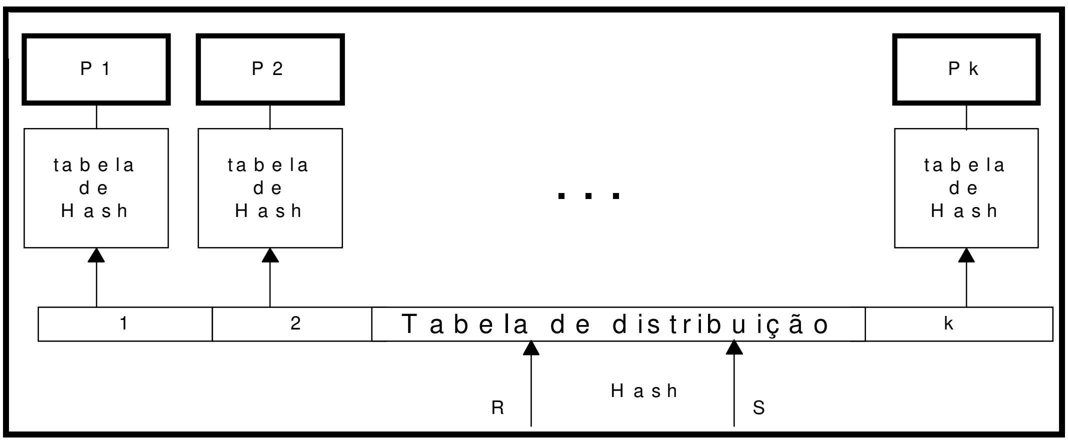

Figura 9 - Junção por *hashing*

III. ALGUNS SISTEMAS EXISTENTES

O objetivo desta seção é fazer uma breve descrição de alguns dos mais importantes SPBDs encontrados no mercado, apresentando suas principais características. Foram escolhidos 6 sistemas em função dos mesmos serem os mais referenciados na literatura. Além dos produtos comercializados serão apresentados os protótipos pioneiros na exploração de paralelismo em SGBDs, GAMMA e Bubba, cujas abordagens muito contribuíram para as soluções adotadas em vários sistemas paralelos, bem como novos protótipos que exploram o modelo de dados orientado a objetos.

- GAMMA (DEWITT et al, 1990)

É um sistema desenvolvido na Universidade de Wisconsin e que pode ser classificado como uma "máquina de bd". Possui um sistema operacional próprio, o NOSE, especialmente projetado para suportar as funcionalidades de um SGBD e utiliza o sistema de armazenamento WISS (Winconsin Storage System).

Utiliza o modelo de memória distribuída e sua primeira versão(1985) consistia de 18 processadores VAX 11/750 cada qual com 2 megabytes de memória, sendo que oito dos processadores estavam conectados a discos com capacidade de 333 megabytes. Em 1988, foi lançada a segunda versão constituída de 32 processadores iPSC/2 da Intel, onde cada nó é configurado como uma CPU 386 com 8 megabytes de memória e disco com capacidade de 330 megabytes.

Utiliza a técnica de fragmentação horizontal distribuindo as tuplas por faixa de valores, circularmente ou por função de *hashing*, sendo que as informações a respeito dos fragmentos são armazenadas no catálogo do sistema. As consultas ao banco podem ser efetuadas através de uma linguagem própria que também pode ser embutida em um programa escrito em linguagem C.

Os algoritmos para as operações relacionais foram escritos para um ambiente monoprocessável e são paralelizáveis através da utilização do mecanismo de uma tabela de distribuição que mapeia as tuplas a serem operadas para os nós apropriados. São utilizados quatro algoritmos de junção baseados em distribuição por *hash: sort-merge*, *Grace*, *hash* simples e *hash* híbrido, sendo este último adotado como *default* por apresentar o melhor desempenho no sistema.

- Bubba (BORAL et al, 1990)

É um sistema desenvolvido no MCC (Microeletronics and Computer Technology Corporation), Texas, EUA. O modelo de dados adotado é o relacional estendido, possui um sistema operacional próprio, o BOS (Bubba Operating System), implementado através de modificações no sistema UNIX da AT&T.

Utiliza o modelo de memória distribuída e está implementado em uma máquina com 40 nós cada qual com uma CPU Motorola 68020 e 2 megabytes de memória. Dispõe de uma linguagem de programação própria FAD que estende as funcionalidades do modelo relacional e é usada tanto para descrição do esquema como para manipulação dos dados. As relações podem ser fragmentadas horizontalmente por faixa de valores ou hash. Como diferencial existe a desagregação parcial, que é realizada em função de diversos parâmetros como o tamanho da relação e a freqüência de acesso.

