ANÁLISE E AVALIAÇÃO DE TÉCNICAS EXISTENTES PARA DISTRIBUIÇÃO DE DADOS E DIRETÓRIOS PROCESSAMENTO DE CONSULTAS E CONCORRÊNCIA EM SISTEMAS DE GERENCIAMENTO DE BANCOS DE DADOS DISTRIBUÍDOS.

Hugo Fernando Spencer Barrenechea

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS GRADUAÇÃO DE ENGENHARIA DE SISTEMAS DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DO MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:

LUIZ ANTONIO C. C. COUCEIRO
(Presidente)

SUELI MENDES DOS SANTOS

JAYME LUIZ SZWARCFITER

RIO DE JANEIRO - RJ - BRASIL

MAIO DE 1983

SPENCER, BARRENECHEA HUGO FERNANDO

Análise e Avaliação de Técnicas existentes para distribuição de dados e diretórios, processamento de consultas e concorrência em sistemas de gerenciamento de Bancos de Dados Distribuídos Rio de Janeiro 1983.

XII, 155, 29,7 cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas de Computação, 1983).

Tese - Universidade Federal do Rio de Janeiro - Faculdade de Engenharia

1. Bancos de Dados Distribuídos I. COPPE/UFRJ II. Título (série).

À minha mãe Zoyla

Aos meus saudosos irmãos:

Luz

Willy/Nancy

Elsa

Ana

Carlos Jré

À minha tia Elsa

Aos meus colegas da COPPE-Sistemas

A G R A D E C I M E N T O S

Ao Dr. Luiz Antônio da Cunha Couceiro, pelas idéias, conhecimentos e paciente orientação ministrada durante o desenvolvimento deste trabalho.

A COPPE e ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil) pelos recursos fornecidos durante meus estudos de Mestrado.

A todos meus familiares e amigos que colaboraram e estimularam na execução e conclusão deste trabalho, em especial ao pessoal da biblioteca do Núcleo de Computação Eletrônica - UFRJ pela orientação bibliográfica e colaboração prestada.

Ao Dr. Julio Villanueva Lara e ao Programa de Computação da Universidad Nacional Mayor de San Marcos (LIMA-PERU) pelo apoio e incentivo constante por eles dados.

Aos demais membros da banca que muito honraram-me com sua participação.

À Dulce Maria Vilela pela boa vontade no trabalho de datilografia.

R E S U M O

Este trabalho avalia problemas relacionados com o desenvolvimento de projetos de Sistemas de Gerenciamento de Bancos de Dados Distribuídos (SGBDD) definindo e localizando métodos e modelos que hoje estão sendo adotados como os mais apropriados na solução desses problemas.

Discute-se os modelos matemáticos e heurísticos para a distribuição dos dados e diretórios, as estratégias de processamento de consultas e os algoritmos de controle de concorrência dentro de um SGBDD. Finalmente apresenta-se as conclusões desta pesquisa.

A B S T R A C T

This work evaluates the problems related to the design development of Distributed Data Base Management Systems (DDBMS) and identifies the methods and models currently in use as the most appropriate to the solutions of these problems.

We discuss heuristics and mathematical models for the Data and Directory Distributions, query processing strategies and concurrency control algorithms within a DDBMS.

Finally, the conclusions of this research are presented.

ÍNDICE

I - INTRODUÇÃO

Os avanços na tecnologia da microeletrônica, que levaram ao surgimento de mini e microcomputadores de baixo custo, estão causando uma profunda alteração na atividade de processamento de dados. Os microcomputadores estão ocupando, hoje em dia, uma boa porcentagem do mercado. Estes avanços permitem ao usuário dispor de capacidade de processamento local para muitas aplicações, fazendo que os sistemas distribuídos sejam uma das áreas de maior futuro tendo influência no desenvolvimento dos sistemas de informação.

Em decorrência desses fatos, surge o conceito de um tipo particular de sistema distribuído. O sistema de Gerênciamento de Bancos de Dados Distribuídos (SGBDD), que é definido como o software gerenciador de um banco de dados fisicamente distribuído sobre vários nós (i.e. mini, microcomputadores, etc.). Um SGBDD pode ser melhor visualizado como um conjunto de bancos de dados locais geograficamente dispersos, mas interligados por uma rede de comunicação que possibilita a transferência de dados entre os diversos bancos de dados locais de forma simples e transparente para os usuários.

Este tipo de sistemas tem aplicação prática nas organizações descentralizadas compostas por um número importante de subunidades tais como bancos, casas comerciais, universidades, etc. No lado teórico o interesse despertado pelo SGBDD tem causado um grande crescimento e diversificação de pesquisa na área, visando conceituar tais sistemas e estabelecer técnicas apropriadas para o melhor desempenho dos mesmos.

Paralelamente a esses estudos no campo teórico, diver

sas Universidades e Laboratórios de Pesquisa têm desenvolvido projetos protótipos tentando estabelecer uma base de conhecimento sólido para o desenvolvimento da área. Entre os projetos mais importantes em pesquisa e desenvolvimento temos: o projeto SIRIUS na França, os projetos DISCO, ESA, POREL e VDN na Alemanha e os projetos INGRES, R* (R estrela) e SDD-1 nos Estados Unidos.

Os principais objetivos desta tese são: 1º) Avaliar problemas relacionados ao desenvolvimento de projetos de SGBDD, 2º) Detalhar esses problemas e estabelecer o nível atual de conhecimento humano no assunto e 3º) Definir e localizar métodos e modelos que hoje estão sendo adotados como os mais apropriados na solução desses problemas.

Para atingir esses objetivos, será feita no Capítulo II uma conceituação dos sistemas de gerência de BDD, estabelecendo um conjunto de conceitos básicos que caracterizem e permitam a análise das principais características funcionais de tais sistemas.

No Capítulo III, analisa-se um dos problemas mais importantes dentro da área dos SGBDD: O problema de determinação dos locais de armazenamento dos arquivos e diretórios do BDD, de forma a se obter um menor custo operacional total levando-se em consideração as restrições impostas pela rede. Os métodos de solução são de dois tipos, métodos de programação matemática e métodos heurísticos.

No Capítulo IV trata-se sobre as estratégias de processamento de consultas. São consideradas duas estratégias: distribuída e centralizada e faz-se a análise dos algoritmos correspondentes apresentando-se os sistemas INGRES Distribuído

e SDD-1 como exemplos práticos de aplicação.

Controle de concorrência e aquela parte do SGBDD que garante que transações simultaneamente executadas em múltiplos nós produzam o mesmo resultado como se elas tivessem sido feitas num simples nó tendo em conta que: 1º) Os usuários podem acessar dados armazenados em diferentes computadores; e 2º) O mecanismo de controle de concorrência num computador não pode instantaneamente saber das interações em outro computador. É difícil comparar os algoritmos de controle de concorrência já que eles são geralmente complexos, numerosos [3,22], descritos em diferentes terminologias e fazem diferentes suposições sobre o SGBDD.

No Capítulo V, estuda-se este tipo de algoritmos dividindo-os em duas categorias: algoritmos baseados na ordem de escalonamento e algoritmos baseados em técnicas de exclusão mútua. Além disso, são dados alguns conceitos básicos que ajudam a compreensão do tema.

No Capítulo VI apresentam-se as conclusões deste trabalho.

II - CONCEITUAÇÃO DOS SISTEMAS DE GERÊNCIA DE BANCO DE DADOS DISTRIBUÍDOS (SGBDD)

2.1 - CONCEITO DE BANCO DE DADOS DISTRIBUÍDOS (BDD)

Um sistema de Banco de Dados Distribuídos (BDD) existe quando um banco de dados integrado logicamente (i.e. integração lógica significa que qualquer nó tem acesso potencial a todo o banco de dados) é fisicamente distribuído sobre diferentes nós de computação interligados por uma rede. Idealmente, a distribuição física deve ser transparente aos programas de aplicação. Define-se um nó de computação [17] como um computador (mini, micro, etc.) localizado numa área da organização com certas facilidades de processamento. Em cada um dos nós o software do SGBDD consiste minimamente de: 1- Um sistema operacional em cada nó; 2- O gerenciador de comunicação, que permite a troca mútua de informação entre programas remotos. Este gerenciador inclui três níveis de controle: (a) Controle da transmissão física dos dados; (b) Controle da ligação lógica de mensagens entre processos (i.e. programas em execução) e (c) Controle das classes específicas de diálogo (i.e. Transferência de arquivos). 3- Um sistema de gerenciamento de Banco de Dados (SGBD) para fornecer localmente todas as vantagens deste tipo de software. O subconjunto do banco de dados distribuido armazenado num nó é chamado Banco de Dados Local. Assim, cada banco de dados local, tem um SGBD local e o sistema de gerenciamento do BDD (SGBDD) é a união de todos os SGBD's locais.

Se os SGBD's locais são idênticos, então se diz que o

sistema é homogêneo, caso contrário é heterogêneo. O SGBDD é o responsável pelo fornecimento de dados e tradução de requerimentos aos processos num ambiente distribuído heterogêneo.

Além dos conponentes já mencionados, uma entidade importante num ambiente distribuído é o Diretório de Dados ou Catálogo Global, que é o meio pelo qual o SGBDD determina que nós requerem ser acessados com a finalidade de atender uma consulta particular. Sua distribuição (centralizado e/ou distribuido) é importante na performance do sistema.

Na figura 2.1 se apresenta um nó dentro de um SGBDD e as relações existentes entre os componentes do sistema.

O sistema de gerenciamento do BDD (SGBDD) é conhecido também com o nome ESTRUTURA DE CONTROLE [17] por alguns pesquisadores. Esta estrutura deve cumprir as seguintes funções de acordo com as recomendações de CODASYL [48]:

- Interceptar os requerimentos dos usuários e determinar aonde enviá-los para seu processamento e que nós do sistema devem ser acessados para satisfazer esse requerimento.
- Acessar o diretório da rede ou, pelo menos, conhecer como requerer e usar a informação nesse diretório.
- Coordenar o processamento e respostas às consultas.
- Servir de interface de comunicação entre os processos dos usuários e os SGBD locais e os SGBD em outros nós.

Assim, a estrutura de controle é responsável para atingir os principais objetivos de um SGBDD: transparência, fornecer respostas eficientes aos requerimentos dos usuários pres

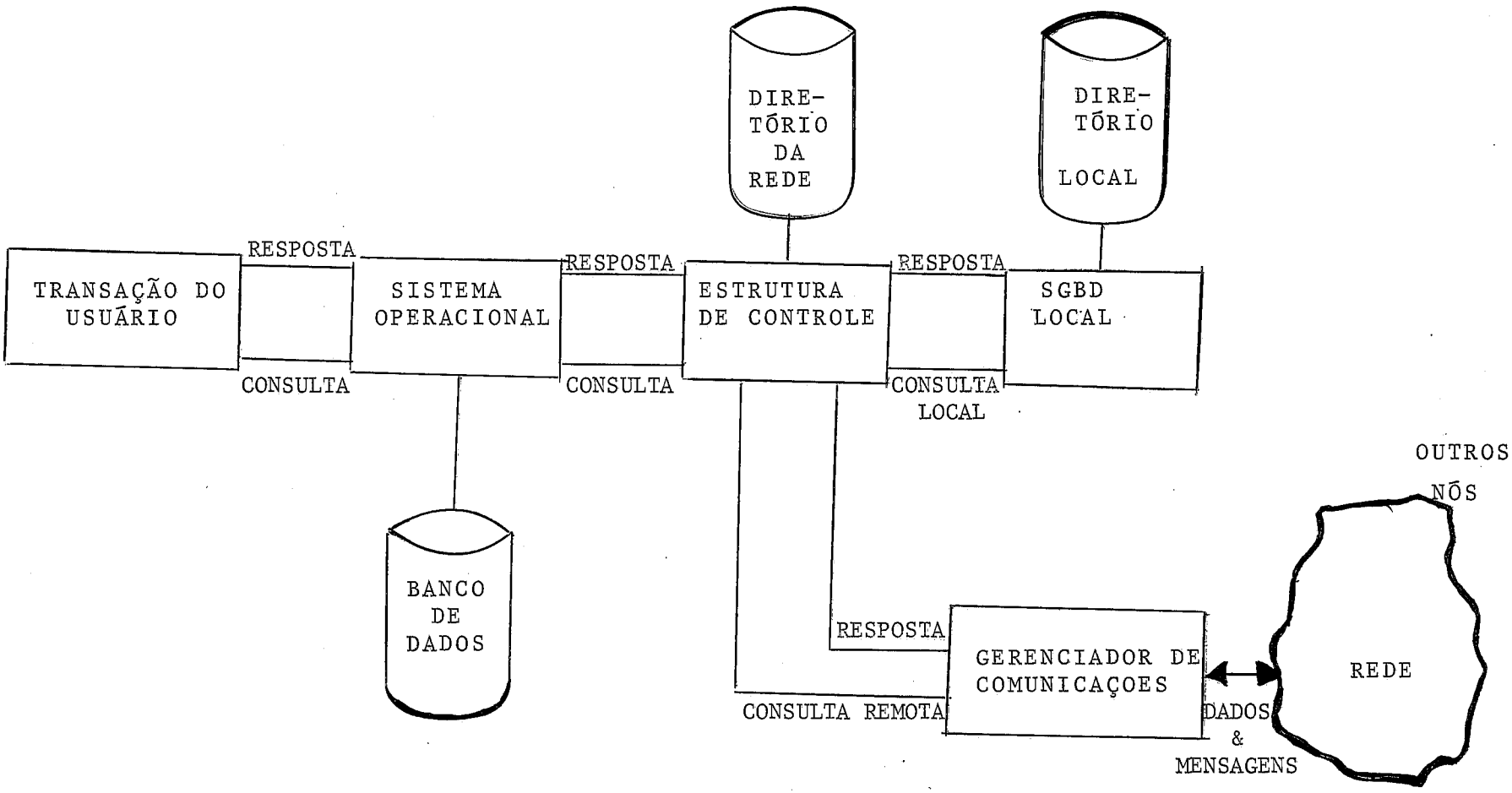


FIG. 2.1 - UM NÓ DENTRO DE UM SGBD (COMPONENTES DO SGBD)

servando a consistência dos dados e a conservação da integrida de do sistema. Um fator crítico deste componente é efetuar o controle em forma centralizada (i.e. só um determinado nó é responsável pelo controle) ou distribuído (i.e. todos os nós, no mesmo nível lógico, cooperam para executar um conjunto de tarefas).

2.2 - ARQUITETURA DE UM SGBDD DESDE O PONTO DE VISTA DO USUÁRIO

Desde o ponto de vista do usuário, um banco de dados consiste de uma coleção de dados lógicos denotados X, Y, Z. Na prática eles podem ser arquivos, registros, etc. Cada dado ló gico é armazenado sobre um ou vários banco de dados locais os quais são administrados por módulos de software especiais cha mados Gerenciadores de Dados (GD). Os usuários interagem com o SGBDD através de transações. As Transações podem ser consultas expressadas numa linguagem especial ou programas de aplicação de propósito geral. A supervisão das interações entre os usuá rios e o SGBDD é feita através de um módulo de software chama do Gerenciador de Transações (GT), existindo um GT para cada nó do sistema.

Assim, um sistema de gerenciamento de Banco de Dados distribuídos contém quatro componentes (Fig. 2.2) [3]:

1 Transações;

2 Gerenciador de Transações;

3 Gerenciador de Dados; e

4 Dados.

As transações se comunicam com GTs, GTs comunicam-se

com GDs e GDs administram os dados (GTs não se comunicam com outros GTs, nem GDs comunicam-se com outros GDs).

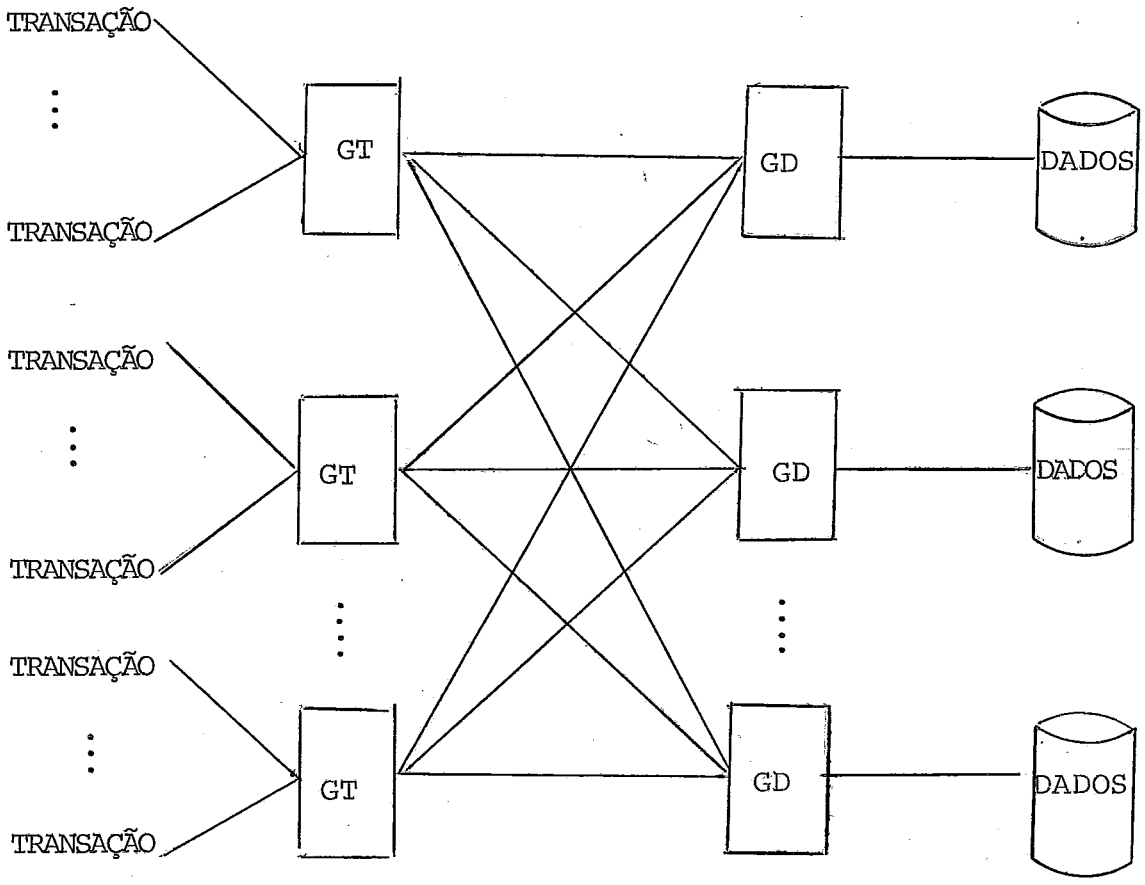


FIG. 2.2 - ARQUITETURA DE UM SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS DISTRIBUÍDO

GTs supervisam transações e cada transação executada no sistema é supervisada por um simples GT. Na interface Transação - GT se definem quatro operações: READ (x), WRITE (x), e as operações que denotam o campo de execução das transações BEGIN e END

Os GDs administram o banco de dados armazenado. Devido ao requerimento das transações GTs enviam comandos aos GDs especificando operações de leitura ou gravação de dados. A interface GT↔GD está influenciada principalmente pela política

de controle de concorrência e processamento de consultas que se estudará posteriormente.

2.3 - ARQUITETURA ANSI - SPARC DE UM SGBDD

Para mostrar como se procede à organização de informação usaremos a abordagem sugerida por grupo de trabalho estabelecido pelo Standard Planning and Requirements Committee (SPARC) do American National Standard Committee on Information Processing (ANSI/X3). Primeiro mostra-se como este grupo considera a organização de informação num banco de dados centralizado (Fig. 2.3).

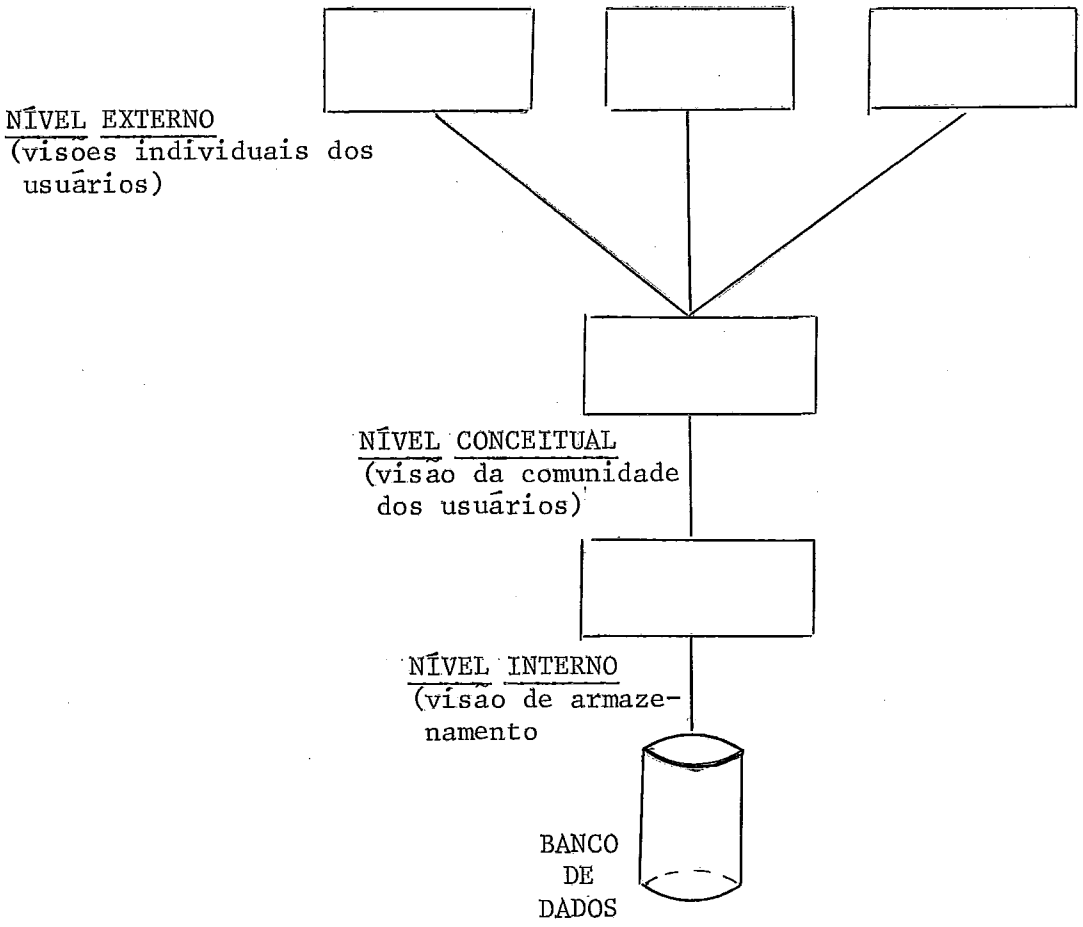


FIG. 2.3 - ARQUITETURA ANSI/SPARC

O nível interno está relacionado com o armazenamento físico. Nesse nível o sistema mantêm a estrutura de armazenamento de informação do Banco de Dados. Nesse nível se decide os métodos de representação das informações sobre o emprego de e truturas de acesso auxiliares, sobre a conveniência de compactar dados, etc.

O nível conceitual tem como objetivos [50]:

1- Documentar e compreender a organização que está projetando o Banco de Dados.

2- Definir os bancos de dados a serem implementados com os SGBD's existentes,com a finalidade de atingir os requerimentos da organização.

3- Fornecer um mecanismo para controlar o uso de dados.

4- Facilitar a migração desde um modelo tradicional a algum modelo de Banco de Dados.

5- Facilitar a migração de um SGBD à seguinte geração de SGBDs.

O esquema conceitual representa o ponto de vista geral da empresa sobre a organização das informações. O esquema conceitual pode ser visto como a definição da visão da comunidade de usuários [15]. Este nível pode também ser visto como a fonte de informação atualizada do Banco de Dados Lógico.

