FACULDADE DE IMPERATRIZ - FACIMP
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
BANCO DE DADOS II
PROFESSOR JORGE COSTA

BANCO DE DADOS PARALELOS

Eduardo Yuji Wada
Everson Santos Araújo
Matheus Santos Saraiva
Teylo Laundos Aguiar

Imperatriz-MA
2003

FACULDADE DE IMPERATRIZ - FACIMP
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
BANCO DE DADOS II
PROFESSOR JORGE COSTA

BANCO DE DADOS PARALELOS

Eduardo Yuji Wada
Everson Santos Araújo
Matheus Santos Saraiva
Teylo Laundos Aguiar

Trabalho apresentado à disciplina de Banco de Dados II, para obtenção de conhecimentos sobre sistemas de banco de dados, no curso de Sistemas de Informação da Faculdade de Imperatriz, ministrada pelo professor Jorge Costa.

Imperatriz-MA
2003

Este trabalho é dedicado a todos aqueles que deram a vida por um mundo melhor: livre do LAG, NetSplit e que lutaram pela conexão em Banda Larga. Amém.

ÍNDICE

P.

ABSTRACT

Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware. These new designs provide impressive speedup and scaleup when processing relational database queries. This paper reviews the techniques used by such systems, and surveys current commercial and research systems.

INTRODUÇÃO

Bancos de Dados altamente Paralelos estão começando a substituir os tradicionais Mainframes para processamento de banco de dados e transação. O sucesso destes sistemas reside em um artigo de 1983 que prediz o protocolo das máquinas de banco de dados. Há dez anos o futuro das máquinas de banco de dados paralelas pareceu inatingível. A maioria das pesquisas de máquina de banco de dados tinha focalizado em hardwares especializados tal como memórias de CCD, memórias de bolha (bubble memories), discos da cabeça-por-trilha, e discos óticos. Nenhuma destas tecnologias cumpriu suas promessas inteiramente; assim havia um sentimento - hoje realizado - de que os processadores centrais convencionais, a RAM eletrônica, e os discos magnéticos de movimentação de cabeça dominariam a cena por muitos anos. Nesse tempo, o barramento de acesso do disco foi predito para dobrar quando as velocidades do processador foram preditas para aumentar por fatores muito maiores. Conseqüentemente, os críticos predisseram que os sistemas de multiprocessadores seriam logo limitados por sistemas de E/S a menos que uma solução ao gargalo de E/S fosse encontrada.

1. Histórico

Apesar destas predições terem sido bem apuradas sobre o futuro do hardware, os críticos estavam certamente enganados sobre o futuro do sistema de banco de dados paralelo. Nas décadas de 70 e 80, Teradata, Tandem, e várias outras empresas obtiveram sucesso criando e vendendo sistemas altamente paralelos.

A adoção do sistema relacional de dados é uma das explicações para o sucesso de sistemas de banco de dados paralelo, buscas relacionadas são criadas pensando no paralelismo; consiste de operações uniformes aplicadas a recepção uniforme de dados. Cada operador produz uma nova relação, então os operadores podem ser decompostos em requisições paralelas. Através do envio da resposta de saída de um operador na entrada de requisição de outro operador, os dois operadores podem trabalhar em séries através do paralelismo de pipeline. Particionando a requisição em vários processadores e memórias, um operador pode também ser dividido em vários operadores independentes, cada um tratando parte da requisição. Esse particionamento de dados e execução é a base do paralelismo particionado.

O sistema de troca de dados para criação de sistemas de banco de dados precisa de um sistema operacional baseado em troca de mensagens entre cliente e servidor, para inter-relacionar os processos paralelos executando as operações relacionais. Essa arquitetura também depende de um rede de alta-velocidade para conectar os processadores, configuração essa que atualmente se tornou a base para os PCs. Esse sistema cliente-servidor é uma excelente base para a tecnologia de sistemas de banco de dados distribuídos.

Os criadores de mainframes encontraram dificuldade em criar máquinas com capacidade suficiente para garantir a demanda dos bancos de dados relacionais servindo um grande número de usuários ou buscando bancos de dados com terabytes de informação. Enquanto isso, sistemas multiprocessados baseados em microprocessadores rápidos e baratos ficaram disponíveis através de empresas como a Encore, Intel, NCR, nCUBE, Sequent, Tandem, Teradata e Thinking Machines. Essas máquinas provinham mais poder de processamento total do que seus mainframes concorrentes a um preço mais baixo. Sua arquitetura modular permitiu aos sistemas crescerem acrescentando-se memória e discos para facilitar o processamento de um trabalho qualquer em paralelo.