- Teradata (NCR COMPANY, 1996)

A companhia Teradata Corporation foi fundada em 1979 com o objetivo de desenvolver um produto específico para suportar processamento interativo em larga escala. Em 1984 foi concluído o primeiro sistema, Data Base Computer (DBC), uma máquina de banco de dados com arquitetura de memória distribuída. Em 1991 a companhia foi adquirida pela NCR Company.

Até janeiro de 1996, o sistema era disponível somente em plataformas paralelas da NCR(DBC/1012 e NCR System 3600) que faziam uso do sistema operacional TOS(Teradata Operating System), encontrando-se agora disponível em versão para UNIX.

A plataforma de hardware desta nova versão é a máquina NCR WorldMark 5100 que pode ser configurada segundo três arquiteturas diferentes. O modelo 5100S é um sistema com arquitetura de memória compartilhada constituído por um único nó que suporta até 32 processadores Pentium da Intel e utiliza a tecnologia de *disk-array* com capacidade máxima de 1 terabyte. O modelo 5100C é um sistema com arquitetura de disco compartilhado formado por até 8 nós 5100S que compartilham um conjunto de controladoras de disco. O último modelo é o 5100M que conecta múltiplos sistemas 5100c em um ambiente de memória distribuída. O sistema utiliza a técnica de fragmentação horizontal adotando a política de distribuição por função de *Hashing*. Para a realização das operações de junção são utilizados os algoritmos de interações aninhadas, ordenação e fusão e *hash* que são escolhidos pelo otimizador em tempo de compilação. Funcionando com uma arquitetura cliente-servidor, o acesso ao banco de dados pode ser feito por SQL padrão ou através de programas COBOL, PL/I e C de acordo com o sistema operacional da plataforma cliente.

- Informix(INFORMIX SOFTWARE INC, 1995, 1996)

Este sistema possui versões para três tipos de arquitetura hospedeira. A versão "*OnLine Dynamic Server*" foi introduzida em 1993 sendo voltada para modelos de memória compartilhada. Em 1995 foi lançada a versão *OnLine XPS* (Extended Parallel Server) que pode ser utilizada em máquinas com arquitetura de memória distribuída compostas por nós com memória compartilhada ou com um único processador.

Utiliza a técnica de fragmentação horizontal distribuindo as tuplas de forma circular, por faixa de valores, através de função de *hashing* ou por expressões definidas sobre os atributos da tabela. As operações são executadas em paralelo fazendo uso do operador de trocas (*exchange operator*) proposto no protótipo Volcano(GRAEFE, 1990). As operações de junção são realizadas utilizando-se os algoritmos de iterações aninhadas, ordenação e fusão e junção por *hash*. O sistema pode ser executado em plataformas de diversos fabricantes que executem o sistema operacional UNIX(R).

- NonStop SQL/MP (TANDEM COMPUTERS INC, 1996)

Este produto da Tandem Computers Incorporated é projetado para as máquinas paralelas deste fabricante que utilizam um sistema operacional próprio. Sua arquitetura com memória distribuída suporta configurações que variam de 2 até 4080 processadores. As relações são fragmentadas horizontalmente segundo a técnica de faixa de valores. Para a realização da operação de junção são utilizados os algoritmos de iterações aninhadas, ordenação com fusão e *hash*. O sistema suporta várias interfaces que permitem que um conjunto diverso de aplicações tenham acesso aos seus serviços. Os programas podem ser escritos em SQL padrão ou embutidos nas linguagens C e COBOL.

- Sybase MPP (SYBASE INCORPORATED, 1996a, 1996b)

É um sistema da empresa Sybase Incorporated, voltado para arquiteturas com memória distribuída. Em produção nas máquinas AT&T 3000 está disponibilizando uma versão para máquinas IBM RS/6000. As tabelas podem ser fragmentadas horizontalmente (por função de *hashing* ou por faixa de valores) ou residirem integralmente em um único nó. Em testes realizados com um computador IBM RS/6000 atingiram crescimento próximo do linear em configurações variando de 1 até 128 nós.

- DB2/PE (BARU et al, 1995)

É um produto da IBM Corporation que opera em máquinas paralelas com memória distribuída e com o sistema operacional AIX, tais como o SP2. Suporta a distribuição total ou parcial dos fragmentos que são gerados horizontalmente através de função de hashing e utilizando o conceito de grupos de nós (*nodegroups*). Um *nodegroup* é um subconjunto de nós do sistema paralelo. Ao ser criada uma tabela é especificado o *nodegroup* onde os seus fragmentos residirão e os atributos que serão usados para a fragmentação.