O esquema externo, representa o ponto de vista particular de cada usuário sobre as organizações. Um esquema externo não é um simples subconjunto do esquema conceitual, tem sua

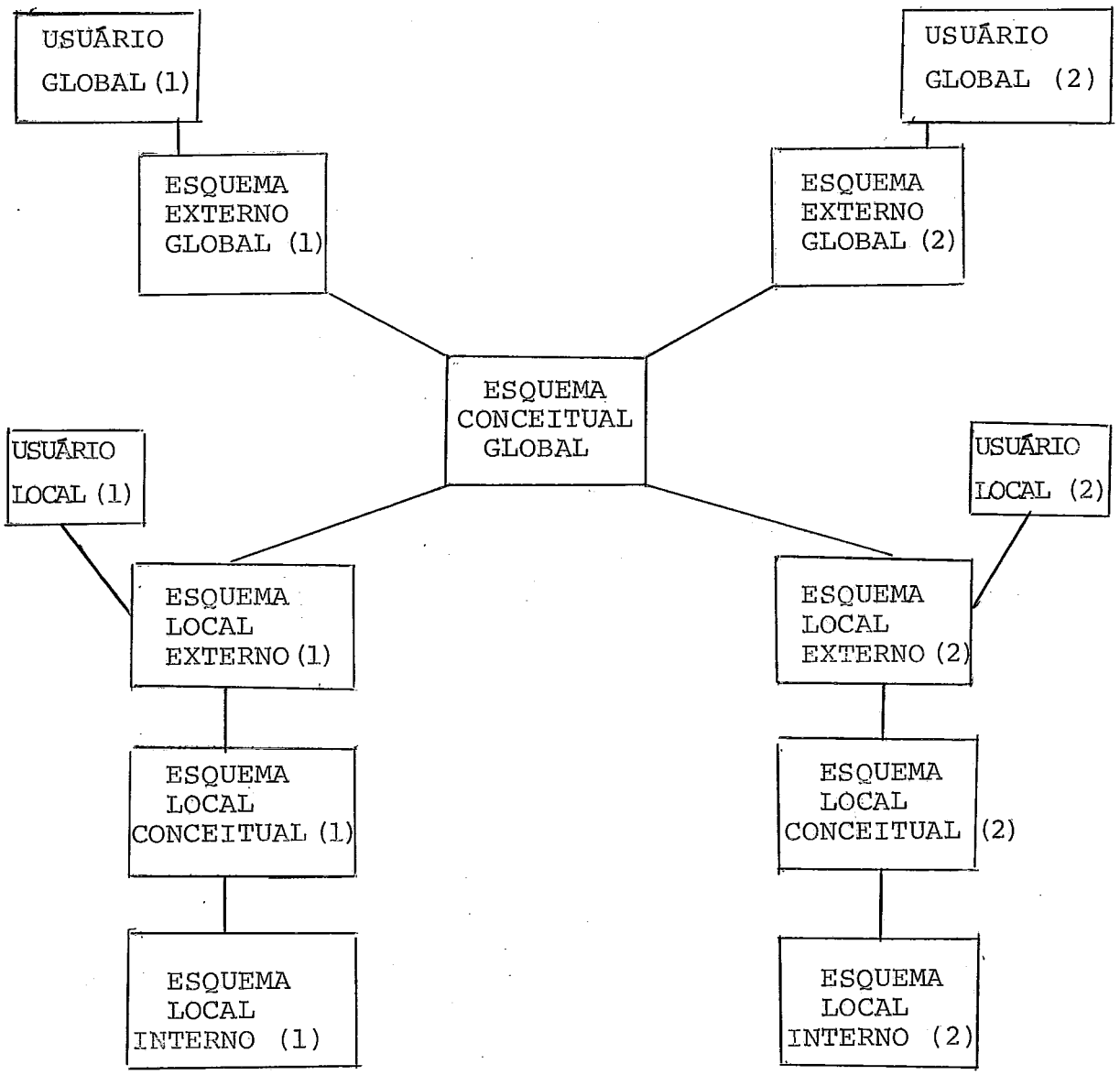
maneira própria de grupar as informações e só pode ser manipulado através de operações determinadas [50].

Entre os vários esquemas são definidos mapeamentos, isto é, os modos de expressar elementos de um esquema em termos de outro esquema. Normalmente os esquemas externos são mapeados para o (expressos em termos do) esquema conceitual [15]. Por sua vez o esquema conceitual é mapeado para o esquema interno. Assim, quando muda o esquema interno (para obter-se maior eficiência), o esquema conceitual não precisa mudar, basta mudar o mapeamento entre os dois. É como o esquema conceitual não mudou, não é preciso mudar os esquemas externos nem os programas de aplicação que o utilizam. Com isso se obtem um alto grau de independência entre os programas e a organização física dos dados [15].

O SGBD provê uma linguagem de definição de Dados (DDL) para especificar o esquema conceitual. Esta linguagem descreve o esquema conceitual em termos de um modelo de dados (Hierárquica, redes ou relacional entre os principais). Também os usuários contam com uma linguagem de Manipulação de Dados (DML) que é a interface entre as transações e o banco de dados e que permite recuperar, eliminar ou atualizar entidades ou objetos do Banco de Dados.

Visto que o Banco de Dados distribuído está formado por uma coleção de Banco de Dados locais, as descrições dos níveis do sistema centralizado mudam num sistema distribuido (Fig. 2.4).

FIG. 2.4 - BANCO DE DADOS DISTRIBUÍDO ANSI/SPARC



No nível geral o Banco de Dados Distribuído se comporta conceitualmente como um Banco de Dados Centralizado simples, e portanto o esquema conceitual global representa a visão lógica dos usuários, a visão unificada de todos os dados armazenados no Banco de Dados distribuídos. Atualmente existem alguns debates para definir se o esquema conceitual global deve ser a união dos esquemas externos locais ou deve ser simplesmente a união dos esquemas conceituais locais [50].

Note-se [1] que num B.D. centralizado, esquemas exter

nos usam subconjuntos do esquema conceitual, enquanto que num ambiente distribuído esquemas externos locais descrevem subconjuntos do esquema conceitual global. Assim esses mapeamentos estão em direções opostas.

2.4 - CRITÉRIOS DE DEFINIÇÃO DE SGBDD: HOMOGÊNEOS E HETEROGÊNEOS

Visto que a arquitetura ANSI/SPARC de Sistemas de Bancos de Dados Distribuídos focaliza melhor a organização de informação, neste ambiente é preciso examinar os diferentes possíveis critérios para definir bancos de dados distribuídos homogêneos e heterogêneos.

Se diz que os sistemas de gerenciamento de bancos de dados distribuídos (SGBDD) são homogêneos, se todos os DBMS locais [47]:

1- Usam o mesmo modelo de dados (redes,hierárquico,relacional, etc.) para descrever o banco de dados.

2- Usam o mesmo DDL e DML (ou, pelo menos DDLs da mesmafamília);

3- Provêm os mesmos serviços aos usuários em diferentes nós;

4- Executam as mesmas funções através dos mesmos algoritmos.

No caso contrário, se diz que o sistema de gerenciamento de Banco de Dados Distribuídos é Heterogêneo.

Os sistemas homogêneos aparecem com maior frequência

(47,19) em aplicações desenvolvidas e os heterogêneos aparecem pela integração de aplicações existentes devido aos seguintes fatores: aproveitamento do hardware, custos de aproveitamento do ambiente existente, hábitos dos usuários e grau de colaboração (47).

2.5 - VANTAGENS DOS SGBDD

Entre as principais vantagens dos SGBDD pode-se citar as seguintes:

- Crescimento modular: Um sistema distribuido pode crescer mais facilmente do que um sistema centralizado. Se é necessário expandir o sistema porque o volume de processamento aumentou, então é mais fácil incrementar um novo nó à Rede de computadores do que substituir um sistema centralizado por outro maior. As adições de novos nós ao sistema podem ser feitas em pequenos incrementos, sem causar interrupções no funcionamento geral do SGBDD.

- Maior confiabilidade: Um SGBDD é construído baseado em computadores múltiplos localizados em multiplas locações com a vantagem consequente de que a falha de um nó não impede o funcionamento normal (num nível reduzido) do sistema distribuído.

- Eficiência e flexibilidade: Num SGBDD é possível armazenar os dados nos nós onde eles são mais freqüentemente usados, reduzindo assim os tempos de respos

ta e custos de comunição. Além do mais, os computadores da rede são dedicados a uma tarefa ou a um conjunto de tarefas específicas. Consequentemente,o máximo "throughput", o qual é necessário para o sucesso de sistemas em linha pode ser atingido.

Ao lado das vantagens já expostas, os SGBDD apresentam algumas dificuldades que serão analizadas neste estudo, entre as quais se podem citar às três seguintes:

1- A distribuição de arquivos e diretórios dentro da rede para satisfazer de forma eficiente os requerimentos dos usuários.

2- O processamento de consultas mediante uma estratégia eficiente que minimize os acessos inter-nós e os custos de comunicação.

3- O processamento concorrente de transações preservando a integridade do sistema.

No capítulo seguinte se analisará o problema de distribuição de arquivos e diretórios apresentando-se diferentes modelos de solução para atingir os objetivos de um SGBDD.

III - DISTRIBUIÇÃO DE ARQUIVOS E DIRETÓRIOS EM BANCO DE DADOS DISTRIBUIDOS

3.1 - INTRODUÇÃO

Um dos principais problemas na área de Banco de Dados Distribuídos (BDD) está relacionado com a determinação dos locais de armazenamentos dos arquivos e diretórios do Banco de Dados nos diversos nós da rede.

Alguns dos fatores que devem ser considerados na alocação de arquivos e diretórios são [39]:

- Frequência de acesso - o número de vezes em que cada usuário acessa um arquivo;
- Custo de transmissão - o custo de transmitir ou receber mensagens de consulta ou manutenção de determinado arquivo;
- Custo de armazenamento - custo de armazenar o arquivo no nó da rede;
- Atividade inter-nós - frequência de comunicação entre os nós da rede;
- Atividade intra-nós - frequência do manuseio local de determinado arquivo nos nós;
- Compartilhamentos dos dados pelos usuários a fim de encontrar a melhor maneira de agrupá-los;
- Tamanho do diretório - custo de armazenamento do diretório em função do seu tamanho;
- Freqüência de atualização do diretório - o número de vezes em que o diretório será acessado em função

da criação, exclusão e alteração (tamanho) dos arquivos.

3.2 - DISTRIBUIÇÃO DE ARQUIVOS

Um BDD pode ter seus arquivos armazenados de três formas através do sistema:

1º) Arquivos Individuais;

2º) Arquivos Replicados;

3º) Arquivos Particionados.

3.2.1 - ARQUIVOS INDIVIDUAIS

Cada usuário cria seus próprios arquivos e os manipula num único processador. Note-se que os arquivos podem ser acessados por outros nós do sistema.

Os múltiplos arquivos que formam um banco de dados distribuidos podem estar separados em grupos, cada qual localizado em um nó da rede.

Critérios para alocar arquivos individuais:

3.2.1.1 - LOCALIZAÇÃO GEOGRÁFICA

Alocar os arquivos pela localização geográfica dos usuários que os acessam, i.e., pela quantidade de usuários próximos que acessam esses arquivos. Por exemplo, se 10 usuários

do sul do país acessam o mesmo arquivo que dois usuários do centro, deve-se colocar o arquivo no sul do país.

Uma desvantagem desta alocação é que a frequência de acesso pode fazer mudar o critério. No exemplo anterior se os dois usuários do Centro tiverem uma frequência de acesso muito maior do que a dos 10 usuários do sul do país, o procedimento pode-se tornar inviável [38].

3.2.1.2 - LOCALIZAÇÃO PELO TIPO

Agrupar os arquivos com base no seu tipo (empregados, saldo, produto, etc.).

Como é desejável que um arquivo seja localizado no nó que mais o acessa, a distância de um arquivo a um nó deve ser inversamente proporcional a frequência de acesso de um nó ao arquivo. Isto é, quanto maior a frequência de acesso de um nó a um arquivo, mais próximo deste nó deve estar localizado o arquivo. Outro fator de relevante importância é o volume dos dados, porque, mesmo com baixa frequência de acesso a um arquivo, o volume de dados acessados pode ser alto. Por exemplo, um nó acessa 1.000 vezes os itens de um registro de determinado arquivo, enquanto outro nó acessa 100 registros desse mesmo arquivo. Sendo o registro 20 vezes maior que o item, este segundo nó tem um volume de transmissão maior.

3.2.2 - ARQUIVOS REPLICADOS

Neste método, duas ou mais cópias do mesmo arquivo e

xistem no sistema e, no caso extremo, cada nó pode ter uma cópia de um arquivo determinado.

Os principais objetivos a serem alcançados com as duplicações de arquivos são:

- Aumentar a acessibilidade do arquivo - a concorrência no arquivo diminui com o aumento do número de cópias, o que permite respostas mais rápidas, maior disponibilidade e torna possível aumentar o número de usuários do arquivo;
- Fornecer um rápido "backup" nos casos de falhas em nós do sistema que contenham cópias do arquivo;
- Diminuir o volume de comunicação e custo de transmissão;

Os argumentos a favor da duplicação de dados se podem reunir da forma seguinte:

Se um arquivo é armazenado em N nós, então a disponibilidade do dado para um requerimento de leitura é maior do que se o arquivo fosse armazenado só num nó. Se p é a probabilidade de que um nó esteja ativo, então a probabilidade de que um requerimento de leitura possa ser atendido é p se o dado não é replicado. Se o dado é replicado em N nós, a probabilidade de que um requerimento de leitura possa ser atendido é $1 - (1-p)^N$, visto que a leitura pode ser atendida por qualquer uma das N cópias. Assim, se reduz o tempo de resposta e o tráfego de comunicações na rede.

Os problemas principais criados pela multiplicidade de cópias são os seguintes:

- A manutenção de arquivos com cópias atualizadas é

muito difícil e deve ser realizada de preferência quando nós que contêm cópias estão desativados. Considere um nó desativado, com suas cópias desatualizadas devido ao tempo em que este nó não estava operando. Ao tornar-se ativo, esse nó deverá realizar as atualizações em suas cópias. Por isso, a duplicação de arquivos é aceitavel quando esses arquivos têm muita leitura e pouca atualização, devido ao fato da manutenção que pode afetar a integridade do BDD.

- Além disso, a solução do problema da atualização deve considerar o momento em que a atualização for realizada em cada cópia (arquivo de atualização em separado), pois a atualização válida é a mais recente. Também os tempos devem estar sincronizados e não podem ser alterados enquanto a rede estiver ativa.

  A solução para evitar erros nas atualizações das cópias é fazer com que qualquer atualização em um arquivo que tenha cópia implique a imediata atualização de suas cópias. Portanto (16), nota-se que o problema de atualização de cópias é muito complexo,com exigência de pesquisas para se obter um método satisfatório.

3.2.2.1 - ARQUIVOS REPLICADOS SEGUNDO O MODELO RELACIONAL

Um modo elegante de visualizar arquivos replicados é

através das relações [17]. Tomemos como exemplo uma relação de pedidos feitos por clientes num armazem. Seja a relação:

PEDIDOS (CLIENTE, NOME, Nº PARTE, DESCRIÇÃO DA PARTE, QUANTIDADE PEDIDA)

Considere que os valores assumidos pelos atributos das tuplas que formam a relação PEDIDOS sejam os apresentados na Fig. 3.1

	CLIENTE	NOME	Nº PARTE	DESCRIÇÃO DA PARTE	QUANTIDADE PEDIDA
t_1	1	ABC	2	SAL	25
t_2	1	ABC	10	MANTEIGA	16
t_3	2	CDE	10	MANTEIGA	18
t_4	3	DEF	3	PÃO	9
t_5	3	DEF	2	SAL	8

Fig. 3.1 - Exemplo de ocorrências dos valores dos atributos da relação PEDIDOS.

Neste método de alocação de arquivos, as ocorrências de dados são replicadas de maneira planejada [19]. Num sistema COMPLETAMENTE REPLICADO todas as ocorrências de dados aparecem em cada um dos nós, i.e., uma cópia idêntica da relação é armazenada em todos os nós. Num sistema PARCIALMENTE REPLICADO, algumas tuplas são replicadas em diversos nós, por exemplo, as tuplas t_1, t_2 seriam armazenadas sobre o nó 1; t_1, t_4, t_5 sobre o nó 2 e t_1, t_5 sobre o nó 3.

3.2.3 - ARQUIVOS PARTICIONADOS

Nesse esquema os arquivos de dados são divididos em conjuntos disjuntos, e cada um dos conjuntos é alocado a um nó específico e sem duplicações.

Os principais casos de particionamento de arquivos são os seguintes:

- O particionamento é requerido quando o arquivo não pode ser manipulado por um único nó. Isto é, o arquivo é muito grande para ser colocado em um único nó ou sua taxa de acesso, por ser muito alta, não é suportada por um único nó do sistema.
- A freqüência de acesso às diferentes partes do arquivo pode variar em função dos diferentes nós. Por exemplo, no caso de um arquivo sobre brasileiros, o nó localizado em Recife teria frequência de acesso a "Pernambucanos" residentes em Recife muito maior que a freqüência de acesso a paulistas, gaúchos, etc. Deste modo, é conveniente que as informações sobre Pernambucanos fiquem localizadas em Pernambuco, para minimizar o custo das comunicações e o tempo de resposta.

Os problemas principais em arquivos particionados são os seguintes:

- A catalogação do arquivo:

Uma das maneiras é [38] fornecer nomes diferentes para cada segmento, ou um único nome para todo o arquivo com indicações referentes a cada segmento.

- A manipulação do arquivo:

Considere um arquivo sequencial - indexado particionado entre vários nós. Onde localizar o índice: em um único nó? partido entre os nós? ou duplicado entre nós?

- Outro problema é como endereços idênticos em diferentes nós são resolvidos no índice.

3.2.3.1 - ARQUIVOS PARTICIONADOS SEGUNDO O MODELO RELACIONAL

Com base no modelo relacional de Banco de Dados, se pode visualizar o particionamento de arquivos tendo-se dois métodos de distribuição [19,44] por particionamento:

a) Particionamento por ocorrência ou Particionamento Horizontal.

A relação é dividida em sub-relações com a propriedade de que cada uma das sub-relações tem a mesma estrutura lógica (a mesma sequência de campos). Cada uma das sub-relações é conhecida como partição horizontal e são armazenadas sobre os nós de acordo com algum critério de distribuição. Por isso este método também é conhecido como particionamento por ocorrência.

Na Fig. 1, por exemplo, t_1, t_2 podem ser alocados ao nó 1; t_2, t_3 ao nó 2 e t_4 ao nó 3.

b) Particionamento por estrutura ou Particionamento Vertical.

Neste método se divide uma relação ou uma partição ho

rizontal,[44] em sub-relações resultantes de operações de projeção sobre a estrutura original, i.e. armazenando só alguns campos sobre cada um dos nós. Este tipo de particionamento é também chamado particionamento por estrutura.

Na Fig. 3.1, por exemplo, a tabela é cortada verticalmente criando-se três relações: A, B, C.

A (Cliente, Nome) B (Nº parte, Descrição da Parte)
C (Cliente, Nº Parte, Quantidade-Pedida)

Assim, todas as ocorrências de A podem existir no nó 1, as ocorrências de B no nó 2 e todas as ocorrências de C no nó 3.

3.2.4 - CLASSES DE BANCO DE DADOS DISTRIBUIDO: HOMOGÊNEO/HETEROGÊNEO E DISTRIBUIÇÃO DE ARQUIVOS

Se os sistemas de gerenciamento de banco de dados local num SGBDD são idênticos se diz que o sistema é homogêneo, caso contrário é heterogêneo. É aparente que os sistemas heterogêneos possuem mais dificuldades do que os homogêneos. Entre essas dificuldades temos o mapeamento entre os diferentes modelos de dados sobre os quais cada um dos SGBD são baseados; diferenças nas linguagens de manipulação de dados (DML's) e diferenças entre o formato e a apresentação dos dados (47). Num sistema homogêneo o hardware de cada nó é também idêntico facilitando a implementação do sistema. Por outro lado, num sistema heterogêneo o hardware de cada nó pode ser idêntico, mas o software é diferente [19] o que não permite que as implementa

ções dos sistemas locais sejam idênticas.

Os métodos de distribuição de dados, tomando em consideração os conceitos expostos, são sumariamente definidos do seguinte modo [19]:

	REPLICADO	PARTICIONADO
HOMOGÊNEO	Uma ou mais cópias do arquivo existem em múltiplos nós e com idênticas implementações	O arquivo é dividido em partes ou subconjuntos disjuntos, sem duplicações, e cada um deles tem implementação idêntica
HETEROGÊNEO	Uma ou mais cópias do arquivo existem em múltiplos nós e em diferentes formas de implementação	O arquivo é dividido em subconjuntos disjuntos, sem duplicações, mas pode ter variações na implementação de suas partes

3.3 - MODELOS DE ALOCAÇÃO DE ARQUIVOS BASEADOS EM MÉTODOS DE PROGRAMAÇÃO MATEMÁTICA

Introdução

A distribuição de dados é um dos maiores problemas num projeto de BDD. A maioria dos pesquisadores a ele se refere como problema de alocação de arquivos ou Gerenciamento de Dados [17]. Pesquisas têm sido feitas no sentido de aplicar técnicas de programação matemática clássica para solucionar o seguinte problema:

Dado: (1) Uma descrição de demandas dos usuários por serviços considerando-se: volume de recuperações e volume de de atualizações de cada nó da rede a cada arquivo (local do

Banco de Dados); e

(2) Uma descrição dos recursos disponíveis para fornecer essa demanda através da topologia da rede; capacidade de conexão e custos; e capacidade dos nós e custos;

Determinar: Uma alocação de arquivos aos nós, levando em conta as restrições de capacidade e tendo a preocupação de minimizar os custos totais do sistema.

O problema de alocação de dados é uma forma generalizada [18,42] do problema de alocação de arquivos, tomando em conta a seguinte diferença:

A unidade de alocação num sistema de distribuição de arquivos é fixa: arquivos completos. Por outro lado num sistema de BDD a unidade de alocação é variável em função das atualizações, consultas e outros critérios de distribuição que permitem o particionamento do arquivo em fragmentos menores.

3.3.1 - MODELO DE CHU W. W. (1969): CUSTO MÍNIMO/MODELO RELACIONAL

CHU (1969) analisa o problema de distribuição de dados baseado no modelo relacional.

CHU [12] tem considerado um modelo de rede em conexão full-duplex, na qual o principal tráfego é devido às tuplas que são enviadas em respostas às consultas dos usuários ("queries"). Além disso, a demora q_{ik}, requerida para que uma consulta de i seja respondida por uma tupla localizada em k, deve-se unicamente a demoras na fila sobre a linha k_i. Um sumário do modelo (ignorando atualizações) é o seguinte:

Dado: N número de nós;

M número de relações;

C_i capacidade de armazenamento (em bytes) no nó i;

$\$_i$ custo de armazenamento (bytes por segundo) no nó i;

L_j tamanho da relação j;

l_j tamanho da tupla (em bytes) para a relação j;

r_{ij} tuplas/seg de nó i para a relação j;

t_{ij} demora máxima aceitável desde o nó i à tupla da relação j;

$\$_{ik}$ custo de comunicação (por byte) para tuplas desde i ao nó k;

q_{ij} demora da consulta do nó i ao nó j

Variáveis:

$$X_{ij} = \begin{cases} 1 & \text{se a relação j está no nó i} \\ 0 & \text{no caso contrário} \end{cases}$$

Restrições:

1. Cada uma das relações é armazenada só em um nó da rede.

$$\sum_{i=1}^{N} X_{ij} = 1 \qquad (1 \leq j \leq M)$$

M: nº de relações

2. O espaço de armazenamento num nó não pode ser excedido.

$$\sum_{j=1}^{M} X_{ij} \; L_j \leqslant C_i \qquad (1 \leqslant i \leqslant N)$$

3. Se impõe uma demora máxima aceitável para uma transação envolvendo uma tupla

$$q_{ij} \leqslant t_{ij} \qquad (1 \leqslant i \leqslant N, \quad 1 \leqslant j \leqslant M)$$

Objetivo: Minimizar o custo total

$$\sum_{i=1}^{N} \sum_{j=1}^{M} X_{ij} \; \$_i \; L_j \; +$$

Custo de armazenamento por segundo

$$\sum_{i=1}^{N} \sum_{k=1}^{N} \sum_{j=1}^{M} X_{kj} \; \$_{ki} \; r_{ij} \; l_j$$

Custo de comunicação por segundo.

A solução do problema de alocação de relações é uma atribuição de 1's e 0's à matriz X, correspondendo em que nó cada relação vai ser armazenada. A técnica usada para encontrar a solução é a de programação linear zero-um.

Em resumo [48]: No modelo de CHU, os seguintes pontos são considerados fatores primários no problema de alocação de relações a nós.

- Frequência de pedidos de leitura e atualização da relação j devido a consultas desde o nó i.
- Capacidade de armazenamento disponível em cada nó.
- Custo de armazenamento por byte em cada um dos nós.
- Custo de comunicação entre pares de nós.

3.3.2 - MODELO DE CASEY, R.G. (1972): CUSTO MÍNIMO/ARQUIVO SIMPLES

Neste modelo considera-se um modelo matemático (9) de uma rede informação de N nós, alguns deles contendo cópias de um único arquivo. No interior dessa rede, cada nó é capaz de comunicar-se com cada um dos outros nós sobre linhas de comunicação e através de dois tipos de transações:

1. Tráfego de consultas entre um nó e o arquivo; e
2. Tráfego de atualização.

Uma mensagem de atualização é transmitida a cada uma das cópias do arquivo, enquanto uma consulta é comunicada apenas a uma cópia simples.

O modelo propõe os seguintes dados de entrada:

N - Número de nós

$\$_k$ - O custo fixo (principalmente para armazenamento) de localizar uma cópia do arquivo no k-ésimo nó ($k = 1, 2, N$). Este custo é medido em dólares/mês

T_{i_c} - Volume do tráfego de consultas (megabytes/mês) geradas no nó i

T_{i_a} - Volume do tráfego de atualizações (megabytes/mês) geradas no nó i

$\$_{jk_c}$ - O custo de uma unidade de comunicação de nó j ao nó k para uma consulta

$\$_{jk_a}$ - O custo de uma unidade de comunicação de nó j ao nó k para uma atualização

x_i - Variável binária, é 1, quando uma cópia do arquivo é alocada ao nó i; Ø no caso contrário.