Surge então outra arquitetura baseada em share-nothing (compartilhar nada), na qual cada processador se comunica com os outros apenas enviando mensagens através de uma rede interconectada. Nesse tipo de sistemas, as tuplas de cada relacionamento no banco de dados são particionados através dos discos diretamente ligados a cada um dos processadores. O particionamento permite que vários processadores varram grandes relações em paralelo sem necessitar de nenhum sistema de E/S exótico. Está arquitetura surgiu pioneiramente na Teradata no final dos anos 70 e também em inúmeros projetos de pesquisa.

2. Objetivos e métricas do Paralelismo: Velocidade e escalabilidade

O sistema paralelo ideal demonstra duas propriedades chave que são a velocidade e a escalabilidade linear.

1. Velocidade linear: O dobro de hardware pode realizar a mesma tarefa em metade do tempo necessário.
2. Escalabilidade linear: O dobro de hardware pode realizar o dobro de tarefas no mesmo tempo de realização de metade dela.

Formalmente, dado um trabalho fixo, roda-o em um pequeno sistema, e depois roda-o em um grande sistema, a velocidade gerada pelo grande sistema é mensurada por:

$$\text{Velocidade} = \frac{\text{Tempo do sistema pequeno}}{\text{Tempo do sistema grande}}$$

A velocidade se diz linear, se um trabalho N-vezes maior ou mais pesado é detonado por uma velocidade de N. O sistema de velocidade mantém o tamanho do problema do constante, e aumenta o tamanho do sistema. A escalabilidade é definida como a habilidade de em um sistema N-vezes maior realizar um trabalho N-vezes maior resultando no mesmo tempo que o sistema original obteve. A métrica da escalabilidade é:

$$\text{Escalabilidade} = \frac{\text{Tempo do sistema pequeno em um trabalho pequeno}}{\text{Tempo do sistema grande em um trabalho grande}}$$

Se a equação da escalabilidade retornar o valor 1 (um), então a escalabilidade é dada como linear. Existem dois tipos distinto de escala, em grupo ou transacional. Se o trabalho consiste em realizar várias requisições pequenas e independentes requeridas por vários clientes e operadores em um sistema compartilhado, então a escalabilidade consiste em N-vezes a quantidade de clientes, requisitando N-vezes determinada informação em um sistema N-vezes maior. Essa é a escalabilidade tipicamente encontrada em sistemas de processamento transacional e de tempo compartilhado. Essa formula de escalabilidade é utilizada pelo Conselho de Performance em Processamentos Transacionais para mensurar o nível de seus processos transacionais. Conseqüentemente, é chamada de escala-transacional (*transaction-scaleup*). A escalabilidade transacional é idealizada para sistemas paralelos visto que cada transação é um típico pequeno trabalho independente que pode ser realizado em processadores separados.

Uma segunda forma de escalabilidade, chamada de grupo, surge quando o que deve ser realizado é um único e grande trabalho. Esse é o típico sistema de requisição de banco de dados e também típico de simulações cientificas. Nesses casos, a escalabilidade consiste em usar um sistema N-vezes maior para resolver um problema N-vezes maior. Para sistemas de banco de dados o sistema de grupo deve realizar a mesma requisição em um banco de dados N-vezes maior; para resolução de problemas científicos, a escalabilidade em grupo deve realizar o mesmo calculo em uma simulação N-vezes maior.

3. Arquiteturas de Paralelismo em Bancos de Dados.

O sistema de banco de dados ideal deveria ter um único e infinitamente rápido processador com uma infinita memória – e seria infinitamente barato (de graça). Dado essa máquina não haveria necessidade de aumento de velocidade, escalabilidade, ou paralelismo. Infelizmente, a tecnologia não está criando tal máquina – mas está chegando perto.

Então o desafio é criar um processador infinitamente rápido através de infinitos processadores de velocidade finita, e criar uma memória infinitamente grande com infinita área de troca de infinitas memórias de velocidade e armazenamento finito. Isso soa trivial matematicamente; mas na prática quando um novo processador é adicionado à maioria das implementações de computadores, ele diminui um pouco da velocidade de todos os outros processadores. Se essa interferência é de 1%, um sistema de mil processadores teria 4% do poder efetivo de um sistema com um único processador de mesma velocidade.