A operação de junção paralela é realizada através do uso das estratégias de junção direcionada (*direct join*), junção com fragmentação e replicação (*broadcast join*) e junção com distribuição simétrica (*repartitioned join*). No primeiro caso uma das relações é re-direcionada aos nós onde residem os fragmentos da segunda relação através da aplicação de uma função de hashing sobre os atributos nas quais a segunda foi fragmentada. A segunda estratégia é a mesma apresentada no algoritmo de iterações aninhadas, na qual uma das relações é replicada integralmente por todos os nós onde se encontram os fragmentos da segunda relação. A última estratégia redistribui as duas relações de acordo com os atributos de junção. A escolha da melhor estratégia para a junção é baseada em avaliações de custo que levam em consideração o tamanho e a localização dos fragmentos, a existência de índices nos atributos de junção entre outros.

Testes realizados em configurações variando de 4 até 32 processadores e no tamanho da base entre 10 e 100 Gb apresentaram resultados com aceleração próxima a linear em utilitários(“load”, criação de índices) e para a maioria dos tipos de consultas (exceção para “scan” de tabelas a partir de um determinado número de nós, ou seja, quando o tamanho do fragmento se torna pequeno e não compensa o custo de se iniciar a operação no nó).

- Oracle7 (ORACLE CORPORATION, 1996a, 1996b)

É um sistema da Oracle Corporation que dispõe de um servidor paralelo que executa em máquinas com arquitetura de disco compartilhado, memória compartilhada e memória híbrida composta por diversos nós “*shared-memory*” compartilhando um conjunto de discos. Embora não seja obrigatório, uma vez que os dados são redistribuídos em tempo de execução, o sistema trabalha com três formas de fragmentação: fragmentação de arquivos, distribuição agrupada de tabelas e fragmentação horizontal por faixa de valores. Para resolução das operações de junção são utilizados os algoritmos de interações aninhadas e ordenação e fusão. Testes realizados atingiram aceleração linear em configurações de até 20 processadores (memória compartilhada) e 100 processadores em ambiente de memória híbrida.

- XPRS (STONEBRAKER et al, 1988a, HONG, 1992)

XPRS(eXtended Postgres on Raid and Sprite), desenvolvido na Universidade da Califórnia em Berkley, EUA, é um sistema relacional estendido implementado em uma arquitetura com memória compartilhada. O computador hospedeiro é a máquina Sequent executando o sistema operacional Dynix com 12 processadores e 7 discos.

A otimização de consultas é realizada em duas fases: na primeira é escolhido o melhor plano seqüencial assumindo que toda a área de *buffer* está disponível. Na segunda, é escolhida a melhor paralelização do plano escolhido em função da memória e do número de processadores livres na hora da execução. O sistema trabalha com duas formas de distribuição dos dados: faixa de valores e páginas de disco. Para resolução da operação de junção utiliza algoritmos de iterações aninhadas, ordenação e fusão, e *hash*.

- DBS3 (BOOUGANIM et al, 1996)

DBS3 (Database System for Shared Store) é um protótipo desenvolvido em parceria entre a BULL e o INRIA na França. Adota o modelo de memória compartilhada e foi implementado na máquina Encore Multimax com 10 processadores e na plataforma Kendal Square Research KSR1 com 72 processadores.

O seu objetivo é prover altos desempenhos para consultas expressas em ESQL, uma extensão da linguagem SQL que é mapeada para uma extensão da álgebra relacional (LERA). As relações são fragmentadas estaticamente por aplicação de função de *hashing* em um ou mais atributos.

- IDEA (BOUGANIM, 1996)

O projeto IDEA(Intelligent Database Environment for Advanced Applications) está sendo desenvolvido no INRIA, França, e trabalha com uma arquitetura de máquina hospedeira híbrida, formada por um modelo de memória distribuída onde cada nó internamente constitui-se de uma máquina com memória compartilhada. O sistema adota o modelo de dados orientado a objetos e trabalha com duas políticas de distribuição: primária ou secundária. No primeiro caso, as coleções de objetos são fragmentadas e as partições geradas alocadas aos nós do sistema. Na distribuição secundária as referências(OID, PID onde PID é o identificador físico do objeto) e não os objetos é que são distribuídos. As partições podem ser geradas por fragmentação circular, faixa de valores, hashing e localização em relação a outra partição.

- SHORE (CAREY, 1994, DEWITT et al, 1996)

O SHORE(Scalable Heterogeneous Object REpository) é um protótipo desenvolvido na Universidade de Wisconsin, EUA, projetado para execução em máquinas paralelas com memória distribuída. Ele é executado como um grupo de processos que se comunicam denominados *shore servers.* Estes processos executam duas funções básicas: gerenciar objetos persistentes e prover o acesso aos objetos para as aplicações.

As aplicações requisitam objetos aos servidores primários que obtém as páginas nas quais os objetos residem transferindo-os para as aplicações. Se um objeto requisitado reside em um disco gerenciado pelo servidor primário, esta requisição é efetivada localmente. Caso contrário, o servidor primário requisita uma cópia da página ao nó apropriado e ao recebê-la armazena no seu “cache” de páginas local, retornando para a aplicação uma cópia do objeto.