Objetivo: Minimizar a função de custos C tal que:

$$C = \sum_{j=1}^{N} \left(\underbrace{\sum_{k=1}^{N} T_{i_a} \$_{jk_a} x_j}_{\text{Custo de atualização de todas as cópias do arquivo}} + \underbrace{T_{i_c} \min_k \$_{jk_c}}_{\text{Custo mínimo da consulta num determinado nó}} \right) +$$

$$\sum_{k=1}^{N} \underbrace{x_k \$_k}_{\text{Custo de Armazenamento}}$$

Restrições: $x_j = 0$ ou 1 (requerimento inteiro)

O custo total de comunicações resultante da eleição dos nós nos quais se localizará o arquivo é uma soma sobre os nós individuais dos usuários. No caso do tráfego de consultas, assume-se que o usuário acessa uma cópia do arquivo que minimiza seus custos de comunicação.

O principal objetivo do modelo é determinar em que

nós da rede cópias de um único arquivo serão armazenados. Esta alocação, se presume, será feita de tal maneira que possa minimizar o custo total de comunicação entre usuários e arquivos.

Em geral, o custo de realização de consultas é reduzido se for aumentado o número de duplicações de arquivos nos nós da rede (i.e. os usuários podem achar mais perto o arquivo requerido). De outro lado, os custos de armazenamento e atualização aumentarão com as duplicações, já que cada cópia deve ser atualizada. No caso extremo, se nenhuma atualização é feita e se os custos de armazenamento são baixos, uma cópia do arquivo pode ser mantida em cada nó; caso só sejam feitas atualizações, sem consultas, sugere-se que uma cópia simples do arquivo seja mantida em algum nó (o nó ótimo) da rede [9]. Assim, Casey tem pesquisado o problema de achar condições ótimas para criar cópias múltiplas de arquivos.

A técnica usada para atingir uma solução do problema se baseia na programação inteira ou na teoria de grafos (busca da rota ótima num grafo de custos) para determinar o limite máximo do número de cópias de arquivos que estarão presentes na rede levando em conta as proporções do tráfego de atualização em relação ao tráfego de consultas gerado pelos usuários de um arquivo na rede.

3.3.3 - MODELO DE MORGAN E LEVIN (1977): CUSTO MÍNIMO/PROGRAMAS E ARQUIVOS DE DADOS

O modelo de Morgan e Levin considera relevante o relacionamento entre programas e arquivos de dados na formulação do algoritmo de distribuição de dados: os programas e os arqui

vos de dados não são acessados independentemente e qualquer transação executará um programa antes de processar um arquivo de dados.

A rede de computadores considerada neste modelo (37) se compõe de N nós e de um banco de dados consistindo de F arquivos e P programas. O problema é achar uma distribuição ótima de arquivos e programas que minimizem o custo de operação do Banco de Dados. Cada um dos nós na rede demanda os serviços de alguns programas e arquivos. Essa demanda é gerada através de transações, originadas nos próprios nós, que são de duas classes: 1) Tráfego de consultas; e 2) Tráfego de atualizações. Além disso, os arquivos de programas devem ser restringidos para serem executados só sobre determinados nós.

Uma transação é encaminhada inicialmente a seu programa pertinente e desse programa a consulta é transmitida à cópia do arquivo mais próximo, enquanto uma mensagem de atualização é transmitida a cada uma das cópias do arquivo.

Assim, nesse modelo se considera que não são os usuários da rede, os que efetivamente requerem dados, mas os programas que estão em comunicação com eles, possivelmente em outros nós. Se presume que qualquer programa P pode requerer acesso a alguns arquivos f_1, f_2, ..., f_n. Também se presume que o acesso a f_1 é independente dos acessos a f_2, ..., f_n e que o acesso a f_1 é dependente do programa P, i.e.

$$P_r(f_1/p_1) \neq P_r(f_1/p_2)$$

onde: p_1, p_2 são programas diferentes

Em resumo, para processar uma determinada transação, se deve processar o programa pertinente e o arquivo correspon

dente.

Os dados de entrada considerados nesse modelo são:

T_{ipf_c} Tráfego de consultas desde o nó i ao arquivo f mediante o programa P

T_{ipf_a} Tráfego de atualização desde o nó i ao arquivo f mediante o programa P

$\$\ ij_c$ Custo de comunicação da consulta desde o nó i ao j

$\$\ ij_a$ Custo de atualização desde o nó i ao j

Se faz uma diferença entre atualizações e consultas (volume de tráfego e custo de comunicação) com base na premissa seguinte: "As consultas precisam de um tempo de resposta mais veloz do que o tráfego de atualização".

O modelo pode ser formalizado como um problema de minimização de custos, no qual os determinantes dos custos são:

Z_1 Custo de comunicação das consultas: dos nós iniciais aos programas,

Z_2 Custo de comunicação das atualizações: dos nós iniciais aos programas

Z_3 Custo de comunicação das consultas: dos programas aos arquivos

Z_4 Custo de comunicação das atualizações: dos programas aos arquivos

Z_5 Custo de armazenamento de arquivos de dados

Z_6 Custo de armazenamento de programas.

A função objetivo é:

Minimizar $C = Z_1 + Z_2 + Z_3 + Z_4 + Z_5 + Z_6$ sujeita às seguintes restrições:

1) Se deve ter como mínimo uma cópia de cada um dos programas e de cada um dos arquivos para alcançar uma solução;

2) Todos os requerimentos para programas e arquivos de dados devem ser satisfeitos, i.e. cada transação deve ter uma rota definida;

3) Os requerimentos para arquivos de programas e arquivos de dados são reencaminhados só aos nós os quais tem o arquivo requerido;

4) Garantir que cada programa resida só num nó, onde ele possa ser processado.

O problema formulado pertence à classe de problemas de programação não-linear zero-um [37]. O modelo foi ampliado para incluir algumas restrições práticas, tais como:

1 - Tempos de acesso;

2 - Políticas de Segurança; e

3 - Memória de armazenamento.

As restrições impostas no modelo garantem controlar níveis de performance do sistema no processo da busca da solução da função objetivo (18).

3.3.4- MODELO DE GHOSH (1976): CUSTO MÍNIMO/ASSOCIAÇÕES DE DADOS

A pesquisa do problema de distribuir um banco de dados, que contêm associações lógicas entre tipos de segmentos (i.e., um segmento é uma agrupação lógica de informação) sobre

uma rede de computadores de tal forma que os tipos múltiplos de segmentos satisfaçam uma consulta que possa ser recuperada em paralelo de diferentes nós [23]. O processamento em paralelo de diferentes subconjuntos de dados pertinentes a uma consulta é uma característica importante pela redução do tempo de resposta.

Nesse modelo se discutem as propriedades de distribuições de dados sem ou com redundância e se estabelecem os limites mínimos e máximos de tais distribuições. O limite menor depende do armazenamento redundante de tipos de segmento na rede ser permitido ou não; distribuição com redundância apresenta o limite menor.

A técnica usada no modelo é baseada numa busca combinatória através de várias soluções possíveis. A contribuição do algoritmo é a de formular um modelo que leva em conta as associações lógicas entre subconjuntos de dados para que possam ser recuperados em paralelo, minimizando assim o tempo de resposta.

3.4-MODELOS DE ALOCAÇÃO DE ARQUIVOS BASEADOS EM MÉTODOS HEURISTICOS

Introdução

Os algoritmos baseados nos métodos de programação inteira ou teoria de grafos têm como desvantagem, para o problema de distribuição de dados, de serem de dificil implementação; isto ocorre devido às seguintes razões:

a) Consomem grande tempo de CPU - Ramamoorthy et alii [22] afirmam que esses algoritmos são muito caros de ser executados em tempo real. Além disso, o tempo de computação para todos os algoritmos de otimização aumenta exponencialmente com o tamanho do problema: CASEY [5] observou que um programa demorou aproximadamente uma hora para resolver um problema particular de alocação ótima para uma rede de 30 nós. Assim, se n representa o tamanho do problema, então o tempo de computação cresce na ordem de k^n, onde $k > 1$ [24].
GRAPA et alii [24] estabeleceram regras para determinar a "priori" que nós devem ser (ou não devem ser) incluidos no problema de alocação ótima com a finalidade de tornar factível a solução do problema, mediante a redução de seu tamanho.

b) Os requerimentos de armazenamento são muito grandes pelo número de restrições geradas.
Devido a essas duas razões gerar-se-ão soluções praticáveis para obter soluções de baixo custo para problemas de grande porte, nos quais a aplicação de técnicas de programação linear ou não-linear são computacionalmente intratáveis. Esse tipo de soluções é chamado de método heurístico. Considera-se neste estudo dois principais modelos:
1º) Modelo de Custo Mínimo/Arquivo Simples (CHANDY & HEWES - 1976); e
2º) Modelo de Custo Mínimo/Arquivos Múltiplos -

(MAHMOUD & RIORDON - 1976).

3.4.1- Modelo de Chandy e Hewes: Custo Mínimo/Arquivo Simples

Neste modelo [18] se assume que os custos de atualização dependem do lugar onde o arquivo é alocado e são independentes do número e alocação das outras cópias do arquivo. Então, a função objetivo é baseada sobre os custos de armazenamento/atualização e custos de comunicação das consultas.

Parâmetros de Entrada:

X_i = { - Variável binária de alocação do arquivo:
1 Quando uma cópia do arquivo é alocada ao nó i
0 No caso contrário

Y_{ij} = { - Variável binária de alocação de rotas de busca de arquivos:
1 Quando consultas do nó i são enviadas ao nó j que possui uma cópia do arquivo
0 No caso contrário

T_{i_c} Volume do tráfego de consultas geradas no nó i

$\$_i$ Custo do armazenamento do arquivo, ou uma cópia de um arquivo, no nó i

$\$_{ij_c}$ Custo de comunicação das consultas do nó i ao nó j

N Número de nós

Função Objetivo:

$$\text{Minimizar} \sum_{i} \left(\underbrace{X_i \ \$_i}_{\text{Custo de armazenamento/atualização}} + \sum_{\substack{j \\ j \neq i}} \underbrace{Y_{ij} \ T_{i_c} \ \$_{ij_c}}_{\text{Custos das Consultas}} \right)$$

$$i, j = 1, 2, \ldots N$$

Restrições:

1) As consultas devem ser satisfeitas só num único nó. Se a consulta é formulada num nó onde o arquivo é residente, então a consulta deve ser local.

$$\sum_{\substack{j \\ j \neq i}} Y_{ij} + X_i = 1$$

2) Sempre devem existir o arquivo requerido e sua rota de busca: Requerimento de consistência.

$$Y_{ij} \leq X_j \qquad i \neq j$$

A técnica de solução da função objetivo usada para atingir um limite superior de custos é uma técnica iterativa (18) Inicialmente uma cópia do arquivo é alocada a um nó Random. Todas as alocações de arquivos são avaliadas dentro de uma "distância" M. Isto significa que M ou um número de cópias menor que M são somadas ou eliminadas da alocação anterior. O custo mínimo de alocação é selecionado e o procedimento continua ite

rativamente até que não exista um custo de alocação menor dentro dessa distância M. Uma distância de 2 parece ser ótima no procedimento heurístico [18].

3.4.2 - MODELO DE MAHMOUD E RIORDON (1976): CUSTO MÍNIMO/ARQUIVOS MÚLTIPLOS

Este modelo é um exemplo da relação entre o problema de alocação de arquivos e o problema de alocação de capacidades aos canais de comunicações [18]. Seu objetivo é alocar cópias de arquivos de informação aos nós e capacidade às conexões da rede (i.e. canais) de modo que o custo mínimo seja atingido em base às demoras na fila de espera da rede e às restrições na disponibilidade dos arquivos.

A função objetivo procura minimizar os custos do canal e custos de armazenamento sujeita às seguintes restrições:

a) O tempo de resposta deve ser menor do que um máximo aceitável previamente estabelecido,

b) Os arquivos devem estar disponíveis no momento do requerimento. Este algoritmo é também conhecido como o MÉTODO DE SOMA-E-DIMINUIÇÃO.

É muito difícil achar uma solução pelos métodos de programação matemática [33] e por isso só se aplicam os métodos heurísticos. Uma das técnicas de solução consiste de duas rotinas [33]; A rotina de inicialização e a rotina de otimização. A rotina de inicialização gera um número de soluções iniciais possíveis (uma solução possível é uma alocação de arquivos e capacidade que satisfaça as restrições de demora e dispo

nibilidade). Começando com uma solução inicial a rotina de otimização tenta otimizar o custo total por sucessivas somas e diminuições de cópias de arquivos. Quando uma solução possível com menores custos é achada, ela é adotada como uma nova solução inicial e o processo continua. Às vezes, é encontrada uma solução possível, cujo custo não pode ser reduzido por maiores realocações de arquivos. Então, essa é considerada uma "solução ótima local". Cada execução da rotina de otimização para achar um mínimo local correspondente a uma solução inicial é chamada de "execução heurística". O resultado da execução heurística que corresponde ao custo mínimo de todas as soluções achadas é adotado como a solução final.

A realocação de arquivos se baseia em duas rotinas, a rotina de soma para adicionar cópias extras e a rotina de diminuição, que elimina cópias de arquivos da rede.

3.5 - DISTRIBUIÇÃO DE DIRETÓRIOS

INTRODUÇÃO

O diretório é um conjunto de informações pelas quais os arquivos são identificados e localizados na rede.

Um diretório é uma listagem dos arquivos disponíveis para os usuários da rede. Esse diretório permitirá a um usuário de qualquer nó determinar em que lugar da rede um arquivo compartilhável existe. Se assume, cada computador tem seu próprio diretório local que contêm todos os arquivos compartilháveis nesse computador. Para localizar um arquivo que está pre

sente no computador local, o usuário deve consultar o arquivo diretório.

Existem vários modos de fazer um projeto de Diretório de BDD, Chu W. Wesley [13] propõe os seguintes:

(1) Centralizado;

(2) Local;

(3) Distribuído.

Existem ainda combinações desses modos:

(1) Diretório Central e Local;

(2) Diretórios distribuídos particionados;

(3) Diretório distribuído particionado_e central;

(4) Localização pelo nome do arquivo.

3.5.1 - DIRETÓRIO CENTRALIZADO

O diretório centralizado é o mais facilmente criado. Todos os arquivos da rede são descritos em um único nó, isto é, o diretório está em um único nó.

Quando um usuário requer um arquivo que não está armazenado em seu diretório local, presumindo que cada nó tem um diretório, ele consulta o diretório centralizado, para achar a locação ou conteúdo desse arquivo. CHU [13] faz que existem custos de comunicação para cada uma das operações e diz que o custo de operação desse sistema por unidade de tempo é:

$$C_0 = C_c + C_t + C_a$$

onde

C_0 - Custo de Operação

C_c - Custo de Comunicação, para consultas e atualização do diretório

C_t - Custo de Tradução, devido à não uniformidade na representação das informações, nas transações executadas por todos os nós do diretório.

C_a - Custo de armazenamento do arquivo diretório

Vantagens:

- Facilidade de manutenção do diretório.
- Facilidade de localizar arquivos (só existia um diretório)

Desvantagens:

- Custo de comunicação para acessar o diretório;
- Volume de comunicação muito grande para o nó que contém o diretório;
- Dificuldade de recuperação do diretório em caso de de destruição do mesmo; uma falha deste nó implica a paralisação da rede.
- Espaço necessário para armazenar o diretório (principalmente quando existem muitos arquivos na rede).

3.5.2 - DIRETÓRIOS LOCAIS

Não existe um diretório central, existindo apenas diretórios locais a cada nó da rede.

Qualquer pedido a arquivos que não se encontrem em um nó local implica que ações especiais devem ser providenciadas para localizar esses arquivos.

Ohama I. [39] indica duas ações para localizar arquivos.

1a. Ação: Enviar pedidos a todos os nós, perguntando se existe o arquivo desejado. Após o recebimento das respostas dos nós consultados, deve ser solicitado processamento para um nó em que o arquivo exista. Ou então deve ser solicitada uma cópia deste arquivo.
Se mais de um nó contiver o arquivo, então a solicitação deve ser enviada ao nó mais próximo do nó solicitante para a resposta mais rápida e com menor custo de comunicação. Em redes com dois sentidos de propagação deve ser enviado um pedido em cada sentido. Se os pedidos forem afirmativos, selecione o pedido confirmado pelo nó mais próximo.

2a. Ação: Enviar o pedido para o nó mais próximo deste e assim sucessivamente, até que seja encontrado o arquivo.

Se o pedido fechar o circuito da rede,então

o arquivo não existe.

Vantagens:

- Arquivos locais são rapidamente encontrados;
- Fácil manutenção do diretório local;
- Menor espaço ocupado pelo diretório local em relação ao central.

Desvantagens:

- O custo de operação, devido ao custo de comunicação para pesquisar a localização de um arquivo não local, pode ser muito alto. O custo de operação pode ser representado (6) mediante a seguinte equação:

$$C_0 = C_c + C_t$$

C_c: custo de comunicação; C_t: custo de Translação

- No percurso de atualização existem dificuldades para se saber onde estão as cópias de arquivos;
- A inexistência de um arquivo na rede implica o custo máximo de pesquisa;
- Dificuldade de recuperação do diretório em caso de falhas.

3.5.3 - DIRETÓRIO DISTRIBUÍDO

Neste esquema de diretório, cada nó tem uma cópia de todo o diretório da rede.

Pedidos para quaisquer arquivos são endereçados rapidamente e pedidos para arquivos que não existem localmente também são solucionados rapidamente.

Vantagens:

- Rápida localização de arquivos;
- A inexistência de arquivos e facilmente verificada;
- Facilidade de recuperação dos diretórios em caso de falhas.

Desvantagens:

- O custo de operação do sistema [13] por unidade de tempo de cada um dos nós é igual ao custo de operação no caso do Diretório Centralizado. Isto significa um custo de operação muito alto devido: 1º) ao armazenamento do diretório central em cada nó, 2º) aos custos de comunicação para a atualização desses diretórios.
- Custos inerentes à manutenção das cópias do diretório atualizado.

Ohana [38] afirma que talvez este esquema de diretório seja o mais adequado para redes de dados distribuídos, prin

cipalmente quando os arquivos são pouco dinâmicos.

3.5.4 - COMBINAÇÕES E EXTENSÕES DOS DIRETÓRIOS

3.5.4.1 - DIRETÓRIO CENTRAL E LOCAL

Num diretório centralizado, todos os arquivos são descritos em um único nó da rede. Arquivos locais a qualquer nó só podem ser localizados consultando-se o diretório central.

Uma variação do esquema de diretório centralizado, é obtida fazendo-se que todos os arquivos sejam descritos em diretórios locais e um nó qualquer da rede implementa-se o diretório central de toda a rede. Com isto, a definição de diretório centralizado apresentada se deve ao sentido da palavra centralização, pois na definição de Chu [13] os nós têm diretórios locais, o que deixa de ser uma centralização.

Nesta variação uma vantagem é que arquivos locais não são pesquisados no diretório central, só sendo pesquisado arquivos não locais aos nós. Outra vantagem é a facilidade de recuperação dos diretórios (locais e central).

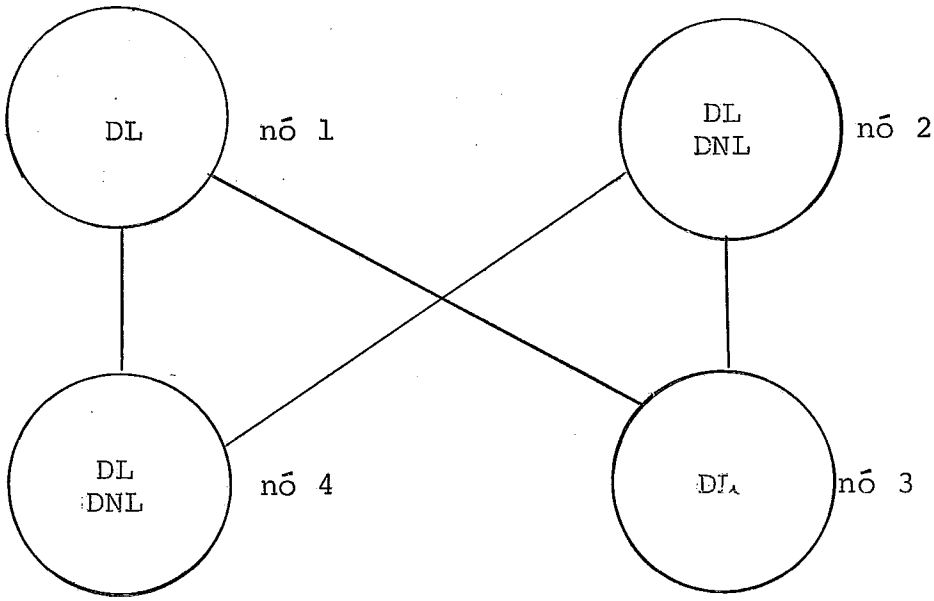
A desvantagem é que todas as alterações em arquivos (mesmo sendo locais) implicam a atualização do diretório central. Também é importante observar que as vantagens e desvantagens dos diretórios locais e centralizados devem ser consideradas.

3.5.4.2 - DIRETÓRIO CENTRALIZADO ESTENDIDO

Num diretório centralizado estendido, uma vez que o usuário acha a locação ou descrição de um arquivo, ele pode anexar esta informação ao seu diretório local. Se o usuário precisa deste arquivo posteriormente, a informação pode ser obtida do seu diretório local, reduzindo os custos de comunicação, assim como o tempo para fazer consultas ao Diretório Central. Não obstante, quando a informação desse arquivo no diretório central é atualizada, também se requer atualização da informação dos arquivos nos diretórios locais.

3.5.4.3 - DIRETÓRIOS DISTRIBUÍDOS E PARTICIONADOS

Nesta variação, cada nó da rede teria um diretório local e um diretório dos arquivos não locais acessados por um nó.



Este esquema oferece todas as vantagens dos diretóri os locais e tem como principal vantagem a rápida localização de arquivos não locais. Uma desvantagem é a dificuldade de atualização dos diretórios local e não local de um nó. Outra desvantagem é a dificuldade de localização das cópias de arqui vos na rede.

3.5.4.4 - DIRETÓRIO DISTRIBUÍDO PARTICIONADO E CENTRAL

Nesta variação, além do diretório particionado em ca da nó, existe um diretório central localizado em um nó qual quer da rede. As vantagens e desvantagens dos diretórios cen tral e distribuído são consideradas.

3.5.4.5 - LOCALIZAÇÃO PELO NOME DO ARQUIVO

Booth [38] propôs um método para localizar arquivos em redes de computadores. O nome do arquivo pode ser usado pa ra indicar a localização deste arquivo, prefixando-se o mesmo com um código de localização.

Este prefixo é fornecido por "software" durante a criação do arquivo e transmitido ao usuário como parte inte grante do nome do arquivo.

Vantagens:

- A localização do arquivo é imediata, não sendo ne cessário processamento extra;

- O prefixo pode ser usado para localizar manualmente o arquivo e pedidos podem ser enviados para a localização correta, automática ou manualmente.

Desvantagens:

- A transferência de arquivo para outro nó, implica a troca do nome do arquivo (troca de prefixo).

3.6 - EXEMPLOS DE IMPLEMENTAÇÕES

3.6.1 - EXEMPLO 1: SISTEMA SDD-1

O SDD-1 [2,16], sistema de Banco de Dados Distribuídos, que está sendo desenvolvido pela "Computer Corporation of América" usa os métodos de particionamento e replicação já explicados para obter unidades de distribuição de dados através do sistema.

O banco de dados SDD-1 consiste de relações lógicas [43] cada uma particionada em sub-relações chamadas fragmentos lógicos, os quais são unidades de distribuição de dados. Fragmentos lógicos são definidos em dois passos.

Primeiro, a relação é particionada horizontalmente em subconjuntos definidos por restrições simples. A Fig. 3.3 mostra um exemplo com a relação CLIENTE e tendo como restrições o número da locação e a quantidade de empréstimo.

CLIENTE

	NOME	LOCAÇÃO	≠ CONTA	RESERVAS	ATIVO	EMPRÉSTIMOS	TID
Clien-1	Paulo	1	1234	$ 100	$ 200	- $ 8	TID 1
Clien-2	Francisco	2	5678	$ 200	$ 300	$ 30.000	TID 2
Clien-3a	Carlos	3	9012	$ 1000	$ 0	$ 20.000	TID 8
Clien-3b	Ana	3	3456	$ 100	$ 50	$ 0	TID 10

Clien-1 = Cliente onde locação = 1

Clien-2 = Cliente onde locação = 2

Clien-3a = Cliente onde locação = 3 e empréstimo $\not\$$ $\neq 0$

Clien-3b = Cliente onde locação = 3 e emprestimo $\not\$$ $= 0$

Fig. 3.3 - PARTICIONAMENTO HORIZONTAL

Logo, após o particionamento horizontal, cada um desses subconjuntos (CLIEN-1, CLIEN-2, ...) é particionado em subrelações definidas por projeções. Para reconstruir a relação lógica a partir de seus fragmentos, um identificador de tupla único (TID) é agregado a cada uma das tuplas e incluído em cada fragmento. Além disso, o TID permite acesso direto às tuplas, quando são armazenados em dispositivos de memória secundária [6].