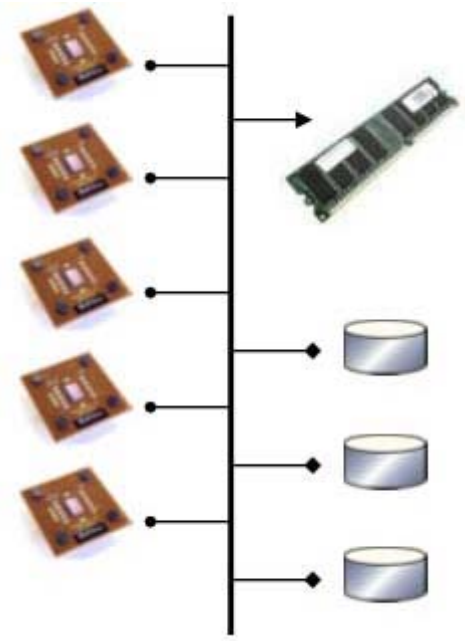
Stonebraker sugere as seguintes taxonomias de design para sistemas paralelos:

- Memória compartilhada: Todos os processadores compartilham a mesma memória e os mesmos discos.
- Discos compartilhados: Cada processador tem uma memória própria, mas tem acesso a todos os discos.
- Sem compartilhamento: Cada memória e disco são próprios de um processador que atua como servidor dos dados que possui.
- Hierárquico: Cada nó pode ser considerado como um sistema independente.

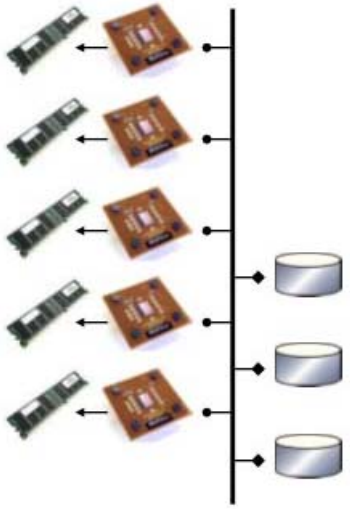
3.1. Memória Compartilhada

Os processadores e os discos acessam uma memória em comum, normalmente, por meio de cabo ou por meio de rede de interconexão

- Vantagem: extrema eficiência na comunicação entre processadores
- Desvantagem: a arquitetura não é adequada ao uso de mais de 32 ou 64 processadores
  - Exemplos: multiprocessadores simétricos (Sequent, Encore) e alguns mainframes (IBM3090, Bull's DPS8)



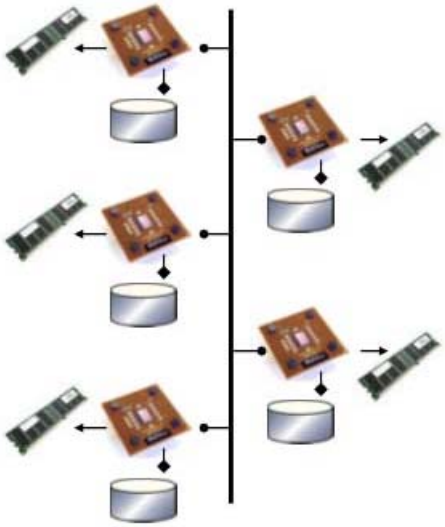
3.2. Discos Compartilhados



Todos os processadores podem acessar diretamente os discos através de uma rede de conexão, mas cada processador possui uma memória privada.

- Vantagens: o acesso à memória não representa um gargalo; é um modo barato de aumentar a tolerância a falhas
- Desvantagem: é novamente o grau de crescimento
  - Exemplos: IBM Sysplex e Digital VAXclusters rodando Rdb (Oracle Rdb)

3.3. Sem Compartilhamento

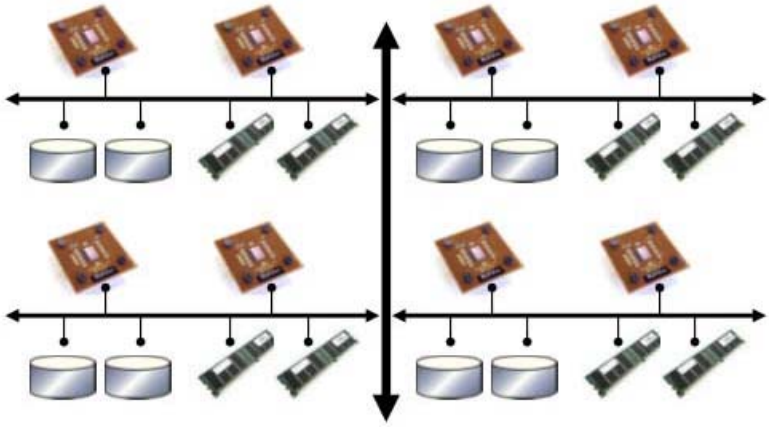


Cada equipamento de um nó consiste em um processador, uma memória e discos.

- Vantagem: suporte a um grande número de processadores
- Desvantagem: comunicação entre processadores é o fator limitante, devido a necessidade de acesso a dados não locais
  - Exemplos: nCUBETeradata's DBC, Tandem, Intel's Paragon, NCR's 3600 e 3700

3.4. Hierárquico

Este modelo combina características de várias arquiteturas anteriores, reduzindo a necessidade e complexidade da comunicação entre processadores.