O sistema utiliza o conceito de *ParSet* (Parallel Set). Um *parset* é definido como uma coleção de objetos de um mesmo tipo ou subtipo apropriado. Existem dois tipos de parsets: os primários que tem implicação física e especificam os nós do sistema onde residirão os fragmentos e o método de fragmentação adotado; e os secundários que representam logicamente a coleção de objetos desconhecendo suas localizações. A geração dos fragmentos pode ser feita por faixa de valores, *hashing* ou de forma circular e o desagrupamento das coleções é total.

A estratégia de distribuição para os *Parsets* primários deve ser escolhida no momento de sua criação. Quando um objeto é incorporado ao *parset,* o método de distribuição é chamado para a determinação do nó onde o objeto deve ser alocado.

- ParGOA (TAVARES, SOARES, 1995, MEYER, MATTOSO, 1997)

O ParGOA (Gerente de Objetos Armazenados Paralelo) é um protótipo desenvolvido no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ e utiliza o modelo de dados orientado a objetos. O sistema utiliza o PVM(GEIST et al, 1994) como ferramenta de configuração de uma máquina paralela virtual e possui versões para dois tipos de arquitetura hospedeira: disco compartilhado e memória distribuída. No modelo de disco compartilhado, utiliza a técnica de fragmentação horizontal circular com distribuição dinâmica. Em tempo de execução da consulta, o conjunto de identificadores de objetos (IDOs) da coleção alvo presente na operação é distribuído de forma igualitária pelos processadores em conjunto com a solicitação do serviço.

Na versão com memória distribuída, a técnica de fragmentação horizontal utilizada é a de faixa de valores, com distribuição estática pelos nós, ou seja, em tempo de carga dos objetos. Além da fragmentação primária, o protótipo oferece a fragmentação derivada de classes, permitindo que classes que se interelacionem tenham seus fragmentos residindo em um mesmo nó.

O protótipo trabalha com um gerente de distribuição de consultas que tira proveito da localidade dos fragmentos para direcionar as consultas somente aos nós necessários a sua resolução. Se o predicado da consulta for simples e sem navegação entre classes, ela é resolvida no âmbito do próprio nó. Entretanto, caso envolva navegação entre classes, é possível que objetos referenciados no predicado, residam em locais diferentes de onde a consulta está sendo processada. Para resolver tal situação dois processos são executados em cada nó. O processo de consultas é o responsável pela avaliação do predicado enviado e pelo processamento da consulta, enviando ao gerente de distribuição a relação dos IDOs da coleção alvo que satisfazem a operação. O processo de objetos responde pelo atendimento das referências externas solicitadas.

A Tabela 1, apresenta as principais características dos sistemas apresentados levando-se em consideração alguns parâmetros da arquitetura hospedeira, da política de distribuição de dados e do modelo de dados adotado.

	Arquitetura Hospedeira			Distribuição de Dados			
	Modelo de Memória	Sistema Operac.	Núm de Nós Aval.	Fragm.	Desag.	Técnica	Modelo de Dados
Gamma	Distribuída	NOSE	32	Horiz.	Parcial	Faixa Hashing Circular	Relac.
Bubba	Distribuída	BOS	40	Horiz.	Parcial	Faixa Hashing	Relac. Estend.
Teradata	Compart. Distribuída Híbrida	UNIX	512	Horiz.	Total	Hashing	Relac.
Informix	Compart. Distribuída Híbrida	UNIX	32 (híbrida)	Horiz.	Total	Faixa Hashing Circular	Relac.
NonStop SQL/MP	Distribuída	Próprio	4080	Horiz.	Total	Faixa	Relac.
Sybase MPP	Distrib.	UNIX / AIX	128	Horiz.	Total	Faixa Hashing	Relac.
DB2/PE	Distrib.	AIX	64	Horiz.	Total / Parcial	Hashing	Relac.
Oracle7	Compart. Distribuída Híbrida	Vários	100	variável	Parcial	Agrupado / Faixa	Relac.
XPRS	Compart.	Dynix	12	Horiz.	Parcial	Faixa páginas	Relac. Estend.
DBS3	Compart.	Não fornecid.	72	Horiz.	Total	Hashing	Relac.
IDEA	Híbrida	UNIX	não inform.	Horiz.	Parcial	Faixa Hashing Circular	Orient. a Objetos
SHORE	Distrib.	UNIX	não inform.	Horiz.	Total	Faixa Hashing Circular	Orient. a Objetos
ParGOA	Disco Comp. Distrib.	UNIX	10	Horiz.	Total / Parcial	Faixa Circular	Orient. a Objetos

Tabela 1- Características dos SGBDs Paralelos

Observando-se Tabela 1, verifica-se que alguns produtos oferecem versões para os mais diferentes modelos de memória. Além disto, projetos que inicialmente possuíam seu hardware próprio, passam agora a utilizar máquinas genéricas com sistema operacional baseado no UNIX.