A Fig. 3.4, mostra o particionamento vertical a partir da figura anterior.

CLIENTE

	NOME	LOCAÇÃO	CONTA	RESERVAS	ATIVO	EMPRÉSTIMOS	TID
CLIENT-1	CLIENT-	1.1	CLIENT- 1.2				TID1
CLIENT-2	CLIENT-	2.1		CLIENT- 2.2			TID2
CLIENT-3a	CLIENT-	3a.1	CLIENT-3a.2	CLIENT-3a.3			TID8
CLIENT-3b	CLIENT-	3b.1	CLIENT-3b.2	CLIENT-3b.3		CLIENT-3b.4	TID7

Fig. 3.4 - PARTICIONAMENTO VERTICAL

CLIENT- 1.1 = CLIENT-1 (nome,locação)

CLIENT- 1.2 = CLIENT-1 ($\neq$ conta, reservas, ativos, empréstimos)

CLIENT- 2.1 = CLIENT-2 (nome, locação, $\neq$ conta)

⋮

Os fragmentos lógicos são as unidades de distribuição dos dados, significando que cada um deles pode ser armazenado em um ou vários nós do sistema. Os fragmentos lógicos são definidos e a atribuição de fragmentos aos nós do sistema é feito no momento da concepção lógica do projeto de Banco de Dados.Uma cópia armazenada de um fragmento lógico é chamada fragmento armazenado. Cada um dos fragmentos pode ser armazenado redundantemente em mais de um nó [43].

Os usuários fazem referência às relações não aos fragmentos. É responsabilidade do SDD-1 traduzir relações em fragmentos lógicos.

3.6.2 - EXEMPLO 2: SISTEMA DO I.E.I. (ITÁLIA)

Para mostrar um segundo exemplo de distribuição de da

dos apresenta-se um sistema relacional de BDD que está sendo implementado no "Instituto de Elaborazione della Informazione (I. E.I.) - Pisa-Itália" [8]. O sistema, no início, será suportado por uma rede de três minicomputadores:

PDP-11 séries 70

PDP-11 séries 34

PDP-11 séries 40

As relações, segundo o critério de distribuição de dados, são divididas em:

- relações locais;

- relações distribuídas.

Relações locais são relações criadas num nó só e acessíveis apenas a este nó.

Relações distribuídas são relações particionadas e geograficamente localizadas sobre os nós através da rede.

Relações distribuídas são particionadas horizontalmente usando-se um predicado de distribuição. Esse predicado é uma restrição imposta a um atributo da relação, de modo que uma tupla possa ser identificada individualmente.

Note-se que se usa o mesmo critério do SDD-1.

Por exemplo, na Fig. 3.5, a relação LIVRO é dividida de acordo com o predicado: LIVRARIA = "CONSTANTE"

LIVRO

CÓDIGO	TÍTULO	AUTOR	ASSUNTO	LIVRARIA	
				I.E.I. I.E.I.	nó 1
				S.N.S. S.N.S.	nó 2
				I.S.I. I.S.I.	nó 3
				I.N.F.N. I.N.F.N.	nó 4

Fig. 3.5 - RELAÇÃO LIVRO - DISTRIBUIÇÃO DE DADOS

As tuplas que satisfazem o predicado LIVRARIA = "I.E.I." são localizados no nó 1; aquelas que satisfazem o predicado LIVRARIA = "S.N.S." estarão no nó 2, etc.

A relação é assim dividida em partes chamadas fragmentos os quais não têm nenhuma tupla em comum e é possível reproduzir a relação original sem redundância.

O predicado de distribuição é uma restrição simples como no sistema SDD-1, i.e. uma expressão booleana da forma:

$$\langle \text{nome do atributo} \rangle \quad \langle \text{operador relacional} \rangle \quad \text{constante}$$

onde $\langle \text{operador relacional} \rangle$ é $=$, $>$, $<$, etc.

Os dois exemplos anteriores são baseados no modelo relacional e muitos autores (17,39) afirmam que o modelo relacional é a melhor forma de modelagem para o Banco de Dados Distribuidos.

3.6.3 - ALOCAÇÃO DE DIRETÓRIOS NO SISTEMA DE BANCO DE DADOS DISTRIBUÍDOS SDD-1

O SDD-1 mantém diretórios [2] que contêm as definições de fragmentos e relações, a localização dos fragmentos e estatísticas de uso. Esse sistema trata os diretórios como se fossem arquivos simples de usuários. Esta consideração permite fragmentar os diretórios exatamente como os outros dados. Esses segmentos são armazenados de maneira distribuída e redundante, através do sistema.

Mas existem alguns problemas de uso. Primeiro, a performance pode ser degradada pelo requerimento de que cada acesso ao diretório aumente o custo de processamento das transações e que cada acesso a diretórios remotos incorra em demora de comunicação. Esses problemas são evitados com o uso de uma memória tipo cache que permite o armazenamento dos fragmentos de diretório mais recentemente usados em cada módulo administrador de transações. Também é necessário ter um diretório que informe onde cada fragmento do diretório é armazenado. Para isso existe um Localizador de Diretórios ("Directory") com uma cópia armazenada em cada módulo administrador do Banco de Dados.

IV. - PROCESSAMENTO DE CONSULTAS EM BANCO DE DADOS DISTRIBUIDOS

4.1 - INTRODUÇÃO

Processamento de consultas num Banco de Dados Distribuídos corresponde à tradução de requerimentos formulados numa linguagem de alto nível sobre um nó da rede, numa seqüência de instruções elementares, as quais recuperam dados armazenados no Banco de Dados Distribuídos [7]. Para outros autores [40] uma consulta é um acesso requisitado, feito por um usuário ou programa no qual um ou mais arquivos necessitam ser acessados. Cada consulta formulada por um usuário tem de ser separada em sub-consultas executáveis pelos diferentes SGBD locais. Além disso, a partir dos resultados das sub-consultas o SGBDD resolve um conjunto de operações lógicas / ex. AND, OR, etc... / expressados na consulta. O resultado dessas operações constitui o resultado final que retorna ao usuário.

Uma possível arquitetura do software para processamento de consultas se apresenta na Fig. 4.1. Nota-se que a evolução da consulta corresponde à sequência de traduções até o esquema interno [7]. A arquitetura do software global pode ser sub-dividido em duas partes ligadas por uma rede de comunicações. A parte inferior da arquitetura corresponde às funções do SGBD local e a parte superior inclui funções de supervisão, distribuição e controle das operações diretamente relacionadas ao ambiente distribuído.

Os SGBD's locais executam funções as quais são características de um BD centralizado. No SDD-1 [4] as duas partes são chamadas LDM ("Local Data Manager") e GDM ("Global Data Manager").

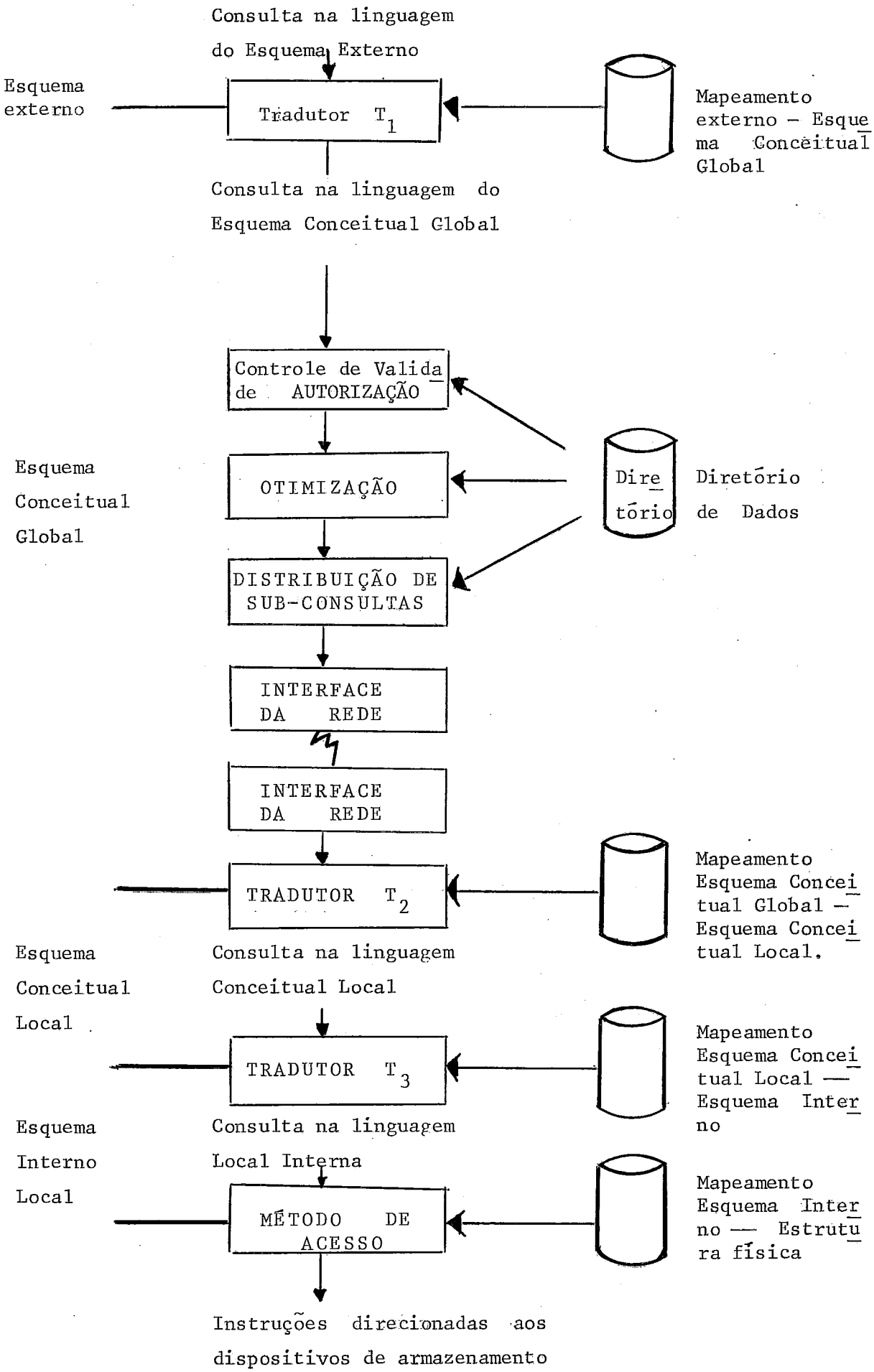


Fig. 4.1 - ARQUITETURA GLOBAL DO SOFTWARE PARA PROCESSAMENTO DE CONSULTAS NUM BANCO DE DADOS DISTRIBUÍDO:

O processamento de consultas inclui um módulo de otimização cujo objetivo é escolher uma sequência de sub-consultas e transferência de dados (e das cópias dos dados, no caso de replicação) a serem usadas com a finalidade de minimizar uma certa função de custo. Um exemplo de função de custo é [7] a média ponderada dos seguintes fatores: tempo de resposta, tráfego gerado na rede e uso de recursos locais (CPU, I/O, etc.). A saída do módulo de otimização é um programa que inclui provavelmente acessos a serem realizados em nós remotos e transferências de dados de um nó para outro.

A parte superior da arquitetura do software é usada no nó onde a consulta é feita. Sua finalidade é a de analisar cada consulta e produzir uma sequência ótima de comandos operacionais. Cada um dos comandos é também analisado pelos SGBD locais dos computadores remotos e a estratégia local ótima de recuperação de dados é executada.

4.2 - PROCESSAMENTO DE CONSULTAS EM BANCO DE DADOS DISTRIBUÍDOS HOMOGÊNEOS/HETEROGÊNEO

Como foi dito no capítulo I, os bancos de dados distribuídos podem ser classificados como homogêneos e heterogêneos.

No caso do Banco de Dados Homogêneo se requer um tradutor idêntico T_2 em cada um dos nós (Veja-se Fig. 4.1). O esquema conceitual global pode ser especificado na mesma linguagem do esquema conceitual local. O tradutor T_2 é simples [7].

Por outro lado, no caso heterogêneo a complexidade do

tradutor T_2 aumenta devido à presença de diferentes SGBD locais. Além disso, o esquema conceitual global deve ter uma grande capacidade de modelagem e flexibilidade e deve ser maior do que cada esquema conceitual local.

O problema de processamento de consultas num sistema heterogêneo é afetado pela complexidade dos mapeamentos, assim como das diferentes representações e estratégias de acesso as quais devem convergir para uma única função objetivo.

No resto deste estudo, problemas relacionados à heterogeneidade dos esquemas de dados não serão considerados e o modelo relacional, devido a sua simplicidade e aplicações atuais, será usado para descrever o esquema conceitual e local. O DDBMS mantém diretórios para que cada consulta sofra um mapeamento quanto à localização dos dados requeridos.

Nos referiremos a uma relação como "local" se ela é armazenada inteiramente num nó, e "distribuída" se partes dela (i.e. fragmentos) são armazenadas em nós diferentes.

Uma consulta num modelo relacional consiste de duas partes. A primeira especificando o(s) domínio(s) da relação a ser recuperada e a segunda especifica o predicado, o qual é uma qualificação representando as propriedades que se deseja do conjunto a acessar. A complexidade de uma consulta aumenta com o número de operadores lógicos AND e OR,e de operadores de comparação $<$, $>$, $=$, $\neq$ incluidos na qualificação.

Seja S uma relação, representando fornecedores,com domínios S # Snome, Status, Cidade; e SP outra relação,representando fornecimentos, com domínios S #, P #, quantidade. As consultas ao Banco de Dados relacional que contêm essas relações podem ser classificadas nas seguintes categorias [15]:

A) Operações de recuperação

A.1) Recuperações simples - o predicado representando a propriedade do conjunto a ser recuperado é definido sobre a mesma relação.

Exemplo: GET (S.Snome): S. cidade = "Paris"
AND S. status > 10

A.2) Recuperações múltiplas - O predicado é definido sobre múltiplas relações.

Exemplo: GET (S. snome): (SP. S # = S. S #
AND SP. P # = "P2")

B) Operações de Atualização

B.1) Atualização simples - Só se atualiza os valores de uma tupla específica de uma relação.

```
UPDATE SP
SET  S #  =  "S1"
   Qtdade  =  200
WHERE P #  =  "P2"
   AND       S #  =  "S2"
```

B.2) Atualização múltipla - Se atualiza os valores de múltiplas tuplas.

```
UPDATE  S
SET STATUS = 2 X STATUS
WHERE CIDADE  =  "RJ"
```

B.3) Adição de tuplas

Ex.: INSERT INTO SP

<"S5", "P5", 200>

B.4) Eliminação de tuplas

Ex.: DELETE S

WHERE S# = "S1"

4.3 - FATORES CRÍTICOS NO PROCESSAMENTO DE CONSULTAS

Um dos mais importantes problemas no processamento de consultas num ambiente distribuído é a eficiência. Dentre dos fatores críticos que podem afetar essa eficiência temos primeiramente a transferência de dados entre nós. Esta transferência deve ser minimizada. Esse é um dos objetivos principais no sistema SDD-1 [4] e na maioria dos outros sistemas [10,46]. Para atingir esse objetivo emprega-se técnicas de otimização para separar consultas múltiplas em tantas sub-consultas simples quanto possível de modo que o movimento de relações de um nó para outro seja mínimo.

Em segundo lugar, o custo de transmissão de dados entre dois nós é definido [26] como uma função linear $C(x) = C_0 + C_1(x)$, onde x é a quantidade de dados transmitidos. Alguns autores [26] definem este custo em unidades de tempo, outros [4,46] como quantidade de bytes.

A constante C_0 representa o tempo inicial para cada uma das transmissões separadamente. Outra importante propriedade de C(x) é que se $x \leqslant y$, então $C(X) \leqslant C(Y)$. Além do mais, ob

serve-se que o custo de transmissão é significantemente maior do que o custo de processamento local [10,26] e do que a transferência de dados entre os dispositivos de armazenamento de um nó [26]. Contudo, a distribuição do sistema faz possível o processamento em paralelo das subconsultas e das operações lógicas o qual contribui na diminuição do tempo de resposta [10].

Por último, o tráfego na rede constitui outro fator crítico devido a tecnologia de comunicações de dados não ter atingido o mesmo nível em termos de redução de custo e aumento de performance [7] como se observa na tecnologia de computadores.

4.4 - ESTRATÉGIAS DE PROCESSAMENTO DE CONSULTAS EM BANCO DE DADOS DISTRIBUIDOS

O acesso aos dados que estão espalhados sobre os nós de uma rede implica necessariamente a transmissão de dados sobre linhas de comunicação introduzindo assim tempos consideráveis. O SGBDD deve considerar esses fatos e decidir de forma eficaz quanto ao processamento local e às transmissões dos dados com a finalidade de processar eficientemente consultas distribuídas.

Essa ordem de processamento de dados e transmissões é chamada estratégia de processamento de consultas. Podemos considerar dois tipos de estratégia: 1º) Estratégia Centralizada e 2º) Estratégia Distribuída.

4.4.1 - ESTRATÉGIA DISTRIBUÍDA DE PROCESSAMENTO DE CONSULTAS

Nesta estratégia se distribuem, para certa consulta, todas as respostas parciais das sub-consultas, executando-se as operações lógicas nos diferentes nós de distribuição. A resposta final é gerada como produto da união dos diferentes processamentos locais nos diferentes nós. Assim, se considerarmos um modelo relacional o resultado da consulta será uma relação distribuída nos diferentes nós de processamento.

Esta estratégia é usada no sistema INGRES Distribuido (Interactive Graphics and Retrieval System) [45,46].

4.4.1.1 - ALGORITMO DE DECOMPOSIÇÃO

O modelo a tratar tem sido implementado como parte do trabalho realizado na extensão do "INGRES" Distribuído [45,46].

O sistema distribuído que suporta a implementação e considerado homogêneo, por esta razão não existe preocupação com a possível tradução entre diferentes modelos de dados. Os usuários do INGRES interagem com ele através da linguagem QUEL ("Query Language" - Linguagem de Consulta). O QUEL aceita os seguintes comandos: RETRIEVE (recuperação); APPEND (para acrescentar tuplas), DELETE (eliminação de tuplas) e REPLACE (modificações).

O banco de dados é considerado como uma coleção de relações $R_1, R_2, \ldots R_n$, sendo que qualquer relação R_i pode ser armazenada em um único "local", como também pode aparecer espalha

da em diferentes nós da rede, em cujo caso é considerada "DISTRIBUIDA". O critério de distribuição dos fragmentos nos diferentes locais, será o lugar onde as tuplas vão ser processadas, ou um local predestinado para uma dada relação R_i.

O algoritmo para decompor uma consulta tem as seguintes entradas:

- A parte conectiva da consulta (Ex. uma qualificação vista como uma coleção de cláusulas separadas por AND's).
- A localização de cada fragmento e sua cardinalidade
- O tipo de rede: "NODO A NODO" ou "DIFUSÃO".
  No caso de comunicação NODO A NODO presume-se que o custo de transmissão por byte desde um nodo X a um outro qualquer é fixo. Na forma por "difusão" o custo de transmissão de um nó para o resto também é considerado igual ao custo de transmitir de um nodo para um outro só.

O local particular onde é originada a consulta é chamado "MESTRE". O nó mestre no INGRES se comunica com um nó "ESCRAVO" em cada local envolvido no processamento da consulta, mediante dois tipos de comandos:

- Executar a consulta Q (local)
- Mover o fragmento R_i (Local) da relação R para um subconjunto de locais da rede (S_1, S_2..... S_n).

O método do algoritmo é o seguinte:

PASSO 1: Executar todas as consultas parciais ou subconsultas que contenham só uma variável ar

gumento.

Como exemplo de consultas parciais com só uma variável argumento, temos a seguinte:

```
SELECT  S. sno
into  temp
where  S. nome  =  "XYZ"
```

Nesta consulta parcial só se usa a variável S como argumento.

PASSO 2: Se houver alguma consulta parcial de uma variável que for falsa em todos os nós (passo 1), então a consulta inteira é falsa e o processo termina.

PASSO 3: Aplicar algoritmo de redução [51]. Mediante o qual a consulta original é decomposta numa sequência de sub consultas ou componentes irreduzíveis, cada um dos quais processado de forma independente. Se diz que uma consulta Q é reduzível se ela pode ser substituída por duas subconsultas (Q' e Q"), que tenham em comum uma só variável, i.e.,

$$Q\ (X_1, X_2, \ldots, X_n) \longrightarrow Q'\ (X_m, X_{m+1}, \ldots, X_n)$$

seguido por:

$$Q''\ (X_1, X_2, \ldots, X'_m)$$

onde X'_m é o resultado da relação Q'

Por exemplo a consulta Q:

```
SELECT  S.  Snome
into    temp  3
from S  (Fornecedor)
     Y  (Fornecimento)
     P  (Peças  )
WHERE S.  s #  = Y.  s #
AND   Y.  p #  = P.  p #
```

A consulta acima possui Y como variável argumento comum da cláusula de qualificação, podendo ser reduzida nas subconsultas Q' e Q"" abaixo:

```
Q'  SELECT  Y. s #  From Y (fornecimento)
     into   TEMP
    WHERE   Y. p #  =  P. p #
Q"  SELECT  S. snome from  TEMP
    WHERE   S. s #  =  TEMP.  s #
```

Note-se que a consulta Q foi decomposta em duas subconsultas parciais Q' e Q" com base no fato de ter uma variável simples comum às duas cláusulas de qualificação. Esta técnica é chamada [51] de SEPARAÇÃO ou DISJUNÇÃO DA CONSULTA ("Detachment").

PASSO 4: Escolher a "peça" seguinte. Onde uma "peça" é uma sub consulta parcial. Se não houver mais sub consultas, pare; caso contrário, es

colha uma subconsulta Q (respeitando a ordem ditada pela redução).

PASSO 5: Se a "peça" pode ser executada em diferentes nós sem necessidade de mover partes de relações, então vá para o passo 9.

PASSO 6: Selecionar o(s) nó(s) que irá(ão) processar a próxima peça da consulta. A seleção é baseada na estrutura da consulta e no tamanho e na localização dos fragmentos. Esssa seleção faz parte da tarefa de otimização do algoritmo de processamento que tenta atingir principalmente dois objetivos:

1º) Minimização do tempo de resposta. Isto implica em minimizar a quantidade de processamento necessário para solucionar a consulta, usando para tal, o maior paralelismo possível dos vários nós de computação; e

2º) Minimizar o tráfego na rede. Isto implica em transmitir uma quantidade mínima de dados necessários para atender a consulta.

Assim, pode-se afirmar [7] que o aumento de tráfego da rede pode melhorar o tempo de resposta, caso tenham um processamento em paralelo maior.

PASSO 7: A subconsulta deve envolver duas ou mais variáveis para poder alcançar este passo. Para processar consultas de n variáveis, fragmentos de n - 1 relações devem ser movidos e a relação restante deverá permanecer fragmentada. Cada nó que processará os fragmentos deverá ter uma cópia das n - 1 relações. Se o processamento é feito em apenas um nó, este deverá ter cópia de todas as n relações.

PASSO 8: Mover os fragmentos selecionados para os nós selecionados. Cada nó será direcionado a enviar uma cópia de seus fragmentos selecionados para os nós selecionados no passo 6.

PASSO 9: O "mestre" agora difunde a consulta para os nós selecionados e espera por todos os nós acabarem.

PASSO 10: O processamento continua retornando para o passo 4.

4.4.1.2 - EXEMPLO DE PROCESSAMENTO DE CONSULTAS NUMA ESTRATÉGIA DISTRIBUÍDA

Considere-se o seguinte Banco de Dados Distribuído com consultas processadas pelo sistema INGRES Distribuído.

Temos os seguintes esquemas:

Esquema Conceitual Global:

S (Fornecedor) = (Sno, Snome, Scidade)

J (Fornecimento) = (Sno, Jno, Jquanti)

Esquemas Conceituais Locais:

Nó 1: S1 (Fornecedor 1), onde

S1 (Fornecedor 1) = S (Fornecedor) [Cidade = "Rio"]

Nó 2: J1 (Fornecimento 1), onde

J1 (Fornecimento 1) = J (Fornecimento)

S2 (Fornecimento 2), onde

S2 (Fornecedor 2) = S (Fornecedor) [Scidade = "SP"]

Nó 3: S3 (Fornecedor 3), onde

S3 (Fornecedor 3) = S (Fornecedor) [Scidade ≠ Rio] and

S (Fornecedor) [Scidade ≠ "SP"]

Considere a seguinte subconsulta:

```
Q:  Select J. jnúmero
      into W
      from Fornecimento J, Fornecedor S
      where J. Sno = S. Sno
      and S. S.cidade = "XYZ"
```

i.e. Achar os números dos fornecimentos dos fornecedores da cidade "XYZ".