4. Implementando o Paralelismo em Bancos de Dados.

O paralelismo pode ser utilizado de 3 (três) formas:

- Na entrada e saída de dados (E/S)
- No processamento de consultas
- No processamento de operações individuais

Aumentando a escala e o desempenho do sistema, oferecendo um maior e mais rápido processamento das transações.

Entretanto, torna o sistema mais exigente quanto ao hardware, podendo ocasionar uma maior quantidade de falhas. Uma solução para tal contratempo seria a replicação dos dados, para um controle mais eficaz da consistência.

4.1. Paralelismo na entrada e saída de dados (E/S)

O paralelismo de E/S tenta reduzir o tempo necessário para recuperar relações do disco por meio do particionamento dessas relações em múltiplos discos.

Nesta forma as operações podem ser executadas em paralelo se cada uma acessar um dispositivo, assim sendo, cada processador pode trabalhar com os dados de uma partição. Um plano de execução em paralelo pode ser criado ao otimizar a operação

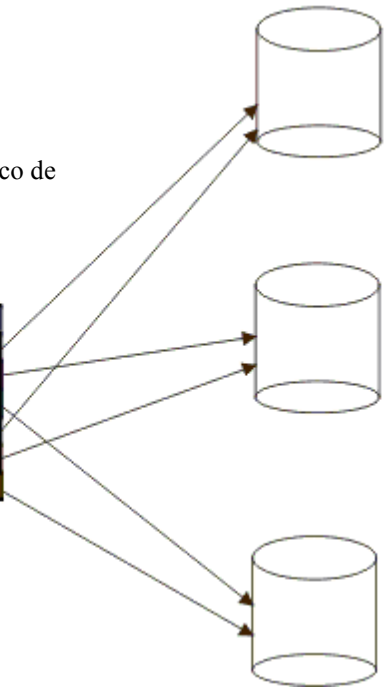
Existem três tipos de particionamento de dados para obtenção de paralelismo de E/S:

- Horizontal
- Vertical
- Misto

4.1.1. Particionamento Horizontal

A forma mais comum de particionamento de dados em um ambiente de banco de dados é o particionamento horizontal:

Nombre	Domocilio	Telefono	Edad	Sexo
Pedro	col. Machado c. 5 de mayo # 11	6488393	33	M
Sandra	col. Morelos c. guadalajara #1343	66268485	21	F
Alejandra	col. Lucio Blanco c. Guillermo troncoso #12	43656886	35	F
Luis	col. Obrera c. corchos #123	48885346	15	M
Jose	fracc. El Lago c. buena vista # 564	6673255	20	M
Elvira	col. Constitusion c. Durango # 65	7543463	42	F



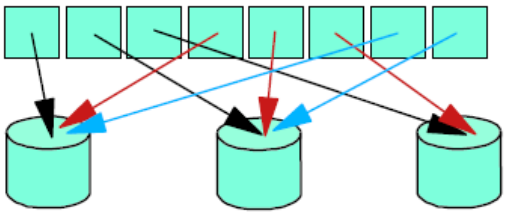
As tuplas de uma relação são divididas entre muitos discos, tal que cada tupla resida em um disco diferente.

4.1.1.1. Técnicas de Particionamento

Existem três estratégias básicas de particionamento, para uma melhor implementação do "trabalho" a ser executado. São elas:

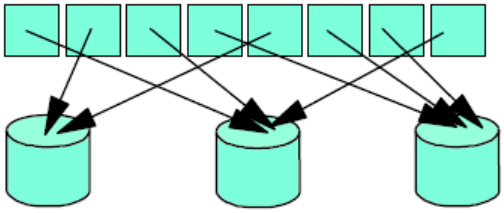
Considerando ***n*** discos, D_0, D_1, D_2, ..., D_{n-1}, entre os quais os dados devem ser particionados, explicaremos as estratégias.

Round-Robin (circular)



- A relação é percorrida em qualquer ordem e a i-ésima tupla é enviada ao disco numerado como $D_{i \bmod n}$.
- Cada nova tupla é colocada em um dispositivo diferente, distribuindo uniformemente entre os discos.