Já na política de distribuição de dados, observa-se que independente do modelo de dados e do modelo de memória, todos os sistemas adotam a fragmentação horizontal, com ênfase nas técnicas de faixa de valores e função de *hashing*. Entretanto é neste campo onde se concentram mais trabalhos abordando a distribuição de objetos em memória distribuída, onde são propostas soluções tanto por fragmentação horizontal como por distribuição agrupada de classes e fragmentação vertical.

IV. CONSIDERAÇÕES SOBRE PARALELISMO EM SGBDOOs

Embora tenham sido apresentados protótipos de SGBDOO que adotam o processamento paralelo, a combinação entre navegação de objetos e processamento paralelo é um desafio. O modelo orientado a objetos dispõe de mecanismos tais como agregação e generalização que facilitam a representação de objetos complexos e seus inter-relacionamentos. Entretanto, o desempenho dos SGBDOO em aplicações que envolvam a execução de consultas complexas que recuperam uma grande quantidade de objetos e seus relacionamentos é ainda insatisfatório (TAHKORE, SU, 1994).

Uma forma de se aumentar o desempenho dos SGBDOO é adotar a arquitetura cliente-servidor que divide os componentes de um sistema entre os dois processos, acarretando numa distribuição de trabalho e em um processamento concorrente. Desta forma, por exemplo, enquanto um cliente analisa os resultados de uma consulta em sua estação de trabalho, o servidor está disponível para o atendimento as solicitações de outros clientes.

A unidade de comunicação entre os processos cliente e servidor é geralmente feita de duas maneiras: objetos ou páginas. Os sistemas que adotam a primeira estratégia são chamados servidores de objetos. Desta forma os clientes requisitam objetos ao servidor que os recuperam do banco de dados e os enviam para os clientes. Os sistemas que adotam a segunda estratégia são chamados servidores de páginas. Neste caso a unidade de transferência entre os clientes e o servidor é um bloco físico de dados como páginas ou segmentos ao invés de objetos.

Uma segunda forma de se aumentar o desempenho dos SGBDOO é a exploração do paralelismo. Segundo DEWITT e GRAY (1990a, 1996), a construção de bancos de dados orientados a objetos paralelos apresenta uma série de novos desafios, tais como a distribuição de objetos e o processamento de consultas.

Enquanto que no modelo relacional o paralelismo se apoia no processamento de conjuntos uniformes de dados através de comandos predefinidos, no modelo OO, os objetos pertencem a classes, que não necessariamente caracterizam um conjunto. Os objetos são manipulados por métodos e por comandos existentes na interface de acesso e de manipulação de objetos (MATTOSO, 1994).

A paralelização de métodos escritos pelos usuários é uma questão de difícil solução. De acordo com DEWITT e GRAY (1990a) e MATTOSO (1993) três abordagens podem ser adotadas: Uma forma de paralelizá-los seria fazer com que fossem escritos utilizando operadores baseados numa extensão da álgebra relacional como nas linguagens de consulta. Outra opção seria a utilização de uma linguagem paralela para programação dos métodos. Numa terceira opção, o próprio programador explicitaria o paralelismo através de comandos do tipo "*parallel do*".

Diferente do modelo relacional, onde todas as tabelas são persistentes, na orientação a objetos é preciso determinar sobre que conjuntos de objetos será feita a distribuição, ou seja, se todas as classes terão seus objetos distribuídos ou somente as que forem do tipo coleção (persistentes e com gerência de extensão). É necessário também levar em consideração como serão distribuídos os objetos de coleções presentes em atributos multivalorados.

Com relação ao modelo de memória mais adequado para a exploração de paralelismo em SGBDOO, alguns autores como VALDURIEZ (1993a), consideram ser mais difícil a utilização de memória distribuída, devido a complexidade de conciliar a distribuição e o agrupamento ao mesmo tempo. Segundo o autor, tanto a memória compartilhada como a de disco compartilhado permitem que as soluções tradicionais de agrupamento utilizadas em sistemas seqüenciais sejam adotadas. Outro autores como DEWITT et al (1996), continuam considerando as vantagens do modelo de memória distribuída (expansibilidade, custos mais reduzidos) adotando-o no SGBDOO SHORE, sem no entanto apresentarem resultados em situações desfavoráveis ao paralelismo.