O algoritmo de decomposição geraria os seguintes passos:

PASSO 1: Note-se que só existe uma sub consulta com uma variável simples, que será a primeira a executar-se: S. Scidade = "XYZ" tem só a variável S como argumento.

```
Q' Select   Sno   into   temp 1
      from  Fornecedor
      where S. Scidade = "XYZ"
```

como S(Fornecedor) = S1 (Fornecedor 1) U S2(fornecedor 2)
U S3(Fornecedor 3)

Três sub consultas deveriam ser geradas a partir de Q'. Mas como XYZ ≠ "Rio e "SP", se poderia reduzir Q' a:

```
Q":  Select J. jno
       into Templ
       from S3 (Fornecedor 3)
       where Scidade = "XYZ"
```

e a consulta original agora fica da seguinte forma:

```
Q1   Select J. jno
       into temp2
       from Fornecimento J. Templ.S
       where J.sno = Templ. Sno
```

PASSO 2: Se Temp 1 (relação temporal) que criou uma subconsulta com uma variável simples, for vazio, i.e. não existem duplas que possam responder Q" em qualquer um dos nós, a consulta Q é falsa e o processo acaba.

PASSO 3: Aplicar algoritmo de redução [51]

Mediante este algoritmo se separa a consulta original em subconsultas, cada uma das quais é processada em ordem independentemente.
Na consulta Q_1, não se tem nenhuma variável argumento comum e portanto, este passo não tem efeito.

i.e. Q1 e irredutível

PASSO 4: Escolher a "seguinte peça" da consulta.

i.e. uma "peça".e uma sub-consulta parcial.
No exemplo, Q_1 só tem uma cláusula e assim só existe uma "peça": Where J. sno = TEMP1. Sno.

PASSO 5: Q_1 não pode ser processado em diferentes nós sem necessidade de mover partes das relações, pois Fornecimento J está no nó 2 e TEMP1 S está no nó 3. Logo, os dados devem ser movimentados para execução.

PASSO 6: Existem as seguintes alternativas básicas para movimentar os dados:

. Mover alguns fragmentos de TEMP1.S para o nó 2 e mover uma cópia de todos os fragmentos de Fornecimento J para o nó 3.

. Mover alguns fragmentos de FORNECIMENTO J para o nó 3 e mover uma cópia de todos os frag

mentos de TEMP1 para o nó 2.

. Mover todos os dados (TEMP1 S e FORNECIMENTO J) para um nó final e resolver a consulta mediante um processamento local no nó final.

Os nós selecionados serão os nós 2 e 3 em base a eleição da primeira alternativa.Cada um desses nós contém fragmentos de TEMP1 e uma cópia da relação FORNECIMENTO J.

PASSO 7: O processamento da consulta se faz nos nós 2 e 3 e cada um deles produzirão como resposta um fragmento local W. A resposta final será a relação distribuída W.

4.4.2 - ESTRATÉGIA CENTRALIZADA DE PROCESSAMENTO DE CONSULTAS

As estratégias centralizadas consistem, primeiramente, em agrupar para uma determinada consulta todos os resultados parciais das sub-consultas e logo executar juntas todas as operações lógicas num simples nó.

Em conseqüência, a característica principal desta estragégia é que na fase final só se inclui um processamento local da consulta. Isto é depois de se ter num único nó a reunião de todos os dados distribuídos.

Com a finalidade de reduzir o tamanho do Banco de Dados os dados de duas ou mais relações devem ser combinados. As

estratégias centralizadas para atingir esse objetivo, usam os seguintes operadores:

Seja R: relação no BDD; A, B, ...: Domínios da relação;

attr(R): atributos da relação; K: Valor específico do domínio

- Restrição: $R\left[A = K\right] = \{ r \in R \;/\; r.A = K \}$

onde r.A é o valor no domínio de A da tupla r

- Projeção

$$R\left[A_1, A_2, \ldots\ldots, A_n\right] = \{ \langle r.A_1, r.A_2, \ldots\ldots r.A_n \rangle \;/\; r \in R \}$$

- Junção

$$R\left[A = B\right] S = \{ \langle r.s \rangle \;/\; r \in R,\ s \in S \text{ e } r.A = S.B \}$$

A estratégia centralizada é usada no sistema SDD-1 (4) e um esboço do algoritmo será descrito a seguir:

4.4.2.1 - ALGORITMO SDD-1 DE PROCESSAMENTO DE CONSULTAS

O algoritmo centralizado para processamento de queries usado no SDD-1 compreende os seguintes 3 passos (43):

PASSO 1 - MAPEAMENTO DA CONSULTA Q

A consulta é expressada em DATALANGUAGE (i.e. linguagem de procedimentos de alto nível mediante o qual o usuário do SDD-1 interage com o sistema) e é convertida a uma forma de cálculo relacional, chamada "ENVELOPE", que especifica um superconjunto do Banco de Dados necessários para atender a Q.

PASSO 2 - AVALIAÇÃO DO ENVELOPE

Neste passo se recupera o superconjunto do Banco de Dados especificado pelo envelope, reunindo posteriormente os resultados parciais num simples nó S_a.

Os envelopes são processados em duas fases: A primeira fase executa operações relacionais em vários nós do Banco de Dados Distribuído com a finalidade de delimitar um subconjunto do Banco de Dados que contém todos os dados relevantes desse envelope. Esse conjunto é chamado uma redução do Banco de Dados e o programa P, que contém as operações relacionais, produto do envelope, é chamado REDUTOR. A segunda fase transmite a redução a um nó previamente designado de acordo a certas condições de otimização.

PASSO 3 - EXECUÇÃO DA CONSULTA EM Sa

Usando os dados reunidos no passo 2. Assim, passo 3 inclui só processamento local da consulta. Esses 3 passos são apresentados na Fig. 4.2.

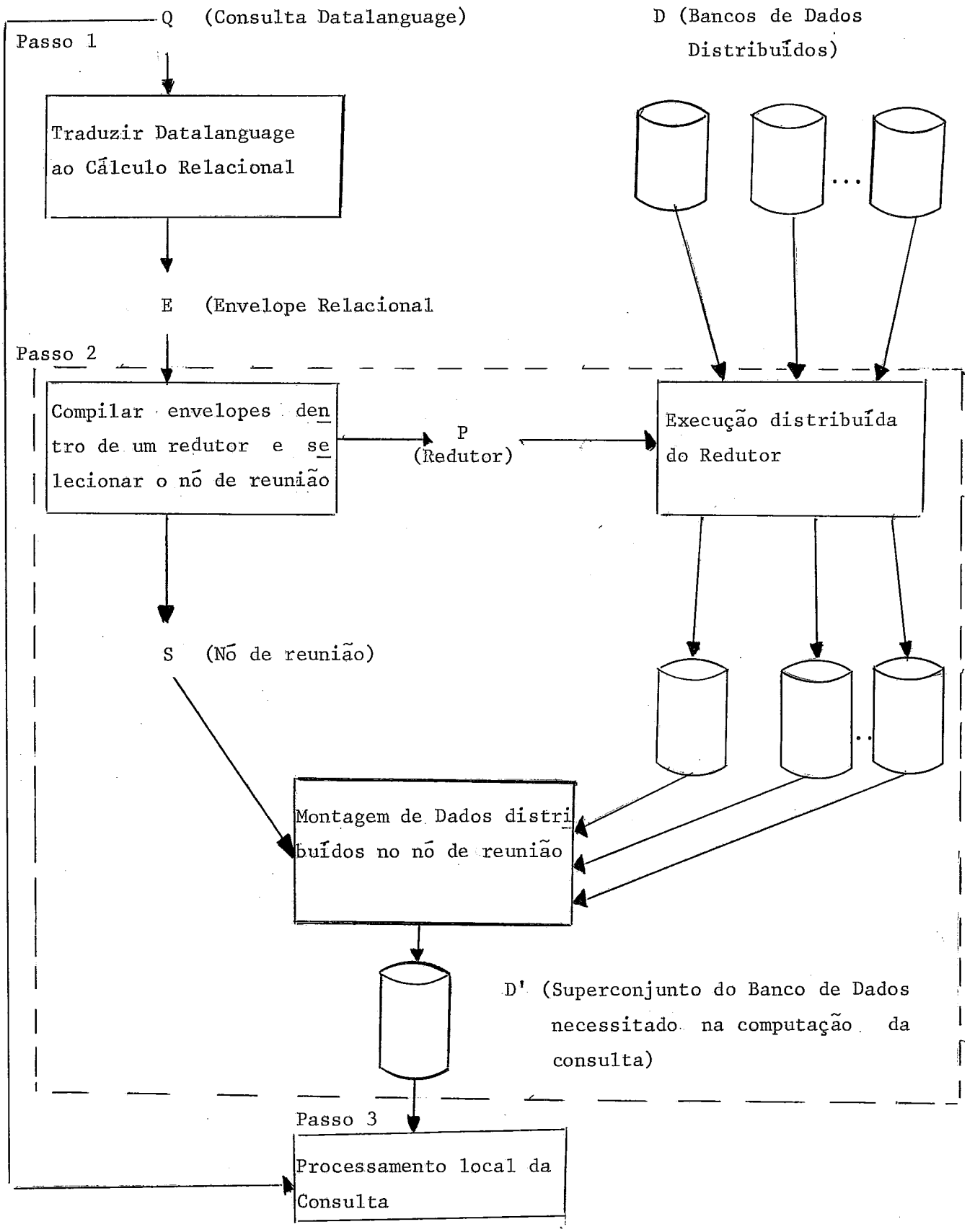


FIG. 4.2 - PRINCIPAIS PASSOS NO ALGORITMO DE PROCESSAMENTO DE CONSULTAS NO SDD-1.

De acordo com a Fig. 4.2 os elementos principais usados na estratégia de processamento de consultas no SDD-1 são: 1) o envelope, 2) o programa redutor.

O envelope é uma expressão em cálculo relacional que faz um mapeamento a partir do Banco de Dados a um sub banco de dados. Consiste de uma qualificação q e uma lista t_1, ... t_n. O termo q é uma fórmula booleana com cláusulas da forma $R_1.A = R_j.B$ ou $R_1.A = K$. (i.e. R significa relação, A, B significam atributos da relação R e K é uma constante ou valor do atributo).

Os termos $R_1.A$, $R_j.B$ são chamados variáveis de indexação. Cada um dos t_i é um conjunto de variáveis indexados por R_i, i.e. t_i e da forma $R_i.A_{i1}$, ..., $R_i.A_{il}$

Assim, um envelope é definido pela seguinte coleção de consultas [2]:

RETRIEVE INTO R'1 (t_1) where q

⋮

RETRIEVE INTO R'n (t_n) where q

Na realidade, existem muitos envelopes, mas para achar o melhor precisa-se resolver um problema de otimização que depende dos detalhes da linguagem de definição de dados (4).

O segundo elemento usado neste algoritmo é o redutor. Um redutor para o envelope E é um programa sequencial P de operadores relacionais tal que para todos os Bandos de Dados D, P(D) é uma redução de D com respeito ao E.

O principal objetivo do redutor é diminuir os custos

já que pela redução se eliminam dados desnecessários, reduzindo a transferência de dados inter nós para satisfazer a consulta. No processo de redução, além dos operadores de RESTRIÇÃO, JUNÇÃO E PROJEÇÃO se usa um operador chamado de semijunção ou metade de junção, definido da forma seguinte:

> Sejam R (a, B) e S (c, d) relações no BDD;
> a semijunção de R com S sobre uma qualificação q (e.g R.b = S.c) é igual à junção de R e S baseado em q, projetada sobre os atributos de R.
>
> Note-se que a semijunção de duas relações baseado em um atributo comum é igual à junção das duas relações sobre esse atributo comum projetado somente sobre o atributo da primeira relação.

4.4.2.2 - EXEMPLO DE PROCESSAMENTO DE CONSULTAS NUMA ESTRATEGIA CENTRALIZADA

SISTEMA SDD-1 - CONSULTAS EM DATALANGUAGE

Usuários interatuam com SDD-1 mediante consultas codificadas numa linguagem de procedimentos de alto nível (i.e. o usuário especifica um método passo a passo para atingir um resultado) chamada DATALANGUAGE. As Figs. 3.3 e 3.4 apresentam um Banco de Dados em SDD-1 e uma consulta de DATALANGUAGE.

Seja o Banco de Dados D seguinte:

S (S≠, nome, locação)		
1,	Acme,	MG
2,	Best,	MG
3,	Mid,	RJ
4,	Nadir,	SP

Y (S≠,	P≠,	qtdade)
1,	1,	20
1,	2,	50
3,	3,	50
4,	1,	10
4,	5,	75

P (P≠,	nome,	tipo)
1,	LSI	micro
2,	P11	mini
3,	360	G.Porte
4,	CRI	G.Porte
5	8080	micro

S descreve fornecedores

P descreve peças

Y descreve quem fornece, que peças e que quantidade

Presume-se que S, Y e P são alocados nos nós 1, 2, e 3 respectivamente.

Fig. 4.3 - EXEMPLO DE RELAÇÕES PARA O BANCO DE DADOS DO SDD-1

CONSULTA: Listar o nome do fornecedor, nome da peça, e quantidade fornecida para todas as peças fornecidas pelo fornecedor de MINAS GERAIS. Também, imprimir quantas destas peças se referem a mini-computadores.

```
CONSULTA  Q
BEGIN
   Count: = 0;
   For S
      If S.locação = "MG" then for Y
         If S.s≠   = Y.s≠ then for p
          If y.p≠  = P.p≠ then
            Begin
              Print S.nome, P.nome, Y.qtdade;
             If P.tip = "mini"
             then
               Count:= Count + 1;
            END;;;
PRINT "Número de minis e", Count;
```

Fig. 4.4 - EXEMPLO DE CONSULTA EM SDD-1

ENVELOPE: O envelope é o primeiro passo no processamento de queries e o objetivo é especificar um subconjunto de cada relação no Banco de Dados.

Os envelopes são expressados numa linguagem de Cálculo Relacional, similar à QUEL (25) e tem-se um exemplo na Fig. 4.5.

ENVELOPE E

RETRIEVE (S.s≠, S.nome,S.locação) Where qualificação

RETRIEVE (Y.s≠, Y.p≠, Y.qtdade) Where qualificação

RETRIEVE (P.p≠, P.nome, P.tipo) Where qualificação

Qualificação: S.locação = "MG"

S.s≠ = Y.s≠ Y.p≠ = P.p≠

Fig. 4.5 - ENVELOPE PARA A CONSULTA DA Fig. 4.3

O objetivo do envelope E é recuperar algum supercon junto do dado especificado por E. Por exemplo (Fig. 4.6):

S (s≠,	nome,	locação)
1	Acme,	MG
2	Best,	MG

Y (s≠,	p≠,	qtdade)
1,	1,	20
1,	2,	50

p (p≠,	nome,	tipo
1	LSI	micro
2	P11	mini
3	360	G.Porte
4	CRI	G.Porte
5	8080	micro

O específico superconjunto recuperado é determinado por condições de eficiência. As relações recuperadas são tam bém transmitidas a um nó simples, por exemplo ao nó 3.

Fig. 4.6 - PROCESSAMENTO DO ENVELOPE

Um dos elementos principais usados nesta estratégia é a redução do Banco de Dados baseado no operador de semijunção (Fig. 4.7).

- REDUÇÃO:

Programa P

1. S: = S(locação = "MG") ; Restrição de S a fornecedores MG

2. Y: = Y⟨S≠ = S≠⟩S; esta operação é uma semijunção - ela calcula o conjunto de tuplas Y que corresponde a fornecedores "MG".

- O OPERADOR DE SEMIJUNÇÃO (Exemplo)

Dado

S(S≠, nome, locação)		
1	Acme,	MG
2	Best,	MG

Y(S≠,	P≠,	qtdade)
1,	1,	20
1,	2,	50
3,	3,	50
4,	1,	10
4,	5,	75

P(P≠,	nome,	tipo)
1,	LSI	micro
2,	P11	mini
3,	360	G.Porte
4,	CRI	G.Porte
5,	8080	micro

. Y ⟨S≠ = S≠⟩ S =

Y(S≠,	P≠,	qtdade)
1	1	20
1	2	50

Tuplas Y que correspondem aos fornecedores de MG

. P ⟨P≠ = P≠⟩ Y =

P(P≠,	nome	tipo)
1,	LSI,	micro
2,	P11,	mini

Peças que são fornecidas por algum fornecedor de MG.

Fig. 4.7 - REDUÇÃO E SEMIJUNÇÃO NO SDD-1

V - PROCESSAMENTO CONCORRENTE DE TRANSAÇÕES NUM SGBDD

5.1 - INTRODUÇÃO

Sempre que é permitido a programas ou usuários múltiplos ter acesso a um banco de dados concorrentemente, o problema de concorrência aparece. O problema é sincronizar interações concorrentes de maneira que se possa ler e gravar dados consistentes permitindo que se complete a execução das transações. Num Banco de Dados Distribuído esse problema se agrava porque um mecanismo de controle de concorrência num nó não conhece instantaneamente as interações nos demais nós. O controle de concorrência é frequentemente citado como um dos mais difíceis problemas na área de gerenciamento de Banco de Dados Distribuídos[3,4,22,31].

O controle de concorrência é a parte de um sistema de gerenciamento de Banco de Dados Distribuído (SGBDD) que assegura, que transações executadas (i.e. consultas do usuário) em nós múltiplos produzam os mesmos resultados como se fossem executadas seqüencialmente num simples nó. Em outras palavras o controle de concorrência possibilita o compartilhamento de dados de forma completamente transparente aos usuários.

Muitos algoritmos de controle de concorrência foram propostos para SGBDD e muitos foram ou estão sendo implementados[3,22]. Estes algoritmos são usualmente complexos. Eles são descritos em diferentes terminologias e fazem diferentes suposições sobre o SGBDD, com isto, é difícil comparar os algoritmos propostos, mesmo em termos qualitativos. Neste trabalho

procura-se apresentar os algoritmos mais comumente encontrados em bibliografia recente, dividindo-os em duas categorias: Algoritmos baseados em ordem de escalonamento e Algoritmos baseados em técnicas de bloqueio ou exclusão mútua.

5.2 - CONCEITOS BÁSICOS

5.2.1 - INTEGRIDADE DOS DADOS

Numa implementação de banco de dados a integridade deve ser garantida [16]. Isto é, assegurar que as informações que entram no sistema, são verdadeiras e estão representadas corretamente. Um sistema bem desenvolvido deve permitir que se imponha restrições [16,22] que constituem regras que devem ser verificadas quanto aos dados; por exemplo,pode-se declarar que o valor do item renda familiar deve ser numérico; que a cada número de metrícula só pode corresponder um estudante, etc.

Se a mesma informação é guardada em diversos registros, possivelmente de arquivos diferentes, seu valor deve ser o mesmo a qualquer momento em todos esses registros. Chama-se a isso consistência dos dados. Pode-se ter uma situação de inconsistência quando vários usuários estão utilizando concorrentemente o banco de dados; enquanto um usuário está alterando alguns registros, um ou mais usuários estão tendo acesso aos mesmos registros sendo que alguns ainda não foram alterados.

O conjunto de restrições tem uma consistência própria

se não contém contradições. Esse conjunto, em geral, é implementado nos SGBD como parte da política de controle.

5.2.2 - OBJETOS

Um objeto é a menor unidade do Banco de Dados acessível pelo sistema de controle de concorrência (i.e. parte do sistema BDD que se ocupa em decidir que ações devem ser tomadas em resposta aos requerimentos dos programas de aplicação para LER ("READ) e GRAVAR ("WRITE") dentro do Banco de Dados). Num sistema particular de banco de dados os objetos podem ser arquivos, páginas, registros, etc.

Alguns autores chamam aos objetos de dados lógicos (3) e outros de grânulos (22). Neste capítulo só usaremos o nome OBJETO[20,29,31], para evitar confusões.

A cada um dos objetos se pode atribuir um VALOR. Como exemplos de valores se têm: inteiros, cadeias de caracteres, etc. Esses valores têm que respeitar as restrições de integridade. Um banco de dados que contém valores que satisfazem todos os requerimentos de integridade se diz que é um BANCO DE DADOS CONSISTENTE.

5.2.3 - AÇÃO

Uma ação é um comando de processamento primitivo e indivisível o qual é executado por um usuário simples.

Os objetos geralmente são modificados por um (número

de ações constituindo unidades funcionais que devem respeitar as restrições de integridade[22].

5.2.4 - OPERAÇÃO

Sequência de ações que executam sobre um objeto uma função que respeita a consistência interna[22].

Assim, por exemplo, se o objeto é uma página, então as operações básicas são LER-PAGINA ("read-pag") e GRAVAR-PÁGINA ("write page"), as quais constituem ações primitivas (indivisíveis ou atômicas) em muitos sistemas.

5.2.5 - TRANSAÇÃO

O termo transação é usado para descrever uma sequência de operações sobre um ou mais objetos do Banco de Dados (arquivos, registros, etc.) transformando o estado consistente atual do sistema num novo estado consistente [20,29]. Os usuários interagem com o SGBDD executando transações. As transações podem ser consultas em linha expressadas numa linguagem especial ou Programas de Aplicação escritos em linguagens de programação de propósito geral[3].

Assume-se que cada uma das transações i.e. uma transação sozinha é processada numa forma completa e correta, isto é, elas são executadas inicialmente sobre um Banco de Dados Consistente, terminam e produzem resultados corretos deixando o banco de dados CONSISTENTE. Na literatura este conceito é co

conhecido como o critério de correção [5].

5.2.6 - ESCALONADOR DE TRANSAÇÕES ("Logs", "Scheduler")

Controla o acesso de transações aos objetos do Banco de Dados [20,22]. Também o escalonador é definido como responsável pela ordem na qual as diferentes operações, para um conjunto de transações, são feitas.

O conceito de Escalonador também é conhecido com o nome de LOG [3,5] denotando uma sequência de leituras e gravações (das várias transações). As leituras e gravações das transações são denominadas "lógicas" pois não necessariamente implicam em acesso aos bancos de dados locais. As leituras e gravações executadas sobre os bancos locais são chamadas de "físicas".

5.2.7 - ANOMALIAS NO PROCESSAMENTO CONCORRENTE DE TRANSAÇÕES

A anomalia no processamento de transações concorrentes se produz como resultado da interferência entre usuários que estão simultaneamente tendo acesso ao banco de dados: Essa interferência causa problemas de inconsistência, caracterizada pela violação das restrições de integridade. Assim, a finalidade principal do controle de concorrência é a prevenção dessa interferência.

As anomalias típicas pertencem aos três seguintes tipos:

a) Anomalia 1: Perda de Operações

b) Anomalia 2: Recuperações Inconsistentes

c) Anomalia 3: Conflito de Operações

5.2.7.1 - ANOMALIA 1: PERDA DE OPERAÇÕES

Esse problema aparece na seguinte sequência de eventos: Durante a execução de uma transação T_1 uma operação a_i pode ler um dado X dentro de uma locação de memória intermediária ("buffer") X_1; então a transação modifica o objeto em X_1 através de uma operação subsequente a_j $(j > i)$, usando o valor previamente armazenado em X_1. Concorrentemente, outra transação T_2 atualiza o mesmo dado X através de operação a_k que ocorre entre a_i e a_j. Logicamente o efeito da operação a_k se perdeu.

Essa situação é representada na figura 5.1.

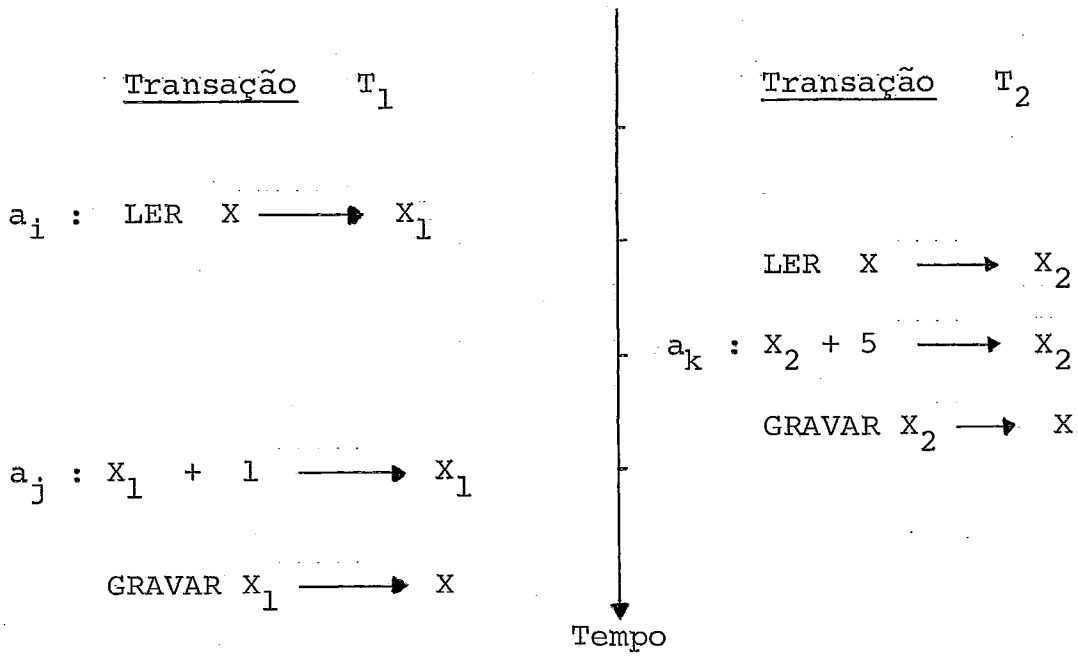


Fig. 5.1 - EXEMPLO DA ANOMALIA 1: PERDA DE OPERAÇÕES

5.2.7.2 - ANOMALIA 2 - RECUPERAÇÕES INCONSISTENTES

Esta anomalia é devida à presença de restrições de integridade locais e/ou globais. Esse problema aparece cada vez que uma transação tem acesso ou pior ainda, modifica um estado transitório do banco de dados caracterizado pelo fato que uma restrição de integridade não é verificada.