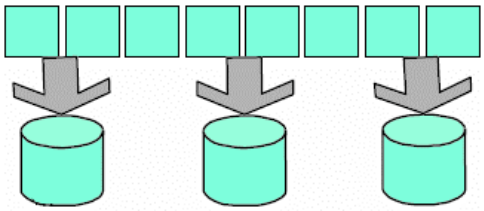
Particionamento *Hash*



- Um ou mais atributos do esquema de relação dado são designados como atributos de particionamento.
- Uma função *Hash* é escolhida em uma faixa entre { 0, 1, ..., n-1 }.
- Cada tupla da relação original é separada pelo atributo de particionamento. Se a função *Hash* retorna ***i***, então a tupla é alocada no disco D_i.

Particionamento por faixa.

- Distribui faixas contíguas do valor de um atributo para cada disco.
- Um atributo de particionamento ***A*** é escolhido como um vetor de particionamento.
- Seja a seqüência $[v_0,v_1,...,v_{n-2}]$ denotando o vetor de particionamento, tal que, se $i<j$, então $v_i<v_j$.
    - Considere uma tupla ***t***, tal que t[A] = x.
    - Se $x<v_0$, então ***t*** é colocada no disco D_0.
    - Se $x = v_{n-2}$ então ***t*** é colocada no disco D_{n-1}.
    - Se $vi = x < v_{i+1,}$ então ***t*** é colocada no disco D_{i+1}.



4.1.1.2. Comparação de Técnicas de Particionamento

Uma vez particionada a relação, podemos recuperá-la usando vários tipos de acesso aos dados:

- Percorrer a relação inteira
- Localizar uma tupla associativamente (por exemplo, nome_empregado = "João")
- Localizar todas as tuplas, tal que o valor de um dado atributo permaneça entre uma faixa especificada (por exemplo, 10000 < salário < 20000)

Round-Robin

- É ideal para aplicações que precisam ler a relação inteira, seqüencialmente, em cada consulta. Entretanto, tanto consultas pontuais, como por faixas têm processamento complexo, já que cada um dos n discos precisará participar da busca.

Hash

- É mais adequado a consultas pontuais baseadas no atributo de particionamento.
- É útil para varreduras seqüenciais em uma relação inteira. O número de tuplas em cada um dos discos é aproximadamente o mesmo, sem muita variação, portanto, o tempo usado para percorrer a relação é 1/***n*** do tempo necessário para percorrer a relação em único disco.
- Não é muito adequado para consultas pontuais sobre demais atributos de não-particionamento.
- Também não é muito adequado para responder consultas sobre faixas de dados, já que normalmente as funções *Hash* não preservam proximidade entre as faixas.

Por Faixa

- É bastante adequado para consultas pontuais e por faixas sobre atributos de particionamento.
    - Um ou poucos discos precisam ser utilizados
    - Os outros discos ficam livres para outros acessos
    - Eficiente se as tuplas do resultado estiverem em poucos blocos de disco
    - Se muitos blocos precisarem ser lidos, há desperdício de paralelismo pois poucos discos serão utilizados
- Em buscas pontuais recorremos ao vetor de particionamento para localizar o disco no qual a tupla reside
- Para consultas por faixas, recorremos ao vetor de particionamento a fim de encontrar a faixa de disco na qual as tuplas podem residir. Em ambos os casos, reduz-se a busca a exatamente aqueles discos que podem ter quaisquer tuplas de interesse

4.1.1.3. Desbalanceamento

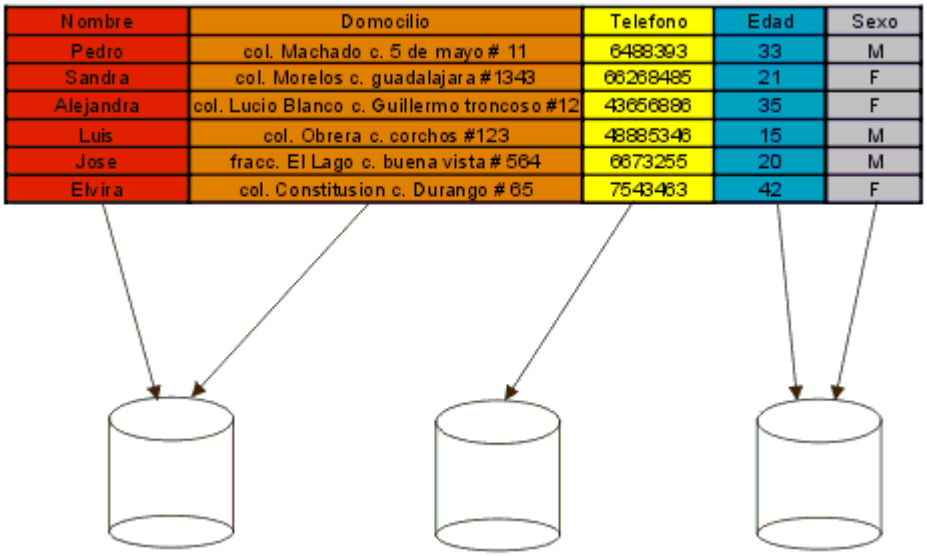
Quando uma alta porcentagem de tuplas são colocadas em algumas partições e poucas tuplas são colocadas nas restantes, ocorre o que chamamos de desbalanceamento.