As consultas em SGBDOOs tendem a envolver uma navegação entre classes("path expressions"). Desta forma, uma estratégia efetiva para distribuição de objetos pelos diversos nós em um ambiente de memória distribuída é crucial para que se obtenha um bom desempenho(LIMA, 1996, BAIÀO, MATTOSO, 1996). Uma vez iniciada em um nó, a consulta deve percorrer os relacionamentos preferencialmente no âmbito do próprio nó. Do contrário, o acesso à objetos residentes em outros nós implicará em intercomunicação entre os diversos processadores que podem comprometer o ganho com o paralelismo. Portanto a estratégia de distribuição deve procurar reduzir a possibilidade de intercomunicação entre os nós, mas ao mesmo tempo, permitir que sejam exploradas as diferentes formas de paralelismo no processamento da consulta.

Entretanto, existe um conflito entre estes dois objetivos. Se por um lado a redução de comunicação entre os diversos processadores favorece que se tenha a base de dados distribuída por um pequeno número de nós, alcançar um alto grau de paralelismo favorece a distribuição da base por um número grande de processadores.

Algumas propostas para distribuição de objetos em SPBD são encontradas na literatura. GHANDEHARIZADEH et al (1994) em seu trabalho, apresentam um algoritmo voltado para arquiteturas de memória distribuída e para aplicações com predominância de consultas com navegação. O seu objetivo é distribuir horizontalmente os objetos do banco tentando minimizar o número de referências entre objetos em nós distintos e ao mesmo tempo obter um bom balanceamento de carga dos objetos entre os processadores. Como forma de se reduzir o número de troca de mensagens entre os nós, duas estratégias de replicação são utilizadas. Como esta distribuição não leva em conta os valores de atributos, quando da execução da consulta, não se consegue tirar proveito da localização dos objetos sendo portanto necessária a sua execução em todos os nós.

THAKORE e SU (1994) em seu trabalho propõem a distribuição de objetos segundo a fragmentação vertical. A fragmentação vertical permite que sejam recuperados somente os atributos pertinentes à consulta evitando-se a recuperação de atributos desnecessários e também que diferentes partições de uma classe sejam recuperadas em paralelo por várias consultas ao mesmo tempo. Entretanto, caso a consulta envolva a recuperação de atributos alocados em diferentes fragmentos, torna-se necessária a realização de uma junção para se obter o resultado.

Um outro método é apresentado por CHEN e SU(1996) também voltado para máquinas com arquitetura de memória distribuída visando os mesmos objetivos: obtenção de um bom balanceamento de carga entre os nós e redução de custos de comunicação entre eles. No modelo apresentado, a unidade de distribuição é a classe, ou seja, todos os objetos de uma classe devem residir em um único nó. A distribuição é baseada nos requisitos de processamento para a aplicação que é caracterizado pelo conjunto de consultas presentes na mesma.

Se por um lado a estratégia de distribuição da classe como um todo favorece as execuções direcionadas, por outro lado não se consegue tirar proveito do processamento em paralelo em consultas por extensão de classe. Além do mais, caso a população de objetos de uma classe seja grande, fragmentar a classe e ter vários processadores recuperando fragmentos pequenos é mais vantajoso do que ter somente um processador para processar todos os objetos.

BIBLIOGRAFIA

AMORIM,C.L. et al, 1991, *The NCP I Parallel Computer System*, Relatório Técnico ES-241/1991, COPPE-SISTEMAS/UFRJ

BAIÃO,F.,MATTOSO,M., 1996, *Uma Estratégia Abrangente para Projeto de Distribuição de Bases de Dados Orientados a Objetos*, Relatório Técnico ES-402/96, COPPE-SISTEMAS/UFRJ.

BARU, C.K. et al., 1995, "DB2 Parallel Edition", *IBM Systems Journal*, vol 34, No 2., pp. 292-322.

BOGLE,P.L., 1994, *A Safe, Efficient Object Database Interface Using Batched Futures*, Ph.D. dissertation, Massachusetts Institute of Technology, Massachusetts, USA.

BORAL,H. et al., 1990, "Prototyping Bubba, A Highly Parallel Database System", *IEEE Transactions on Knowledge and Data Engineering*, v.2(1), março, pp.4-24.

BOUGANIM,L., 1996, "*Data Placement for Parallel and Distributed Queries*", IDEA.DD.24B.002,INRIA,França, março 1996.

BOUGANIM,L., Dageville,B., Valduriez,P.,1996, "Adaptative Paralllel Query Execution in DBS3", In*: International Conference on EDBT* ,Avignon, França.

CAREY,J., et al., 1993, "The 007 Benchmark", In: *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data*, vol. 22,n.2,Washington DC, pp.12-21.

CAREY,J., et al., 1994, "A Status Report on The 007 OODBMS Benchmarking Effort", In: *Proceedings of the 9 Annual Conference on Object-Oriented Programming Systems, Languages and Applications-OOPLSA*, vol.29,n.10,pp.414-426, outubro.

CAREY,J., et al., 1994, "Shoring up Persistent Aplications", In: *Proceedings of the International Conference on Management of Data*, Minneapolis, maio 1994,pp.383-394.