Exemplo: Assume-se que dois itens A e B devem satisfazer a constante de integridade A = B. A Transação T_2 imprime A depois que foi modificada por T_1, e B antes de ser modificada por T_1. A situação é descrita na Fig. 5.2.

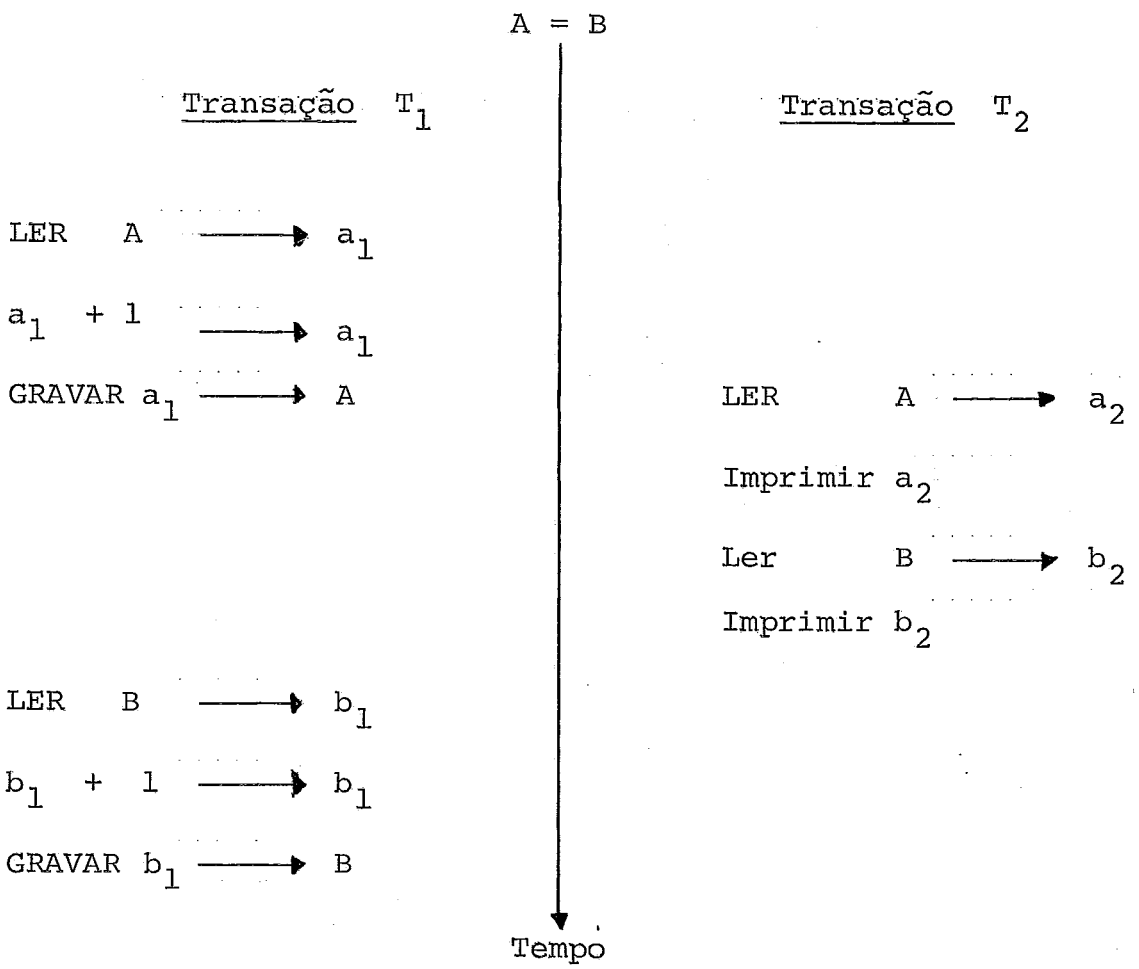


Fig. 5.2 - EXEMPLO DE RECUPERAÇÃO INCONSISTENTE

5.2.7.3 - ANOMALIA 3 - CONFLITOS DE OPERAÇÕES

Esta anomalia aparece como corolário ou sumário das duas anomalias anteriores e ocorre quando duas operações atuam sobre o mesmo dado e uma das operações,é uma gravação ("WRITE")

A ordem de execução das operações é computacionalmente significante se e somente se as operações estão em conflito.

Para ilustrar a noção de conflito considerar o dado X e as transações T_i e T_j. Se T_i executa a operação LER (X) e T_j executa GRAVAR (X), o valor lido por T_i será em geral diferente dependendo se a execução de LER (X) foi anterior ou posterior à GRAVAR (X). Similarmente,se ambas transações executam operações de gravação, o valor final de X depende da ordem de ocorrência da última operação de gravação. Esses conflitos são chamados [3] conflitos de leitura - gravação (lg) ou conflitos de gravação-gravação (gg) respectivamente.

Assim, duas transações estão em conflito quando geram um estado inconsistente no banco de dados se são executadas concorrentemente.

Um exemplo real de conflito que causa uma perda de atualização é ilustrado com o seguinte caso (Fig. 5.3): Dois clientes simultaneamente tratam de depositar dinheiro na mesma conta bancária. Na ausência de um mecanismo de controle de concorrência,esses dois clientes podem ler o saldo ao mesmo tempo, calcular o novo saldo em paralelo após o depósito e então gravá-lo no banco de dados. O resultado final é incorreto porque só refletirá uma atividade devido ao conflito de gravação-gravação como consequência da ordem de ocorrência da operação de gravação do saldo dos dois clientes.

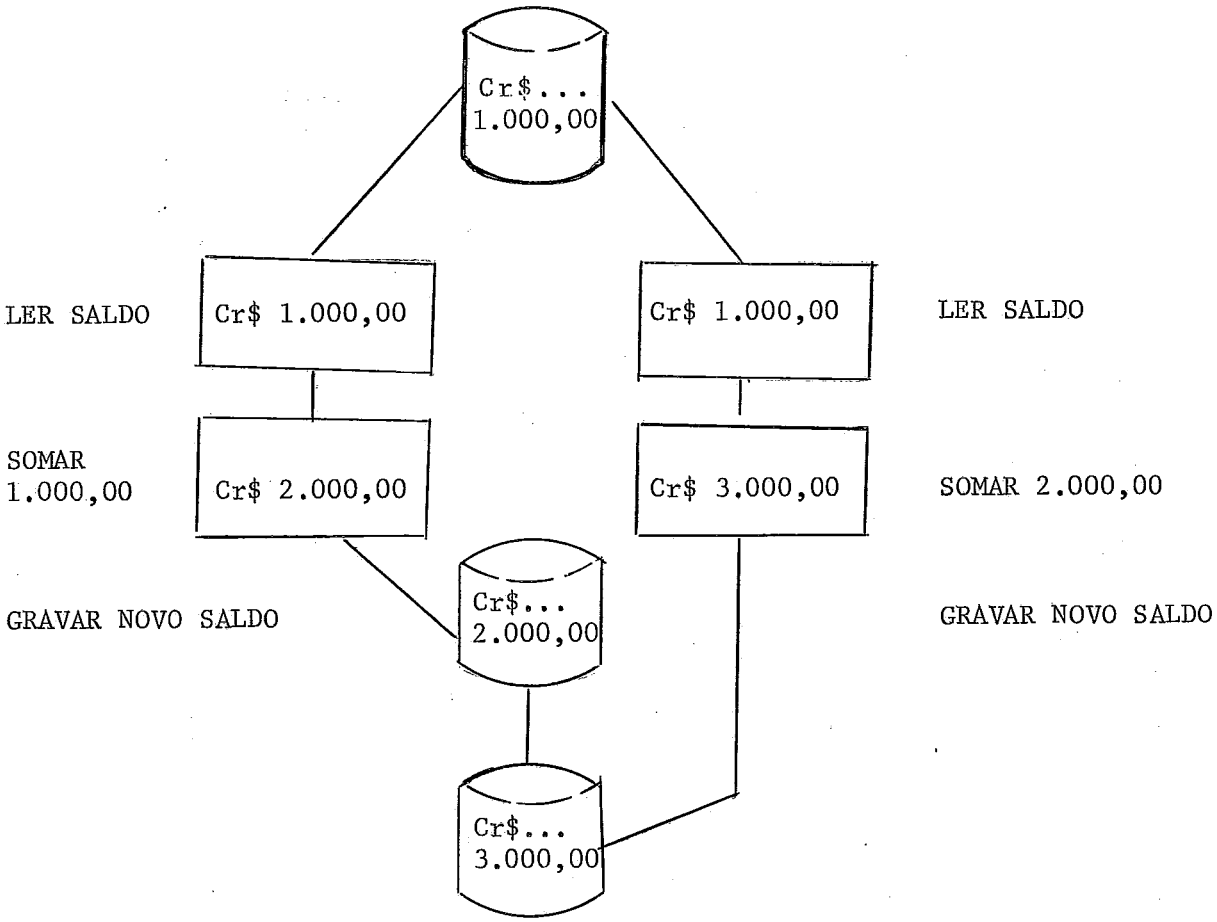


FIG. 5.3 - PERDA DE ATUALIZAÇÃO CAUSADA POR CONFLITO DE GRAVAÇÃO-GRAVAÇÃO (gg)

5.2.8 - ESCALONAMENTO EM SÉRIE DE TRANSAÇÕES

Conforme foi explicado no item anterior (5.2.7) alguns tipos de escalonamentos ou "logs" introduzem anomalias. O problema do controle de concorrência consiste em permitir a execução de somente aqueles escalonamentos que não gerem anomalias. Sabe-se que [22] uma execução sucessiva de transações, i.e., cada uma das transações é executada completamente antes do início da seguinte transação, corresponde a um tipo de escalonamento ou "log" especial que não apresenta anomalias e que exclui as transações simultâneas ou concorrentes. Esse tipo de

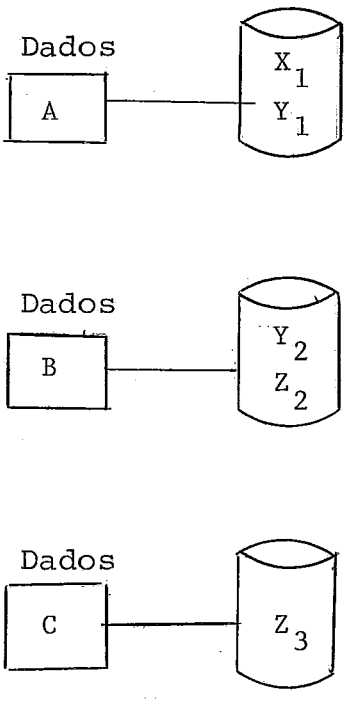
Transações

T_1: Ler (x)
Gravar (y)

T_2: LER (y);
Gravar (z);

T_3: LER (z);
Gravar (x);

Banco de Dados Distribuídos



Uma execução possível de T_1, T_2, e T_3 é representada pelos seguintes escalonamentos (Notação: $r_i[x]$ denota a operação Ler (x) resultante da operação T_i; $w_i[x]$ denota a operação Gravar (x) resultante da operação T_i):

Escalonamento para A: $r_1[x_1] w_1[y_1] r_2[y_1] w_3[x_1] \tilde{=} T_1\ T_2\ T_3$

Escalonamento para B: $w_1[y_2]\ w_2[z_2] \tilde{=} T_1\ T_2$

Escalonamento para C: $w_2[z_3]\ r_3[z_3] \tilde{=} T_2\ T_3$

Fig. 5.4 - ESCALONAMENTO SERIADO

escalonamento é chamado de Escalonamento em Série. Tal escalonamento representa uma execução na qual nenhuma transação se executa concorrentemente. Dado que se assume que cada uma das transações preserva consistência, se é executada sozinha uma sequência serial de transações também preserva consistência.

Um escalonamento é "Serializável", i.e. pode ser executado em série, se é computacionalmente equivalente a uma execução seriada, significando que para todo estado inicial do banco de dados se produz a mesma saída e o mesmo estado final como um escalonamento em série [3]. Uma forma prática de visualizar se um escalonamento é "serializável" é quando as operações de Leitura e Gravação para cada uma das transações são contíguas (Fig. 5.4). Este tipo de escalonamento representa uma execução na qual nenhuma transação executa-se concorrentemente. Na Fig. 5.4, por exemplo, cada escalonador é serial, isto é, não se tem concorrência de operações entre diferentes transações. No escalonamento local A, T_1 precede T_2, e T_2 precede T_3, No B, T_1 precede T_2 e no C, T_2 precede T_3. Assim T_1, T_2, T_3 estão ordenados satisfazendo a definição de serial. O seguinte escalonamento, por exemplo não é serial:

Escalonamento para A: $r_1[x_1]\ r_2[Y_1]\ W_3[x_1]\ W_1[Y_1] \tilde{=} T_1T_2T_3T_1$

Escalonamento para B: $W_2[z_2]\ W_1[Y_2] \tilde{=} T_2\ T_1$

Escalonamento para C: $W_2[z_3]\ r_3[z_3] \tilde{=} T_2\ T_3$

No escalonamento local para A temos que T_1 não executa todas suas operações em forma contigua, i.e.,as operações de leitura $[r_1][x_1]]$ e gravação $[w_1][y_1]$ não aparecem juntas.

Baseado nos exemplos apresentados, pode-se afirmar que um escalonador é serializável se e somente se para cada escalonador local, todas as ações da transação aparecem juntas,e se a seqüência das transações é a mesma em todos os nós [3]. A "Serialização" é o critério de correção adaptado pela maioria dos pesquisadores [3,22,31] na área de controle de concorrência dos SGBD's

O objetivo principal do sistema de controle de concorrência é assegurar que todas as execuções possam ser executadas em série [3] para transformar um escalonamento qualquer num escalonamento seriado é preciso conhecer as características dos escalonamentos em série.

5.2.8.1 - CARACTERÍSTICAS DO ESCALONAMENTO EM SÉRIE

Com a finalidade de apresentar as características dos escalonamentos em série, duas transformações básicas do escalonamento de transações deverão ser introduzidas. A primeira é a separação de duas operações O_i e O_j, as quais são executadas por diferentes transações e consiste em substituir o escalonamento dessas operações $E(O_i, O_j)$ por uma sequência que possa produzir o mesmo resultado O_i, O_j ou O_j, O_i (As operações O_i, O_j têm que ser compatíveis, i.e. duas operações O_i,O_j são compatíveis se, para alguma execução simultânea de O_i e O_j, o resultado da execução é o mesmo como se ele fosse produzido

pela execução de O_i seguido por O_j e vice-versa) [4].

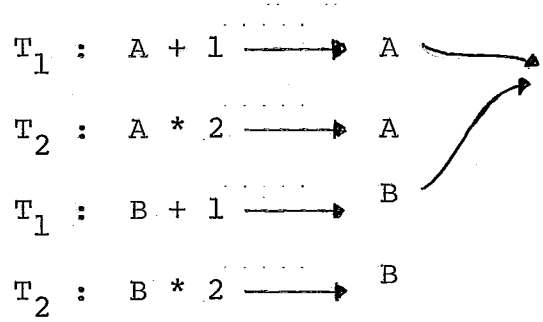
Assim, a separação de operações permite formar uma sequência de operações compatíveis executadas por diferentes transações.

A seguinte transformação é a permutação das operações O_i e O_j, as quais são executadas por diferentes transações, e consiste em intercambiar a ordem da execução dessas operações, (As operações O_i e O_j tem que ser permutáveis, i.e., duas operações O_i e O_j são permutáveis se toda execução de O_i seguida por O_j produz o mesmo resultado como se fosse a execução de O_j seguida por O_i).

Assim, segundo essa transformação, a sequência O_i, O_j é substituída pela sequência O_j, O_i.

Logo, a primeira característica é que uma condição suficiente para que um escalonamento seja em série é que ele possa ser transformado através de uma separação de operações compatíveis e uma permutação de operações permutáveis num escalonamento seriado.

Por exemplo, considerar o escalonamento seguinte:



Não é um escalonamento em série de T_1 e T_2, i.e. as operações feitas por T_1 e T_2 não são contíguas.

As operações $A * 2 \longrightarrow A$; e $B + 1 \longrightarrow B$ são permutáveis porque operam diferentes objetos: A,B. Isto significa, que duas operações que não modificam qualquer objeto e que pertencem a duas diferentes transações [22] podem ser sempre executadas simulta

neamente sem afetar o resultado de sua execução.

Assim, o escalonamento pode ser tranformado em:

T_1: A + 1 ——— A

T_1: B + 1 ——— B

T_2: A * 2 ——— A

T_2: B * 2 ——— B

que é um escalonamento seriado de T1 e então T_2.

A segunda característica do escalonamento em série refere-se à equivalência de execuções produzida por esse tipo de escalonamento e sua finalidade é não permitir que se produza a anomalia 3 (Conflito de Operações). Duas execuções são computacionalmente equivalentes se [3]:

1- Cada uma das operações de leitura (Read) lê valores que foram produzidos pela mesma operação de gravação (Write) em ambas as execuções.
Nesta regra se assegura que cada transação tenha a mesma entrada em ambas as execuções,

2- O comando de gravação final sobre cada dado é o mesmo em ambas as execuções.
Esta regra, combinada com a anterior assegura, que ambas as execuções deixem o Banco de dados num estado final consistente. Além disso, presume-se que há uma transação inicial que "cria" o estado inicial do Banco e que há uma transação final que "lê" o estado final do Banco.

5.2.8.2 - GRAFOS DE PRECEDÊNCIA

As operações LER ("READ") e GRAVAR ("WRITE") são operações básicas, a partir das quais se constróem as outras Além disso, elas podem ser causadoras da anomalia 3 ("conflitos"). Com a finalidade de evitar conflitos o sistema deve assegurar a separação dessas operações, quando elas atuam sobre o mesmo objeto.

Na prática, a separação seguida pela permutação de operações produzidas sobre objetos diferentes é sempre possível. Na verdade, somente certas permutações produzidas sobre os mesmos objetos são impossíveis [22]. Em forma mais precisa, se X é um objeto, pode-se ter os seguintes relacionamentos entre transações:

1- (T_i: LER x) e (T_j: LER x) são operações permutáveis

2- (T_i: LER x) e (T_j: GRAVAR x) são operações não permutáveis

Assim, quando T_i lê um objeto antes do que T_j o escreva, diz-se que T_i precede T_j; esta relação é denotada $T_i \xrightarrow{lg} T_j$ (3).

3- (T_j: GRAVAR x) e (T_j: GRAVAR x) são operações não-permutáveis

Assim, quando T_i grava um objeto antes do que T_j diz-se que T_i precede T_j; esta relação é denotada $T_i \xrightarrow{gg} T_j$ (3).

Conseqüentemente, de acordo com a notação, as operações seguintes não são permutáveis.

4- $T_i \xrightarrow{gl} T_j$

5- $T_i \longrightarrow_{lgl} T_j$ se $T_i \longrightarrow T_j$ ou

$T_i \longrightarrow_{gl} T_j$

Em conclusão:

$T_i \longrightarrow T_j$ se $T_i \longrightarrow_{lgl} T_j$ ou $T_i \longrightarrow_{gg} T_j$

Esses relacionamentos permitem definir um grafo de preferência [31]. Um grafo de precedência de um escalonamento é um grafo cujo conjunto de vértices é o conjunto de transações e tal que exista um arco desde T_i a T_j se T_i precede a T_j ($T_i \longrightarrow T_j$). As pesquisas [22,16] demonstram a seguinte utilidade dos grafos de precedência:

Uma condição suficiente para que um escalonamento seja "serializável" é que seu grafo de precedência associado não tenha circuitos, i.e., o grafo deve ser acíclico [3,5,22].

5.2.8.3 - A ORDEM DO ESCALONAMENTO EM SÉRIE

Para obter um escalonamento em série, o SGBD deve garantir que as operações conflitantes: LER e GRAVAR, sejam processadas numa certa ordem e de acordo com o seguinte teorema [3,22]:

"Seja $\{T = T_1, \ldots, T_m\}$ um conjunto de transações e seja "E" a execução dessas transações produzidas pelos escalonamentos $\{L_1, L_2, \ldots L_m\}$. "E" pode ser processada em série se existe uma ordenação total de T tal que, para cada par de operações em conflito O_i e O_j de diferentes transações T_i e T_j

(respectivamente), O_i precede à O_j em qualquer escalonamento L_1, ..., L_m , se e somente se T_i precede T_j na ordem total",i. e.,se as transações têm sido executadas em série na ordem do processamento seriado, a "computação" executada pelas transações teria que ser idêntica à "computação" representada por E. A ordem total formulada no TEOREMA é chamada a ordem do escalonamento em série.

Assim, Berstein [2,3,4] diz que o controle de concorrência é a atividade de controlar a ordem relativa das operações em conflito e que algoritmos para fazer esses tipos de controle são chamados de TECNICAS DE SINCRONIZAÇÃO. Berstein sugere que os conflitos gg ou lg devem ser sincronizados independentemente tanto quanto se deve ter uma ordem total das transações consistente com ambos tipos de conflito. Como exemplo, nas figuras 5.6.a e 5.6.b se mostra um escalonamento para duas transações T_1 e T_2 (Fig. 4.5) que devem ser processadas em série.Na Fig.4.7 mostra-se o grafo de precedência para os dois escalonamentos e se demonstra que o escalonamento da Fig. 5.7 não é "Serializável" por apresentar um grafo de precedência cíclico. Além disto, o escalonamento da Fig. 5.7 apresenta a anomalia 1: perdas de operações, porque das duas últimas operações: T_1 : Gravar b_1 ⟶ B

T_2 : Gravar b_2 ⟶ B

só tem efeito a operação de T_2 devido a operação T_1 ter se perdido.

Transação T_1		Transação T_2	
Ler A	$\longrightarrow a_1$	Ler A	$\longrightarrow a_2$
$a_1 + 1$	$\longrightarrow a_1$	$a_2 * 2$	$\longrightarrow a_2$
Gravar a_1	$\longrightarrow$ A	Gravar a_2	$\longrightarrow$ A
Ler B	$\longrightarrow b_1$	Ler B	$\longrightarrow b_2$
$b_1 + 1$	$\longrightarrow b_1$	$b_2 * 2$	$\longrightarrow b_2$
Gravar b_1	$\longrightarrow$ B	Gravar b_2	$\longrightarrow$ B

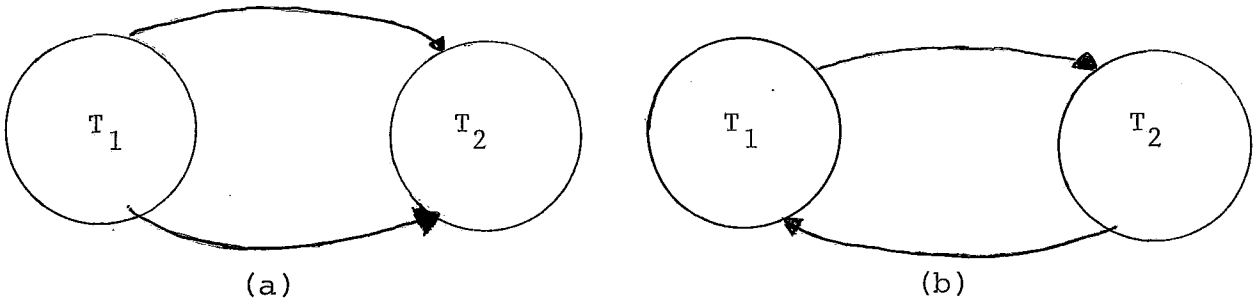
Fig. 5.5 - TRANSAÇÕES T_1 e T_2

Um escal. para T_1 e T_2		Outro escal. para T_1 e T_2	
T_1: Ler A a_1	$\longrightarrow a_1$	T_2: Ler A	$\longrightarrow a_2$
T_1: Ler $a_1 + 1$	$\longrightarrow a_1$	T_2: $a_2 * 2$	$\longrightarrow a_2$
T_1: Gravar a_1	$\longrightarrow$ A	T_1: Ler A	$\longrightarrow a_1$
T_2: Ler A	$\longrightarrow a_2$	T_1: $a_1 + 1$	$\longrightarrow a_1$
T_2: $a_2 * 2$	$\longrightarrow a_2$	T_2: Gravar a_2	$\longrightarrow$ A
T_2: Gravar a_2	$\longrightarrow$ A	T_2: Ler B	$\longrightarrow b_2$
T_1: Ler B	$\longrightarrow b_1$	T_2: $b_2 * 2$	$\longrightarrow b_2$
T_1: $b_1 + 1$	$\longrightarrow b_1$	T_1: Gravar a_1	$\longrightarrow$ A
T_1: Gravar b_1	$\longrightarrow$ B	T_1: Ler B	$\longrightarrow b_1$
T_2: Ler B	$\longrightarrow b_2$	T_1: $b_1 + 1$	$\longrightarrow b_1$
T_2: $b_2 * 2$	$\longrightarrow b_2$	T_1: Gravar b_1	$\longrightarrow$ B
T_2: Gravar b_2	$\longrightarrow$ B	T_2: Gravar b_2	$\longrightarrow$ B

(a) (b)

Tempo

Fig. 5.6 - DOIS ESCALONAMENTOS PARA AS TRANSAÇÕES T_1 e T_2 DA FIG. 4.5



(a) (b)

O grafo de precedência é cíclico, então o escalonamento não é "serializável".