Formas de desbalanceamento:

- Desbalanceamento de valor de atributo
    - Muitas tuplas possuem valores idênticos para o atributo de particionamento
- Desbalanceamento de partição
    - Os critérios de particionamento concentram muitas tuplas em poucas partições
    - Menos provável no *hash* do que no particionamento por faixas, se a função de *hash* for boa o suficiente

4.1.2. Particionamento Vertical

Os campos de uma relação se particionam entre os discos, onde cada campo reside em um ou mais discos. A partição vertical de uma relação $\boldsymbol{R}$ produz $\boldsymbol{R}_1$, $\boldsymbol{R}_2$, ..., $\boldsymbol{R}_R$, cada um dos quais é um subconjunto dos atributos de $\boldsymbol{R}$. O objetivo consiste em dividir a relação em um conjunto de relações menores que a original, de tal forma que se minimize o tempo de execução das aplicações que implementam esses fragmentos.



A partição vertical resulta ser mais difícil que a horizontal devido a necessidade de análises estatísticas sobre os acessos realizados a cada atributo da relação. Deve-se auxiliar replicando as chaves primárias da relação para poder reconstruir a relação original

4.1.3. Particionamento Misto

Em alguns casos o uso do particionamento horizontal ou vertical não se apresenta satisfatoriamente para a aplicações que acessão a base de dados, de tal forma que se faz necessário o uso da partição mista.

Existem dois tipos de particionamento misto:

- HV: Faz-se uma partição vertical, e posteriormente, sobre os fragmentos resultantes se aplica uma partição horizontal.
- VH: Faz-se uma partição vertical sobre os conjuntos de tuplas resultantes da aplicação de uma partição horizontal.

Nesse tipo de partição necessitaremos dados quantitativos sobre a base de dados, as aplicações que funcionam sobre ela, a rede de comunicações, as características de processo e o limite de armazenamento de cada local da rede.

4.2. Paralelismo em Consultas

O processamento de consultas em arquiteturas paralelas deve considerar o tipo de compartilhamento de discos e memória existente e a fragmentação e alocação de dados empregada .

Há dois tipos de paralelismo possível para o processamento de consultas em um banco de dados paralelos:

- Inter-consultas
- Intra-consultas

4.2.1. Inter-consultas

Esta é a alternativa mais simples. Cada consulta submetida pelo usuário é executada totalmente em um único processador. O tempo de processamento de uma certa consulta é idêntico ao tempo em um servidor monoprocessado, pois a consulta é monoprocessada. Por conseqüência, os processadores devem ser de grande capacidade, para não inviabilizar o sistema. A vantagem é que o gerenciamento de tarefas é bastante simples.

Consultas ou transações diferentes são executadas em paralelo umas com as outras. A principal aplicação do paralelismo inter-consultas é melhorar o sistema de processamento de transações processadas por segundo. Os processadores têm de realizar algumas tarefas como bloqueio e log (registro diário), de forma coordenada e isso exigem que troquem mensagens entre si, além de assegurar que dois processadores não atualizem o mesmo dado, de modo independente, ao mesmo tempo.

Quando um processador acessa ou atualiza dados, o sistema precisa garantir que o processador tenha a última versão dos dados em sua área de buffer. Esse último problema é conhecido como problema de coerência de *cache*. Vários protocolos têm sido desenvolvidos para garantir a coerência de *cache*;

Regras do Protocolo:

- Antes de qualquer acesso para leitura ou escrita de uma página, uma transação bloqueia a página no modo exclusivo ou compartilhado, conforme apropriado. Imediatamente após a transação obter esse bloqueio, ela lê a cópia mais recente da página no disco compartilhado.
- Antes de uma transação liberar um bloqueio exclusivo em uma página, ela descarrega a página no disco compartilhado; só depois libera o bloqueio.

4.2.2. Intra-consultas

Neste modo, partes de uma consulta é executada em paralelo nos diversos processadores e discos, o que diminui o tempo de resposta das consultas.

Planos de execução são feitos na forma de árvores, e cada ramo pode ser processado em paralelo. Pode ser feito uso de *Pipelining* entre as operações, assim, a saída de uma operação é a entrada da outra. A memória e os discos compartilhados servem para troca de dados entre processadores caso não seja possível fazer o *pipelining*

4.3. Paralelismo em Operações

4.3.1. Formas Paralelismo em Operações

Há duas formas de paralelismo quem podem ser utilizadas no processamento de Operações em um banco de dados paralelos:

- Paralelismo inter-operação: as operações de uma consulta são executadas em paralelo
- Paralelismo intra-operação: uma operação é dividida em várias partes, sendo cada uma delas executada por um processador

As duas formas de paralelismo podem ser usadas simultaneamente por um SBD paralelo.