CATTEL,R.G.G., 1994, "*Object Data Management*", Addison-Wesley Publishing Company, Revised Ed.

CERI,S. PELAGATTI,G., 1984, "*Distributed Databases : Principles and Systems*", McGraw Hill, 1984.

CHEN,YAW., SU,STANLEY., 1996, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object-Oriented Databases", *Distributed and Parallel Databases*, volume 4, n.2, Abril, pp. 107-141.

COOK, J., et al, 1996, "Semi-Automatic, Self-Adaptative Control of Garbage Collection Rates in Object Databases", In: *Proceedings International SIGMOD*, Montreal, Canadá, junho, pp. 377-388.

COPELAND,G. et al, 1988, "Data Placement in Bubba", In: *Proceedings ACM SIGMOD International Conference on Management of Data*, Chicago, Junho, pp 99-108.

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

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

DEWITT,D. et al, 1990b, "The GAMMA Database Machine Project", *IEEE Transactions on Knowledge and Data Engineering*, Vol 2, Março, pp 44-62.

DEWITT,D.,GRAY,J., 1992, "Parallel Database Systems: The Future of High Performance Database Systems", *COMMUNICATIONS OF THE ACM*, Vol. 35,No 6, Junho, pp 85-98.

DEWITT,D. et al, 1996, "Parallelizing OODBMS Traversals: a performance evaluation" *VLDB Journal* v(5)(1), pp. 3-18.

DEWITT,D., 1996, "Parallel Object-Relational Database Systems: Challenges and Opportunities", *Parallel and Distributed Information Systems*.

EZEIFE,C.I. BARKER,K., 1995, "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", *Distributed and Parallel Databases*,Volume 3, Number 3, July, pp. 247-271.

GHANDEHARIZADEH,S. et al, 1994, "Object Placement in Parallel Object-Oriented Database Systems", In: *Proceedings of the 10th International Conference on Data Engineering*, Houston, Texas, fevereiro, pp. 253-262.

GEIST,A., et al, 1994, *PVM 3 USER's GUIDE AND REFERENCE MANUAL*, ORNL/TM-12187, Oak Ridge National Laboratory, Tennessee, EUA.

GOA, 1996, "*GOA - Um Servidor de Objetos Persistentes*",
URL: http://www.cos.ufrj.br/~goa.

GRAEFE,G., 1990, "Encapsulation of Parallelism in the Volcano Query Processing System", In: *Proceedings ACM SIGMOD International Conference on Management of Data*, Atlantic City, EUA, maio, pp 102-111.

GRAEFE,G., 1993 "Query Evaluation Techniques for Large Databases", *ACM Computing Surveys*, vol. 25, No2, Junho, pp 73-170.

HONG,W., 1992, "*Parallel Query Processing Using Shared Memory Multiprocessors and Disk Array*", Ph.D. dissertation, University of California at Berkeley, Berkeley, Califórnia, USA.

INFORMIX CORPORATION, 1995, "*Informix-Online Extended Parallel Server*"
URL: http://www.informix.com/informix/corpinfo/zines/whitpprs/wht3/6828whi.html.

KALAPALEM,K. NAVATHES.B. MORSI,M.M.A., 1994, "Issues in Distribution Design of Object-Oriented Databases", In: Ozsu, M., Dayal, U., Valduriez, P., *Distributed Object Management*, Morgan Kaufmann Publishers, pp. 148-164.

KIM,W., 1990, "Architecture of the ORION Next-Generation Database System", *IEEE Transactions on Knowledge and Data Engineering*", v.2(1), março, pp. 44-62.

LERNER,A.,LIFSCHITZ,S.,PLASTINO,A., 1996, *Análise de Custos, Desempenho e Balanceamento de Carga de Algoritmos de Junção em Ambientes com Memória Distribuída*, Relatório Técnico PUCRioInfMCC16/96,PUC, Rio de Janeiro.

LIMA,F., 1996, "*Distribuição em Sistemas Gerenciadores de Bases de Dados Orientados a Objetos: Um Estudo de Caso Usando O2",* Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

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

MATTOSO,M.L.Q., SOUZA, J.M., AMORIM, C.L., 1993, "PARGOA: Um Servidor Paralelo para Sistemas de Banco de Dados Orientados a Objetos", In: *Anais VI Simpósio Brasileiro de Arquitetura de Computadores*, Florianópolis, SC, Setembro.

MATTOSO,M.L.Q., 1994, "Sistemas de Banco de Dados Distribuídos e Paralelos", *Jornada de Atualização em Informática*, Caxambu, MG, julho.

MATTOSO,M.L.Q., SILVA,G.Z.1994, "Explorando o Processamento de Consultas no Servidor de Objetos GOA", In: *Anais 9 Simpósio Brasileiro de Banco de Dados*.