Fig. 5.7 - GRAFOS DE PRECEDÊNCIA CORRESPONDENTE À FIG. 5.6

5.2.9 - FALHAS NO PROCESSAMENTO CONCORRENTE DE TRANSAÇÕES

Se acontecer uma falha durante o processamento de transações, um problema sério no controle de concorrência é criado: cada operação de gravação executada por cada uma das transações deve ficar sem efeito, i.e. o banco de dados deve ser restaurado até o estado anterior às gravações, para evitar que resultados parciais causem problemas de integridade no Banco de Dados.

As falhas principais que afetam o sistema de gerenciamento do Banco de Dados pertencem aos tipos seguintes [21,32].

. Falhas do processador e do software num nó (ou vários) do sistema ("crashes") interrompendo-se a operação normal dos processos.

. Falhas de comunicação: Ocorrem quando as mensagens entre os nós não são transmitidas apropriadamente.

. Falhas da memória intermediária ou principal, as

quais impossibilitam a recuperação "on.line" dos dados.

. Falhas múltiplas: quando mais de uma das anteriores falhas ocorrem ao mesmo tempo.

Proteção contra todos os tipos possíveis de falhas é em muitos casos impraticável [31], de modo que o primeiro passo num projeto de recuperação de transações é limitar os tipos de falhas que manipularão o algoritmo.

Um SGBDD para evitar que a ocorrência desses tipos de falhas causem problemas de inconsistência no banco de dados, deve possuir a propriedade da atualização atômica, que requer que todas ou nenhuma das operações de gravação sejam processadas. A implementação Standard dessa propriedade é o protocolo de atualização em duas etapas para Banco de Dados Distribuído [3] ("Two Phase Commit protocol"): Suponha-se que a transação T está atualizando os dados X e Y. Antes de T processar o último comando da transação (por exemplo END), a primeira fase da atualização começa, durante a qual o gerenciador de dados (GD) processa comandos de pré-gravação para X e Y. Esses comandos armazenam os valores de X e Y em arquivos temporários. Se acontecer uma falha durante a primeira fase, nenhum problema existe, já que nenhuma das atualizações tem sido sobre os arquivos permanentes do Banco de Dados. Durante a 2a. fase, o gerenciador de transações (GT) armazena ou copia os valores de X e Y nos arquivos permanentes do sistema. Se acontecer uma falha durante esta fase, o banco de dados pode achar informação incorreta, mas visto que os valores de X e Y estão armazenados em arquivos temporários, esta inconsistência pode

ser corrigida mediante um procedimento de recuperação: têm-se os valores de X e Y dos arquivos temporários e continuam-se as atividades normais do sistema.

Esta técnica atinge 100% de proteção [5] contra as falhas no processamento das transações, mas não é completamente confiável com respeito às falhas em múltiplos nós. Hammer e Shipmon [24] descrevem mecanismos para otimizar a confiabilidade de múltiplos nós do protocolo de atualização em duas fases.

5.2.9.1 - IMPLEMENTAÇÃO DO PROTOCOLO DE ATUALIZAÇÃO EM DUAS ETAPAS

Uma das formas de implementação do protocolo de atualização em duas etapas é mediante o uso de um nó como COORDENADOR CENTRAL comunicando-se com os outros chamados PARTICIPANTES do processo. Este método é conhecido como PROTOCOLO CENTRALIZADO DE ATUALIZAÇÃO EM DUAS ETAPAS [16,29,32] e sumariamente é da forma seguinte:

ETAPA-1

(1) O nó coordenador envia uma mensagem a todos os nós para conferir se "estão prontos" para iniciar o processo de atualização.

(2) Esperar pela resposta de cada um dos nós participantes:

a) Se qualquer nó, ante uma falha, envia uma resposta negativa (ex. "NÃO PRONTO") ou o tempo de espera pela resposta se esgotou, o coordena

dor considera que a FASE 1 não foi um sucesso

b) Se todos os nós enviam uma resposta positiva, o coordenador considera que a FASE 1 foi bem sucedida.

ETAPA 2

Se a etapa 1 foi bem sucedida, o coordenador central envia uma mensagem a todos os nós participantes para fechar com êxito, ("COMMIT") o processamento local da transação. Caso contrário, o nó coordenador envia uma mensagem a todos os nós informando a conclusão sem êxito do processamento da transação ("ROLLBACK"). Assim, todos os nós participantes têm que deixar sem efeito as atualizações parciais feitas, para manter a integridade do Banco de Dados.

É importante notar que o protocolo de atualização em duas etapas é requerido, em princípio, sempre que uma transação tenha interação com recursos múltiplos e independentes. Assim, é particularmente importante em SGBDD [16] onde recursos diferentes residem em diferentes nós ligados através de uma rede de comunicação e nós individuais ou linhas podem falhar em qualquer tempo.

Assim, o protocolo de atualização em duas etapas permite que todos os nós participantes possam fechar a transação com êxito se não acontecer falha ou rejeitar o processo em caso contrário.

Logo, podemos reconhecer as seguintes operações principais num sistema de Banco de Dados Distribuídos [49].

(1) READ-OBJ: Examina mas não altera o valor de um objeto e se representa:

⟨T, READ-OBJ, E, N , Val⟩

i.e. a transação T efetua uma operação de leitura do objeto E no nó N, sendo val o valor do E.

(2) WRITE-OBJ: ⟨T, WRITE-OBJ, E, N , Val⟩

i.e. a transação T efetua a operação WRITE-OBJ sobre o objeto E armazenado no nó N alterando-se o valor: Val, do objeto. O novo valor de E não está armazenado no Banco de Dados permanente neste momento.

(3) COMMIT-OBJ: Representa o fechamento de operações com êxito, permitindo-se a atualização dos dados em forma permanente para evitar afetar a integridade do Banco de Dados. É representada por:

⟨T, COMMIT-OBJ, -, N , -⟩

e a implementação desta operação é representada pelo PROTOCOLO DE ATUALIZAÇÃO EM DUAS ETAPAS("Two Phase Commit").

5.3 - ALGORITMOS BASEADOS NA ORDEM DO ESCALONAMENTO

5.3.1 - DEFINIÇÕES BÁSICAS

Os algoritmos baseados na ordem do escalonamento se caracterizam por usar uma técnica de sincronização onde a or

dem de execução das transações é selecionada a priori com o objetivo de que o acesso aos objetos de dados seja na ordem atribuída. Para atingir esse objetivo é necessário definir o "timestamp" das transações e dos objetos.

5.3.1.1 - TIMESTAMP DE TRANSAÇÕES

É um número único atribuído a uma transação e é escolhido de uma sequência monotonicamente crescente que frequentemente é uma função do Tempo no dia [2,9]. O "timestamp" da operação LER é denotado ts-LER.

Existem muitos métodos de geração de "timestamps" [22], que devem ser únicos para todo o sistema. Assim, os "timestamps" consistem geralmente da concatenação de dois componentes: um valor do relógio local (componente maior) e um identificador do nó (componente menor). O valor do relógio local é o tempo do relógio local lido no nó onde a transação ocorreu.O identificador do nó deve ser único para todo o sistema.

5.3.1.2 - TIMESTAMPS DE OBJETOS

É uma variável numérica associada com um objeto de dados que armazena o "timestamp" da última transação que operou sobre este objeto [22].

Para fazer uma melhor diferença entre as operações que estão sendo executadas é possível distinguir entre um "timestamp" de leitura e um "timestamp" de gravação associado a

um objeto ou dado x.

Timestamp de leitura $\left[T_s - \text{ler}(x)\right]$ É o valor associado ao objeto x do timestamp da última leitura física feita do objeto x.

Timestamp de Gravação $\left[T_s - \text{gravar}(x)\right]$ É o valor associado ao objeto x do timestamp da última gravação física feita do objeto x.

5.3.2 - SINCRONIZAÇÃO DE LEITURA/GRAVAÇÃO E GRAVAÇÃO/GRAVAÇÃO BASEADO NO "TIMESTAMP" DAS TRANSAÇÕES E OBJETOS

Se não se toma em conta o protocolo de atualização em duas fases, o algoritmo é muito simples [3].

Em cada módulo gerenciador de dados (GD), o escalonador de transações registra o maior "timestamp" de alguma operação de leitura ou gravação que tenha sido processada. Estes "Timestamps" são denotados como Ts - LER (x) e Ts - GRAVAR (x). O algoritmo de leitura/gravação e gravação/gravação se mostra na Fig. 5.8. Quando uma Transação executa uma LEITURA, o escalonador controla a seqüência correta da LEITURA com respeito à última gravação, i.e. o "timestamp" da transação de leitura deve ser maior do que o valor do timestamp de gravação associado ao objeto. Por outro lado, quando uma transação executa uma gravação, o escalonamento deve controlar a seqüência correta dessa operação com respeito à LEITURA previamente executada, i.

e. o valor do timestamp de gravação deve ser maior do que timestamp da leitura associado ao objeto. No caso de uma sincronização gravação/gravação, o escalonador rejeita a transação de gravação se seu timestamp é menor do que ts-gravar (x), caso contrário atualiza o timestamp e processa a transação.

No caso que não se cumpram as premissas anteriores, a transação é abortada. As transações abortadas recebem um timestamp maior e são re-executadas posteriormente.

Seja:
- ts-ler — Timestamp da transação de leitura.
- ts-ler (x) — O maior timestamp de alguma operação de leitura do objeto x que tenha sido processada.
- ts-gravar — Timestamp da transação de gravação.
- ts-gravar(x) — O maior timestamp de alguma operação de gravação do objeto x que tenha sido processada.

```
Case Operação-Leitura ── Se ts-gravar(x) < ts-ler
                         então
                         -  /* Processar a leitura
                         -  /* Atualizar o timestamp
                         -  /* ts-ler(x):=MAX(ts-ler,
                                              ts-ler(x))
                         -
                         Caso contrário
                               Rejeite a transação de leitura
                               "Transação abortada"

     Operação-gravação ─ Se ts-ler(x) < ts-gravar
                            V ts-gravar <ts-gravar(x)
                         então
                            Processar a gravação
                            /* Atualizar o timestamp
                            ts-gravar(x):=MAX(ts-gravar,
                                              ts-gravar(x))
                         Caso contrário
                            Rejeite a transação de gravação
                            "TRANSAÇÃO Abortada"
```

Fig. 5.8 - ALGORITMO DE SINCRONIZAÇÃO DE LEITURA/GRAVAÇÃO E GRAVAÇÃO/GRAVAÇÃO (IGNORANDO O PROTOCOLO DE ATUALIZAÇÃO EM DUAS FASES) USANDO O TIMESTAMP DE TRANSAÇÕES E OBJETOS.

Se uma transação envia comandos de gravação a dois ou mais nós, e se alguns deles e não todos são rejeitados, então a execução não é serial. Assim, ignorando o protocolo de atualização em duas fases, não se garante que o escalonamento das transações seja em série [3,5]. Por isso, deve-se modificar o algoritmo anterior para atingir o critério de escalonamento em série.

O protocolo de atualização em duas fases é incorporado mediante a pré-gravação dos "timestamps" e aceitando ou rejeitando pregravações em vez de transações de gravação. Uma vez que o escalonador aceita uma pre-gravação, deve-se garantir a aceitação da gravação correspondente sem importar quando. Para uma sincronização de leitura/gravação (ou gravação/gravação), uma vez que o escalonador aceita uma pré-gravação (x) com "timestamp" ts-pregravar, não se deve processar qualquer operação de leitura (ou operação de gravação) com timestamp maior do que ts-pregravar até que a operação de gravação tenha sido processada. Isto equivale a estabelecer um lock de gravação sobre x enquanto dure o protocolo de atualização em duas fases.

Os algoritmos de leitura/gravação e gravação/gravação mediante "timestamps" é usando o protocolo de atualização em duas fases; são apresentados nas Figs. 5.9 e 5.10 respectivamente.

Fig. 5.9 - SINCRONIZAÇÃO DE LEITURA-GRAVAÇÃO USANDO TIMESTAMPS E O PROTOCOLO DE ATUALIZAÇÃO EM DUAS FASES

Seja: min-ts-ler (x) — O mínimo timestamp de alguma transação de leitura do objeto x.

min-ts-gravar (x) — O mínimo timestamp de alguma transação de gravação do objeto x.

min-ts-pregravar (x) — O mínimo timestamp de alguma transação de pre-gravação do objeto x

```
Passo 1: /* Processo da transação de leitura */
    Se timestamp-ler < timestamp-gravar (x)
    então
        "Rejeitar a transação de leitura"
    caso contrário
        Se timestamp-ler < min-ts-pregravar (x)
        então
            "A transação de leitura está pronta ("READY")
             para ser executada"
        caso contrário
            "A transação de leitura é armazenada na memó
             ria intermediária"("buffer")
```

Passo 2: /* Processo de pregravação /*

Se timestamp-pregravação < timestamp-ler (x)
então
    "A pregravação é rejeitada"
caso contrário
    "A pregravação é armazenada na memória intermediária"

Passo 3: /* Processo da transação de gravação */

"A transação de gravação nunca é rejeitada"

Se timestamp-gravar > min-ts-ler (x)
então
    "Armazenar a transação de gravação na memória intermediária"
caso contrário
    "A transação de gravação está pronta para ser executada"

OBSERVAÇÃO: Quando a transação de gravação é executada, sua correspondente pre-gravação é eliminada da memória intermediária, isto causa um incremento do min-ts-pregravar (x) e por conseguinte os três primeiros passos serão verificados novamente até que nenhuma variação aconteça.

Fig. 5.10 - SINCRONIZAÇÃO DE GRAVAÇÃO-GRAVAÇÃO USANDO TIMESTAMPS E O PROTOCOLO DE ATUALIZAÇÃO EM DUAS FASES

Passo 1: /* Processo de pregravação */

Se timestamp-pre-gravação (x) < timestamp-gravar (x)
então
    "A pregravação é rejeitada"
caso contrário
    "A pregravação é armazenada na memória intermediária"

Passo 2: /* Processo de gravação */

/* A transação de gravação nunca é rejeitada */

Se timestamp-gravar (x) > min-ts-pre-gravar (x)
então
    "A transação de gravação é armazenada na memória intermediária"
caso contrário
    "A gravação é executada"

OBSERVAÇÃO: Quando a gravação é executada sua correspondente operação de pre-gravação é retirada. Portanto min-ts pregravar (x) é incrementado. Com este incremento, as gravações armazenadas são novamente testadas para verificar sua possível execução.

5.3.3 - SINCRONIZAÇÃO DE GRAVAÇÕES COM GRAVAÇÕES (THOMAS WRITE RULE - TWR)

Seja X um comando de gravação sobre x e suponha-se que ts-W < ts-W (x). Em vez de rejeitar W podemos simplesmente ignorá-la. Chama-se a esse procedimento a Regra Thomas de gravação.

Vê-se no esquema seguinte; a descrição do algoritmo:

Se ts-gravar (x) < ts-gravar
então
    Processe a gravação
    Atualize o ts-gravar (x)
caso contrário
    Ignore a gravação ("não rejeitar")

Intuitivamente, a Regra Thomas se aplica a um comando de gravação que trata de colocar informação obsoleta dentro do Banco de Dados. A Regra garante [3] que o efeito de aplicar um conjunto de comandos de gravação a x é idêntico à aplicação de comandos de gravação em ordem serial.

5.3.4 - ALGORITMO BASEADO EM VERSÕES MÚLTIPLAS

As estratégias descritas podem ser otimizadas mediante a preservação de várias versões do objeto de dados. Para cada objeto x, o sistema preserva [22] :

- O conjunto de timestamps de gravação {ts-gravar (x)} com o valor associado do objeto {V_i(x)}

- O conjunto de timestamps de leitura associado ao objeto x $\{$ ts-ler- (x) $\}$

Assim, os conjuntos de pares $\{$ts-gravar (x), V_i (x)$\}$ são chamados versões [3]. O objetivo deste algoritmo é assegurar a ordem das operações de LEITURA com respeito às operações de GRAVAÇÃO sem precisar do processo de "REINÍCIO" ("RESTART") das operações de LEITURA. Para fazer isso é necessário forçar à transação T_i a LER um objeto x com a versão cujo timestamp de gravação associado a x seja imediatamente menor que o anunciado a i. Dessa forma T_i sempre precederá todas as transações de timestamp de gravação maior e prosseguirá a todas as transações com "timestamps" menores.

A seguir é apresentado um exemplo descritivo do algoritmo:

a) Representar-se-ão versões associadas ao dado x sobre uma "linha de tempo"

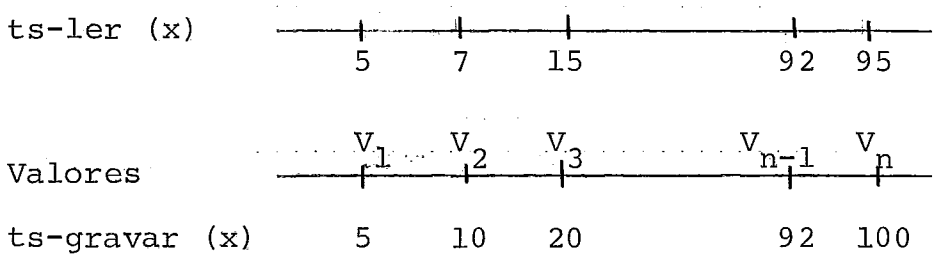
Valores	V_1	V_2	V_3	V_{n-1}	V_n
ts-gravar (x)	5	10	20	92	100

Seja L uma operação de leitura ---➤ LER (x)

L é processado lendo a versão de x com o timestamp de gravação imediamente menor do que o timestamp de L e incluindo esse último timestamp dentro do conjunto ts-leitura (x). Por exemplo, para processar uma operação de leitura com timestamp 95, acha-se o timestamp de gravação (ts-gravar (x)), imediatamente menor de 95, nesse caso 92. Assim, o valor

lido pela operação de leitura é (92) V_{n-1}

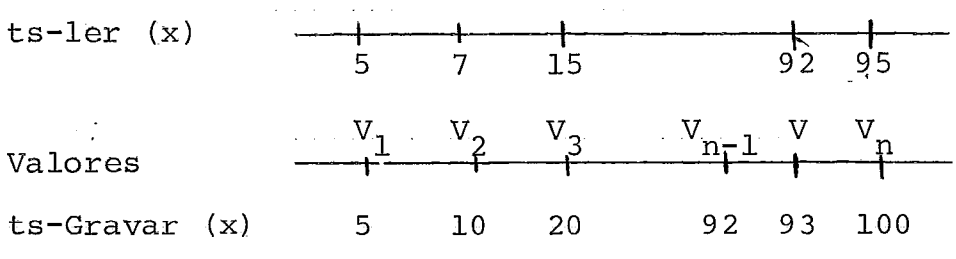
b) Similarmente, representa-se os timestamps de leitura associado ao objeto x:



Seja G uma operação de gravação do objeto x e seja I (G) o intervalo entre ts-G (timestamp da operação G) ao menor valor entre ts-G (x) > ts-G. Se algum ts-ler (x) está no intervalo I(G) a operação gravação é rejeitada, em caso contrário processa-se G criando-se uma nova versão de x com timestamp ts- G.

c) Continuando com o exemplo: Seja G uma operação de gravação com timestamp 93. O intervalo (G) = (93,100).

Para processar G cria-se uma versão nova de x com esse timestamp, i.e. 93



Contudo, nesta versão se "invalida" o comando de

leitura da parte (a) porque se o comando de leitura chegou primeiro do que o comando de gravação, ele deve ter lido V = 93 em vez de V_{n-1} = 92. Assim, deve-se rejeitar o comando de gravação porque o timestamp da operação de leitura = 95 está dentro do intervalo (93,100).

Ignorando o protocolo de atualização em duas fases não se garante que o escalonamento das transações seja em série [3,5]. Para integrar o protocolo de atualização de duas fases se requer que as transações de leitura e pre-gravação sejam armazenadas na memória intermediária como no caso de sincronização de leitura/gravação ou gravação/gravação apresentado no item 3.2, com a diferença de que as transações de leitura nunca são rejeitadas e as de gravação podem ser rejeitadas [3].

5.3.5 - ALGORITMOS CONSERVATIVOS

Os algoritmos conservativos pertencem a uma classe especial de técnica para eliminar os reinícios ("restarts") durante o escalonamento dos "timestamps".

Nesses algoritmos se requer que cada escalonador receba as operações de leitura e gravação na ordem estrita dos "timestamp" [4,22]. Por exemplo, se o escalonador E_j recebe LER (x) seguido por LER (j) do escalonador E_i, então ts-LER (x) $\leq$ ts-LER (j).

Para atingir seu objetivo de eliminar reinícios ("restarts") durante o escalonamento dos timestamps, o escalo

nador, quando recebe uma operação X que poderia causar um futuro reinício, demora a operação X até que não sejam possíveis mais futuros reinícios.

Como exemplo, apresentaremos, segundo esse algoritmo, a sincronização de leitura-gravação:

1- Seja L uma operação de leitura: LER (x)

Se $\{$ts-LER (x)$\}$ > mínimo $\{$ts-gravar$\}$ para alguma operação de gravação no sistema, então L é "conservada" na memória intermediária (buffer). Em caso contrário, processa-se L

2- Seja G uma operação de gravação: GRAVAR (x)

Se $\{$ts-GRAVAR (x)$\}$ > mínimo $\{$ts-LER$\}$ para alguma operação de leitura no sistema, então G ficará na memória intermediária. Em caso contrário, se processa G.

3- As operações "conservadas" na memória intermediária são "restauradas" para um futuro processamento.

5.4 - ALGORITMOS BASEADOS NA TÉCNICA DE BLOQUEIO OU EXCLUSÃO MÚTUA

5.4.1 - INTRODUÇÃO

Existem estratégias que permitem execução de transações enquanto verificam se acessos conflitantes para o mesmo

objeto são executados na ordem inicialmente definida quando no lançamento das transações. Se essa ordem não é respeitada uma das transações de conflito é iniciada novamente. Entretanto, estas estratégias podem ser descritas como técnicas de detecção de escalonamentos não serializados que são corrigidos pelo reínicio de certas transações. Em contradição, estratégias de exclusão mútua consistem em evitar a geração de escalonamentos incorretos através do atraso de uma ou mais transações que tentam executar operações de conflito no mesmo objeto.

O primeiro objetivo dos algoritmos de bloqueio é permitir que só execuções simultâneas daquelas operações que são compatíveis ocorram. As operações compatíveis são operações não geradoras de conflitos.

5.4.2 - PROTOCOLO DE BLOQUEIO OU EXCLUSÃO MÚTUA

Antes de definir protocolo de bloqueio é preciso conhecer a propriedade que caracteriza uma operação e que permite determinar sua compatibilidade com outras operações. Esta propriedade é chamada de modo de operação. Os modos mais comuns de operação são as operações de LER e GRAVAR. Geralmente é possível determinar a compatibilidade desses modos através de uma matriz simétrica C chamada matriz de compatibilidade [22]. Assim, por exemplo, a matriz de compatibilidade para os modos LER e GRAVAR, (sendo M_0 = Modo-LER e M_1 = Modo-GRAVAR), é a seguinte:

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Essa matriz é definida tendo os seguintes elementos:

C_{ij} = 1 se os modos de operação são compatíveis e

C_{ij} = 0 no caso contrário

Um escalonamento permitiria somente a execução simultânea do mesmo objeto de operações, havendo modos compatíveis. Para isso, deve-se conhecer quando uma operação inicia em um grânulo assim como os modos de operação requerido na execução de uma operação. Do mesmo modo, é necessário indicar para o escalonador que controla o objeto que uma operação foi completada. Com esta finalidade, duas ações especiais são introduzidas (22).

BLOQUEIO (G,M): Permite que uma transação informe ao escalonador de controle de um objeto g o início de uma operação sobre G, de modo M.

LIBERAÇÃO (G): Permite que uma transação informe ao escalonador de controle de um objeto G, o fim de uma operação sobre G.

O protocolo de bloqueio pode-se expressar mediante as seguintes regras (16):

1- Antes da execução de uma operação de leitura, a transação deve executar um BLOQUEIO sobre pelo menos uma cópia do objeto a se ter acesso,

2- Antes da execução de uma operação de gravação, a transação deve executar um BLOQUEIO sobre cada cópia do objeto a se ter acesso,

3- Cada transação deve executar uma ação de LIBERAÇÃO sobre o objeto selecionado depois da execução de uma operação.