4.3.1.1. Inter-operação

Considerando que uma consulta do usuário é dividida em várias operações mais simples, a alocação em processadores é definida para cada operação. Ou seja, uma determinada operação de uma consulta pode ser realizada no processador 1 e outra operação, da mesma consulta, no processador 2, em paralelo. Deste modo, o tempo de resposta para uma determinada consulta pode ser reduzido. O algoritmo de alocação de operações em processadores deve atribuir operações relacionadas, onde uma produz um resultado parcial que é entrada para a outra operação no mesmo processador, sempre que possível, para reduzir a comunicação entre os processadores.

Escalas de execução podem prever a avaliação de operações em paralelo, nesse tipo de operação cada consulta é dividida em operações simples, que são alocadas para processadores individuais. Cabe ao otimizador de consultas determinar as operações que serão executadas em paralelo. O paralelismo é usado se o ganho de processamento for maior que o custo de comunicação.

As escalas possíveis são ainda mais numerosas que nas execuções seqüenciais: usar heurística (ex.: escolher a escala seqüencial mais eficiente e paralelizar), para tal é necessário alocar os recursos – processador, memória e disco – que serão usados por cada operação executada em paralelo.

4.3.1.1.1. Formas de Paralelismo de Inter-operação

O paralelismo inter-operação pode ser implementado de duas formas:

- Paralelismo Independente:
    - Operações de uma consulta não dependem necessariamente uma da outra

- Operações independentes são executadas em paralelo por processadores diferentes

- Paralelismo Pipeline:
  - Algumas operações não são independentes
  - As tuplas produzidas por uma operação vão sendo passadas para as operações que precisam delas
  - Cada operação é executada por um processador

4.3.1.2. Intra-operação

Os algoritmos de cada operação podem ser paralelizados, permitindo que uma operação complexa, como a junção espacial, seja realizada utilizando vários processadores. O tempo de resposta de uma operação, desta forma, é reduzido. A dificuldade deste tipo de paralelismo reside na necessidade de desenvolver novos algoritmos e otimizá-los para cada tipo de arquitetura e particionamento de dados utilizado.

4.3.1.2.1. Implementações na Intra-operação

Algumas implementações utilizadas no paralelismo de Intra-operações são:

- Seleção Paralela:
  - Cada processador procura na sua partição dos dados as tuplas nas quais a condição de seleção é válida. Este pode ser simplificado, se a tabela for particionada por faixa ou *hash*, uma vez que não é preciso usar todos os processadores.

- Classificação em paralelo:
  - Algoritmo de classificação por faixas
    - Particionar a tabela por faixas com base no atributo de classificação
    - Ordenar cada partição independentemente
  - Algoritmo de *sort-merge* externo paralelo
    - Supondo que a tabela já foi particionada em vários discos usando qualquer método
    - *Sort*: cada processador ordena uma partição
    - *Merge*: partições já em ordem são particionadas novamente por faixas, e são enviadas aos processadores, que as unem às outras partições recebidas e as classificam

- Eliminação de duplicatas em paralelo:
  - Pode ser feita na classificação das tuplas ou nas faixas ou *hashs* por cada processador

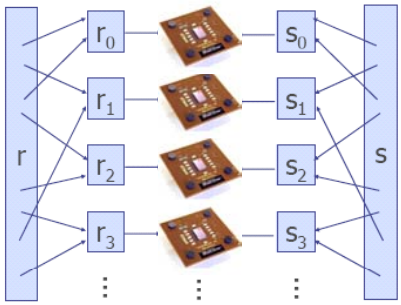
- Projeção paralela:
  - Efetuada pelos processadores à medida que as tuplas são lidas em paralelo dos discos

- Agregação:
  - Particionar em faixa ou *hash* a relação usando os atributos de agrupamento
  - Computar os valores agregados em cada processador

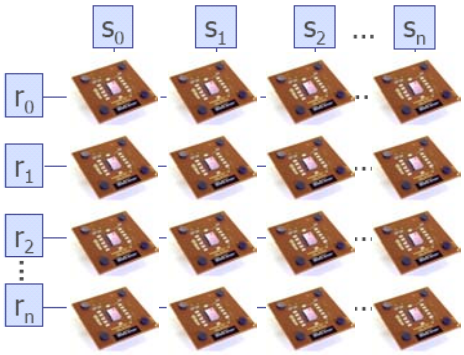
4.3.1.2.2. Junção Paralela

Na junção paralela algoritmos dividem entre os processadores os pares de tuplas a serem testados na junção. Assim, pares de tuplas oriundos de todos os processadores para os quais a condição de junção é válida são reunidos para se ter o resultado final da junção.