MEHTA, M., DEWITT,D., 1997, "Data Placement in Shared-Nothing Parallel Database Systems", VLDB Journal v(6), pp. 53-72.

MEYER,L.A .V.C, MATTOSO,M.L.Q., 1997, "Paralelismo em Banco de Dados com Memória Distribuída: Uma Análise de Desempenho no ParGOA-MD", In: *Anais 12 Simpósio Brasileiro de Banco de Dados*.

NCR COMPANY, 1996, "*Teradata Database System For Unix on The NCR WorldMark 5100*",URL: http:/www/attgis.com/product/teradata/docshtm/5100wp.htm

NORMAN,M. ZUREK,T. THANISCH,P., 1996, "Much Ado About Shared-Nothing", *SIGMOD RECORD*, vol.35, num. 3, setembro, pp. 16-21.

ORACLE CORPORATION, 1996a, "*Oracle7 Server Release 7.3 Datasheet*"
URL: http://www.oracle.com/products/oracle7/oracle7.3/html/oracle7.3_ds.html.

ORACLE CORPORATION, 1996b, "*Running Oracle In A Parallel Environment*"
URL: http://www.oracle.com/

OZSU,M. VALDURIEZ,P., 1991, "*Priciples of Distributed Database Systems*", Prentice-Hall, Califórnia.

OZSU,M. VALDURIEZ,P., 1994 "Distributed Data Management: Unsolved Problems and New Issues", In: T. Casavant, M. Singhal, *Readings in Distributed Computing Systems*, IEEE/CS Press, pp.512-544.

OZSU,M. VALDURIEZ,P., 1996, "Distributed and Parallel Database Systems", In: A.Tucker, *Handbook of Computer Science and Engineering*, CRC Press, pp.1093-1111.

O2 TECHNOLOGY, 1995, *The O2 System Administration Guide*, Version 4.5.

PIRES,P., MATTOSO,M., 1996, "Aspectos de Interoperabilidade na Arquitetura Heterogênea HIMPAR", In: *Anais XI Simpósio Brasileiro de Banco de Dados*, São Carlos, SP, outubro, pp. 43-57.

ROCHA, A.R.C., SOUZA,J.M., 1990, "A Heuristic Workstation for Software Development", In: *COMPEURO 90*, Tel Aviv, Israel, maio.

SCHNEIDER,D. DEWITT,D., 1989, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", In: *Proceedings of ACM-SIGMOD International, Conference on Management of Data*, Portland, Oregon, pp. 110-121.

SILVA,G., 1995, *Uma Experiência na avaliação de Sistemas Gerenciadores de Bancos de Dados Orientados para Objetos*, Relatório Técnico ES-333/95,
COPPE-SISTEMAS/UFRJ.

SOUZA,J., MATTOSO,M. XEXEO,G., 1996, *Heterogeneidade, Interoperabilidade, Paralelismo e aspectos semânticos em SGBDOOs*, Proc 52.3603/94-0, COPPE/UFRJ.

STONEBRAKER,M. et al, 1988, "The Design of XPRS", In: *Proceedings of International Conference on Very Large Data Bases*, Los Angeles, pp. 339-350.

SYBASE INC., 1996a, "*Sybase MPP*",
URL: http://www.sybase.com/products/system11/mppdata.html#technical_highlights.

SYBASE INC., 1996b, "*Sybase MPP: Parallel High Performance for Real Workloads*",
URL: http://www.sybase.com/Offerings/Whitepapers/mpp_Wpaper.html.

TANDEM C. INCORPORATED, 1996, "*NonStop SQL/MP*",
URL: http://www.tandem.com./INFOCTR/HTML/PROD_DES/NSSQLPD.html.

TAVARES,F.,SOARES,J., 1995,"*PARGOA 2-Análise de desempenho no paralelismo virtual*", Projeto Final de Curso,Instituto de Matemática,UFRJ,Rio de Janeiro, Brasil.

THAKORE,A . SU,S., 1994, "Performance Analysis of Parallel Object-Oriented Query Processing Algorithms", *Distributed and Parallel Databases*, vol.2, núm.1, Janeiro, pp. 59-99.

VALDURIEZ,P., 1993a, "Parallel Database Systems:open problems and new issues", *Distributed and Parallel Databases*,vol.1, num.2, abril, pp. 137-165.

VALDURIEZ,P., 1993b, "Parallel Database Systems: the case for shared-something ", In: *Proc. of 9th International Conference on Data Engineering*, Viena, Austria, 1993.

WHITE,D.,DEWITT,D., 1995, "Implementing Crash Recovery in QuickStore: A Performance Study", In: *Proceedings of 1995 ACM SIGMOD International Conference on Management of Data*, pp.187-198, maio.