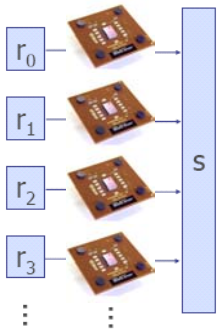
- Algoritmo de junção particionada
    - Usado em junções naturais e junções de igualdade
    - A mesma função de faixa ou *hash* deve ser usada para particionar as tabelas
    - Cada processador faz a junção das partições usando um algoritmo de junção seqüencial



- Algoritmo de junção por fragmentação e replicação
    - Usado para junções que não sejam de igualdade ou naturais
    - As duas tabelas são fragmentadas em m e n partições respectivamente
    - Cada processador executa a junção entre uma partição da primeira tabela e outra da segunda tabela, usando qualquer algoritmo de junção local
    - Ao final, as tuplas resultantes são reunidas para obter o resultado
    - Caso especial: junção assimétrica, onde ***n***=1



Junção Simétrica



Junção Assimétrica

- Algoritmo de junção hash paralela
    - Usa funções de hash para dividir as relações
    - Uma variante do algoritmo de junção hash calcula o resultado da junção

- Algoritmo de junção paralela de laço aninhado
    - Cada processador faz a junção indexada de laço aninhado da relação menor, replicada em todos os processadores, com uma partição da relação maior
    - Usa índice da relação maior no atributo de junção
    - É usado quando uma das relações é muito menor que a outra

4.3.1.2.3. Custo da Avaliação Paralela de Operações

- Particionamento de E/S entre diversos discos
- Particionamento de CPU entre diversos processadores
- Custos de inicialização em diversos processadores
- Desbalanceamento da distribuição do trabalho entre os diversos processadores
- Retenção de recursos resultando em atrasos
- Custo de montagem do resultado final, devido à transmissão de resultados parciais a partir de cada processador

5. Projeto de Sistemas Paralelos

- Paralelização do armazenamento de dados
- Paralelização do processamento de consultas
- Carregamento paralelo de dados a partir de fontes externas
- Resistência à falha de alguns processadores ou discos
- Reorganização on-line de dados e troca de esquemas

Conclusão

Este trabalho procurou descrever as técnicas de fragmentação e alocação de dados, e de processamento e otimização de consultas em um Banco de Dados Paralelos.

Sistemas de banco de dados paralelos existem para facilitar a exploração de inúmeros hardwares interligados em uma ou varias maquinas, compartilhando assim os dados e processamentos entre estes.

Um sistema de banco de dados paralelos é a resposta para a necessidade de implementação de grandes bancos de dados, com grande número de acesso, sem necessitar de maquinas que tenham grande poder computacional ou memória para isto.

Nenhum autor abordou, em profundidade, a questão das demais formas de particionamento em Paralelismo (E/S), salvo o Particionamento Horizontal. É aceitável supor que o desempenho destes sejam muito inferiores ao referenciado, entre tanto, é importante o conhecimento de todos. Um trabalho futuro interessante é buscar estatísticas que comprovem esta suposição.

Concluindo, o trabalho fora de grande importância para uma compreensão mais profunda sobre Bancos de Dados Paralelos.

Bibliografia

BORAL, H. and DEWITT, D. "**Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines**." Proceedings of the 1983 Workshop on Database Machines, edited by H.-O. Leilich and M. Missikoff, Springer-Verlag, 1983.

The Performance Handbook for Database and Transaction Processing Systems, J. Gray editor. Morgan Kaufmann, San Mateo. 1991.

STONEBRAKER, M., "**The Case for Shared Nothing**," Database Engineering, Vol. 9, No. 1, 1986.

Parallel Database Systems: The Future of High Performance Database Processing. Appeared in Communications of the ACM, Vol. 36, No. 6, June 1992

FORNARI, Miguel Rodrigues. **Sistemas Gerenciadores de Bancos de Dados Geográficos Distribuídos e Paralelos**. – Porto Alegre: PGCC da UFRGS, 2002.

HWANG, K. **Advanced Computer Architecture: Parallelism, Scalability, Programmability**. New York: McGraw-Hill. 1993. 770 pp.

SILBERSCHATZ, A.; KORTH, H.F. SUDARSHAN, **S. Sistemas de Bancos de Dados**, 3ª ed. São Paulo: Makron Books. 1999. 779 pp.