Assim, o protocolo exposto requer a execução de duas ações especiais por operação: BLOQUEIO e LIBERAÇÃO.

Um algoritmo de exclusão mútua requer a definição de uma estratégia para o caso em que os modos da operação requisitada durante a execução da primitiva BLOQUEIO ("LOCK") não são compatíveis com os modos das operações executadas concorrentemente

A estratégia mais simples consiste em retornar a mensagem "objeto ocupado" para a transação. A fim de controlar os estados de espera, é preferível colocar a requisição em estado de espera e bloquear a transação até o objeto requisitado tornar-se disponível. Uma fila de espera Q [g] pode então ser associada a cada objeto ocupado. Esta fila de espera contém as requisições bloqueadas, que são colocadas na fila pela ordem de chegada.

Assim, as regras de bloqueio para os modos LER e GRAVAR, segundo a matriz de compatibilidade é o protocolo de bloqueio já expostos, podem ser vistas como descrito na Fig. 5.11.

T_2 \ T_1	BLOQUEIO PARA LEITURA	BLOQUEIO PARA GRAVAR
Tentar bloquear φ para LER	COMPATÍVEL - T_2 continua a execução	T_2 é colocada na fila de espera de φ
Tentar bloquear φ para GRAVAR	T_2 é colocada na fila de espera de φ	T_2 é colocada na fila de espera de φ

Fig. 5.11 - REGRAS DE BLOQUEIO PARA LER/GRAVAR

O protocolo de bloqueio por si só não é suficiente para garantir serialização [3,16,22] por isso é preciso introduzir restrições nas regras de bloqueio para atingir o objetivo da serialização.

5.4.3 - PROTOCOLO DE BLOQUEIO BIFÁSICO

O protocolo de bloqueio-bifásico pode ser considerado como uma regra de restrição [4] do protocolo de bloqueio com a finalidade de que o processamento de transações seja em série. Nesta restrição o bloqueio ("Locks") sobre um objeto de dados e governado por duas regras [3]:

1- Transações diferentes não podem ter simultaneamente bloqueios em conflito;

2- Uma transação não pode bloquear novos dados depois da liberação de algum dado.

A definição de bloqueios em conflito depende do tipo de sincronização que está sendo executado. Numa sincronização lg (leitura-gravação) dois bloqueios estão em conflito se:

a) Ambos são bloqueios sobre o mesmo dado; e

b) Uma ação é um bloqueio de leitura, e a outra é um bloqueio de gravação;

Para uma sincronização gg dois bloqueios estão em conflito se:

a) Ambos são bloqueios sobre o mesmo dado; e

b) Ambos são bloqueios aplicados à gravação.

Muitos sistemas introduzem as operações de LIBERAÇÃO ("unlock") no final das transações ou em certos pontos intermediários onde o Banco de Dados é consistente. A interface BLOQUEIO/LIBERAÇÃO é transparente aos usuários e sempre existe pelo menos uma interfase lógica equivalente dentro do sistema (22).

A segunda regra de apropriação de bloqueios obriga que cada transação tenha um comportamento bifásico através do tempo em relação ao número de objetos bloqueados. De acordo com a Fig. 5.12, a primeira fase é de crescimento (desde o início da transação até o ponto de bloqueio máximo), a transação obtem locks, sem a liberação de qualquer outro bloqueio. Devido à liberação de um bloqueio, a transação ingressa na fase de contração, a segunda fase (desde o máximo até o término da transação. Durante esta fase a transação libera os bloqueios,

e pela regra 2 é proibido o pedido de bloqueios adicionais.Quan

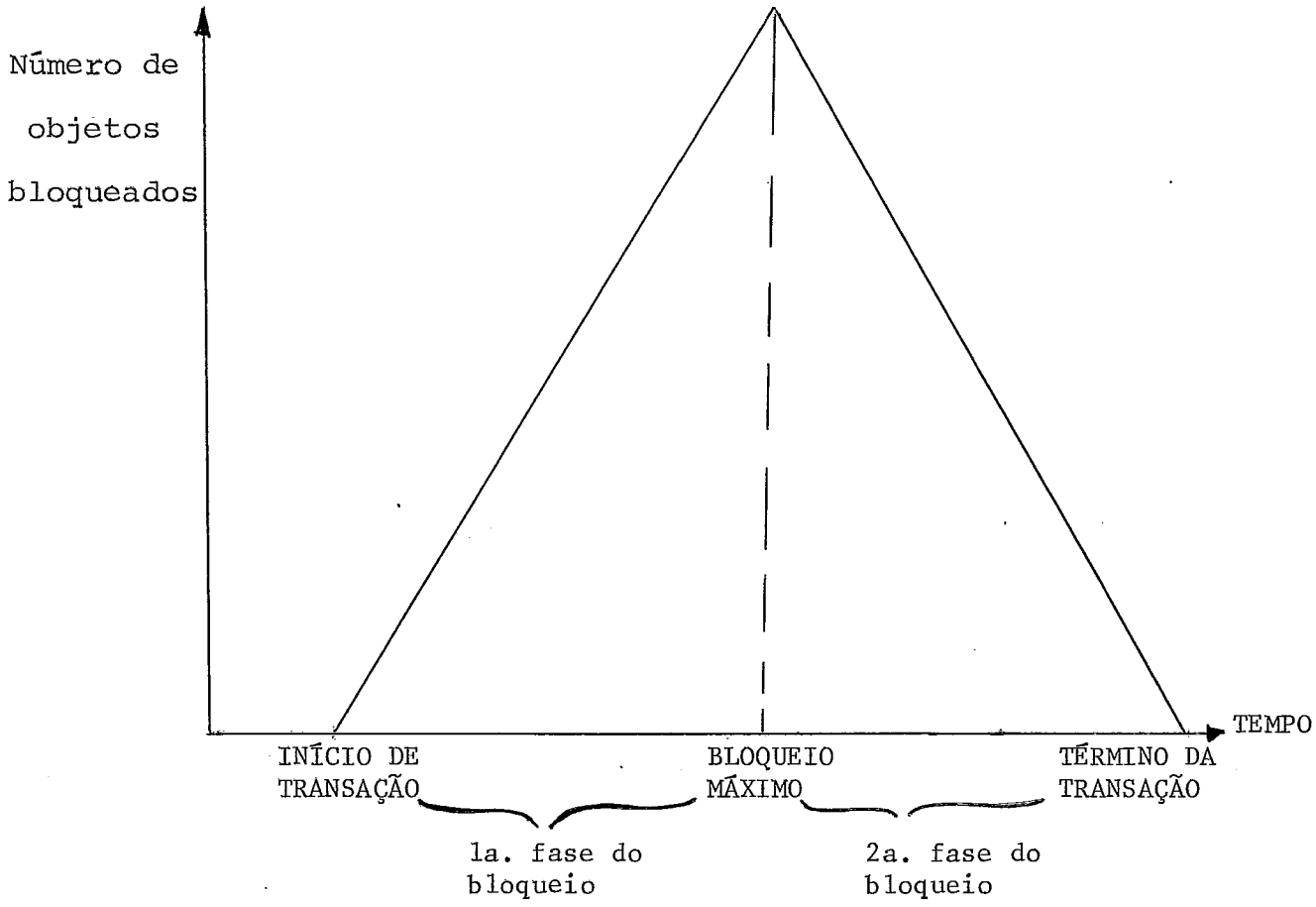


Fig. 5.12 - VARIAÇÃO NO NÚMERO DE OBJETOS BLOQUEADOS NUMA TRANSAÇÃO BIFÁSICA

do a transação termina ou aborta, todos os bloqueios adicionais são automaticamente liberados.

Demonstra-se [34,20,22] que o bloqueio bifásico é suficiente para garantir o processamento seriado, visto que nenhuma transação requer novos bloqueios depois de abandonar um dado. Isto aumenta a possibilidade de ter-se transações retendo todos os bloqueios antes de acabar sua execução. A ordem de serialização é determinada pela ordem de obtenção de bloqueios das transações [5]. Finalmente, o bloqueio bifásico requer que cada uma das transações passe através de um estado de blo

queio máximo onde todos os objetos a que tem acesso são bloqueados, e este máximo (i.e. o bloqueio máximo na fig. 5.12) determina a ordem das transações.

5.4.4 - ALGORITMOS DE CONTROLE CENTRALIZADO DO BLOQUEIO BIFÁSICO

A aplicação do bloqueio bifásico requer a seleção de um ou mais nós para execução dos algoritmos de BLOQUEIO/LIBERAÇÃO. Aliás, visto que os objetos de dados são distribuídos sobre vários computadores, o problema é saber quem vai gerenciar os estados de bloqueio dos objetos e as filas de pedido de bloqueio. Uma das soluções desse problema é o controle centralizado onde um simples nó toma conta do lock/UNLOCK.

5.4.4.1 - BLOQUEIO BIFÁSICO CENTRALIZADO

Nesta técnica o escalonador de transações é centralizado num simples nó, chamado "o nó primário". Quando uma transação deseja ter acesso a um determinado objeto do Bando de Dados, o lock deve ser obtido do nó primário. Uma vez que a transação possui o lock, pode ter acesso a qualquer objeto de dados em qualquer nó. É importante notar [5] que, uma transação atualiza um objeto que tem muitas cópias armazenadas, todas as cópias são efetivamente atualizadas antes que o lock sobre esse objeto seja liberado. É também importante que transações de só leitura respeitem essa disciplina para evitar leitura in

consistente de dados.

Uma desvantagem dessa técnica é o alto custo de comunicação, aliás, a capacidade do nó primário para processar locks pode limitar a capacidade do sistema distribuído indo contra a performance do Banco de Dados.

5.4.4.2 - BLOQUEIO BIFÁSICO COM CÓPIAS PRIMÁRIAS

Esta técnica é uma extensão do bloqueio bifásico centralizado que elimina a desvantagem de ter só um nó central. Para cada objeto de dados lógico, uma cópia é designada como a "cópia primária"; quando uma transação deseja ter acesso a um objeto, o lock apropriado deve ser aplicado sobre a cópia primária [5].

Para aplicar locks de leitura esta técnica requer mais comunicação do que a implementação básica. Suponha-se que x_1 é a cópia primária de algum dado X, e suponha-se que a transação T quer ler alguma outra cópia X_i de X. Para ler X_i, T deve comunicar-se com os dois gerenciadores de dados: o gerenciador onde x_1 é armazenado (de modo que T possa aplicar o bloqueio x_1) e o gerenciador onde x_1 é armazenado.

Uma desvantagem deste algoritmo é que pode-se apresentar o problema de deadlock; que será visto mais adiante.

5.4.5 - ALGORITMOS DE CONTROLE DISTRIBUÍDO DO BLOQUEIO BIFÁSICO

O controle distribuído é feito quando cada escalonador é responsável pelo BLOQUEIO/DESBLOQUEIO dos objetos que são gerenciados pelo SGBD local associado. Neste método tem-se um controlador de cada nó cuja tarefa é fazer o bloqueio bifásico para todas as transações locais.

Uma implementação básica do bloqueio-bifásico distribuído é construir um escalonador, i.e.,um módulo de software que receba pedidos de bloqueio e desbloqueio e os processe de acordo com o protocolo de bloqueio bifásico. Este módulo seria distribuído através de todos os gerenciadores de dados.

Se um pedido de bloqueio não pode ser atendido, a operação é colocada sobre uma linha de espera do objeto requerido (Esta situação pode causar um bloqueio perpétuo, que será estudado na continuação).Bloqueios de gravações são implicitamente renovados pelas operações de gravação. Contudo, para renovar os bloqueios sobre leituras requerem-se operações especiais de renovação de bloqueios. Estas renovações de bloqueios podem ser transmitidas em paralelo com as operações de gravação, dado que as operações de gravação sinalizam o início da fase de contração. Quando um bloqueio é liberado, as operações sobre a linha de espera dos objetos de dados são processadas na ordem "FIRST-IN, FIRST OUT". Este método evita a espera indefinida, que é um dos típicos problemas das técnicas de bloqueio.

5.4.6 - O PROBLEMA DO BLOQUEIO-PERPÉTUO "DEADLOCK"

O problema do Bloqueio-perpétuo surge quando os objetos do Banco de Dados são bloquedos em tal sequência, que um grupo de transações possui as seguintes duas propriedades (14).

1- Cada uma das transações é bloqueada, esperando por um objeto;

2- A execução de cada uma das transações que não pertence ao grupo não permite o desbloqueio de qualquer transação no grupo.

O bloqueio perpétuo é um problema gerado por "locking" e é difícil de resolver. O problema do bloqueio perpétuo aparece porque nas técnicas de bloqueio bifásico as transações podem estar esperando locks que não são disponíveis. Se essas esperas não são controladas, o deadlock pode aparecer (Veja-se figura 5.13)

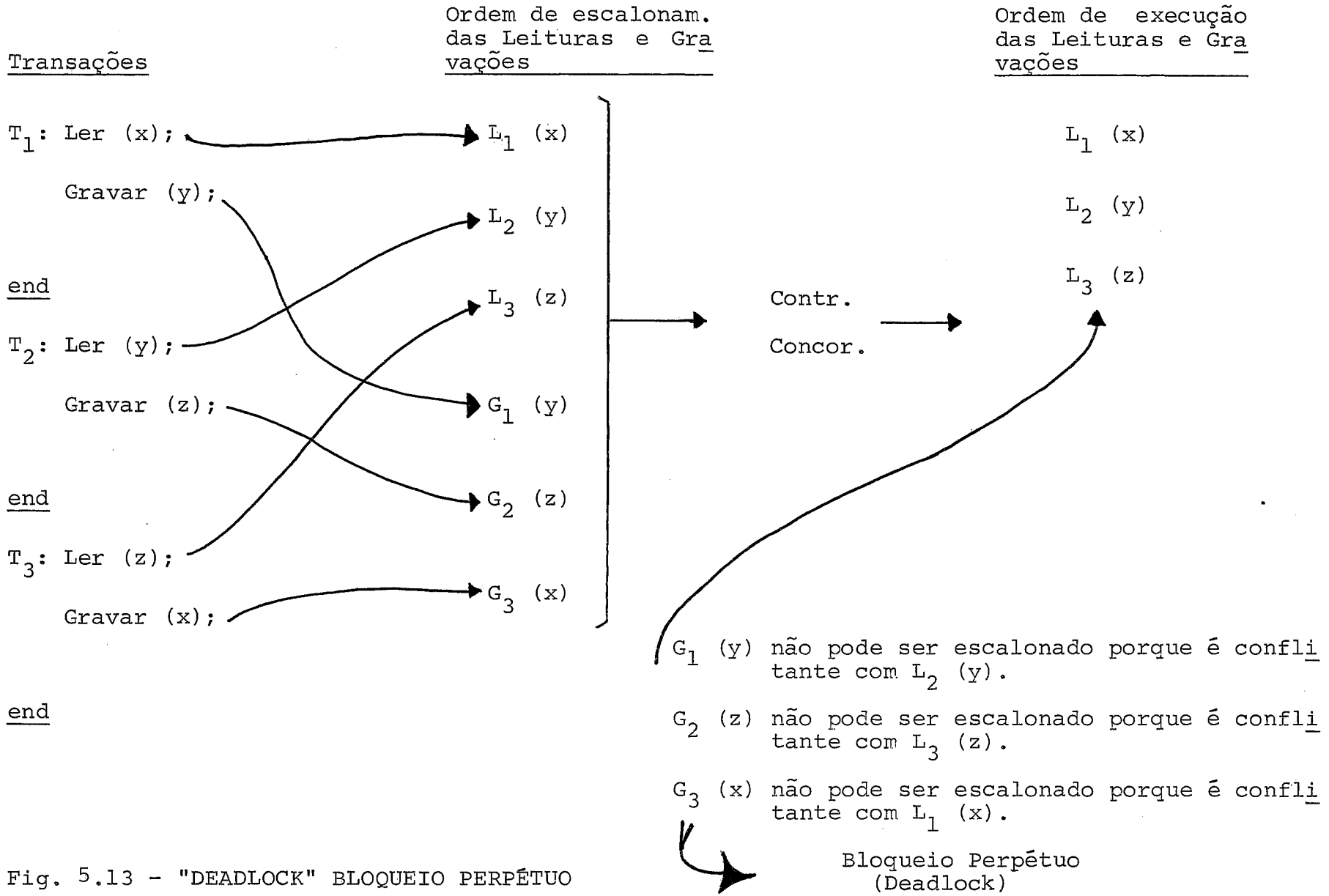


Fig. 5.13 - "DEADLOCK" BLOQUEIO PERPÉTUO

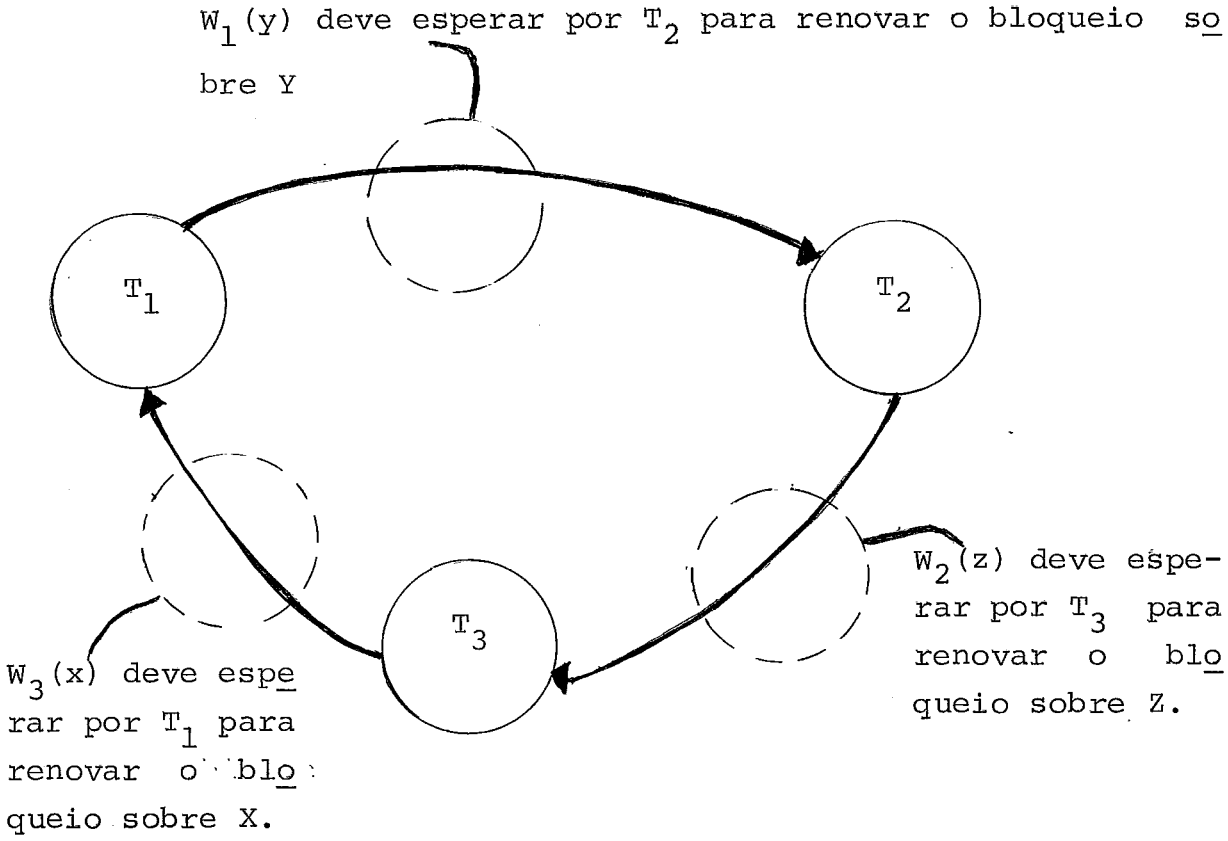


Fig. 5.14 - GRAFO DO DEADLOCK PARA A Fig. 5.13

Os bloqueios perpétuos podem ser detectados mantendo grafos de Deadlocks locais e globais [14] (Veja-se Fig. 4.14). Os vértices do grafo representam transações e as arestas representam o relacionamento "esperando-por"; uma aresta é estabelecida da transação T_i à transação T_j se T_i está esperando por um "lock" retido por T_j tem-se um bloqueio perpétuo no sistema se o grafo do "deadlock" é cíclico. Se um deadlock existe, alguma transação no ciclo é abortada e restaurada posteriormente. Esta técnica de eliminação pode causar RESTAURAÇÕES CÍCLICAS [22]. Uma forma simples de evitar este problema, é abortar sempre a transação mais nova no sistema [22]. O custo maior do deadlock é a restauração de transações executadas

parcialmente.A pre-declaração pode ser usada para reduzir este custo. Para obter o bloqueio de uma transação antes da execução, o sistema restaurará somente transações que tenham sido executadas.

Quando um computador deseja verificar a presença do bloqueio perpétuo, coleciona partes do grafo de deadlock a partir de cada um dos computadores da rede. Essas partes do grafo podem ser representadas por uma matriz booleana $G = (G_{ij})$ onde $G_{ij} = 1$ se a transação T_i espera pela transação T_j e $G_{ij} = 0$ no caso contrário. Assim, com a união das diferentes matrizes, o computador pode obter a matriz completa representando o grafo de esperas.

5.5 - EXEMPLOS DE APLICAÇÃO

5.5.1 - CONTROLE DE CONCORRÊNCIA NO SDD-1

O método de controle de concorrência no SDD-1 é qualitativamente diferente dos métodos já expostos. No SDD-1, a informação acerca do conflito entre transações é pré-analisada antes do ingresso das transações no sistema. A pre-análise é o mecanismo principal do SDD-1 para o controle de concorrência.

Internamente, o SDD-1 consiste de dois tipos de módulos; o primeiro é o módulo gerenciador de transações (GT) e o segundo é o módulo gerenciador de Dados (GD).

A execução de cada uma das transações é processada em 3 fases [29]: leitura, execução e gravação. Durante a fase de leitura, o SDD-1 estabelece o Conjunto de Leitura ("READ-SET")

i.e. as partes do Banco de Dados lógico que serão lidas. Quando todas as mensagens de leitura forem processadas, termina a fase de leitura. Na fase de execução o GT supervisiona a execução das transações. Durante a fase de gravação, produz-se uma lista de dados que serão gravados no banco. Na pré-análise também se determina o Conjunto de Gravação ("Write set") i.e. as partes do Banco de Dados que serão atualizadas.

Um aspecto importante do mecanismo de controle de concorrência do SDD-1 é distinguir as transações que requerem sincronização. Examinando o conjunto de leitura e o conjunto de gravação, determina-se o conflito existente entre transações. Duas transações estão em conflito se o conjunto de gravação de uma delas intercepta o conjunto de gravação ou conjunto de leitura da outra transação [3].

O SDD-1 usa dois mecanismos de controle para atingir seus objetivos:

1º) Análise do grafo de conflitos

2º) Protocolos baseados sobre "Timestamps"

5.5.1.1 - ANÁLISE DO GRAFO DE CONFLITOS

A análise do grafo de conflitos é uma técnica para determinar que conflitos requerem sincronização. O método começa com a definição de Classes de Transações. Formalmente cada classe de transação é definida por seu Conjunto de Leitura e seu Conjunto de Gravação. Uma transação pertence a uma classe se o conjunto de leitura e o conjunto de gravação da transação estão contidos (respectivamente) no conjunto de leitura e con

junto de gravação da classe. Uma transação pode pertencer a muitas classes.

As definições do conjunto de leitura e conjunto de gravação são expressas usando restrições simples (Fig. 5.15) de tal modo que os membros de uma classe possam ser checados rapidamente. Associado a uma classe tem-se um módulo gerenciador de transações (GT) que processa em série as transações pertencentes a essa classe. Visto que transações numa classe simples se processam em série, só transações em diferentes classes podem "interferir". Assim, só conflitos entre classes precisam ser considerados.

As transações de diferentes classes podem estar em conflito se suas classes, também diferentes, estão em conflito. Assim, os conflitos entre transações podem ser determinados por conflitos entre classes. Conflitos entre classes são modelados por grafos de conflitos. (Fig. 5.15). Um grafo de conflito é um grafo não direcionado que faz o resumo dos conflitos potenciais entre transações de diferentes classes. Para cada classe C_i o grafo contém dois vértices, chamados l_i e G_i. As arestas do grafo são definidas da seguinte forma [3,29].

1- Para cada classe C_i tem-se uma aresta vertical entre l_i e G_i;

2- Para cada par de classes C_i e C_j $(i \neq j)$ tem-se uma aresta horizontal entre w_i e w_j se e só se o Conjunto de Gravação C_i intercepta-se com o conjunto de gravação C_j

3- Para cada par de classes C_i e C_j $(i \neq j)$ tem-se uma aresta diagonal entre T_i e w_j se e só se o con

junto de leitura C_i intercepta o conjunto de gravação C_j.

Intuitivamente, uma aresta horizontal indica que o escalonador E pode ser forçado a retardar as operações de gravação com a finalidade de sincronizar as operações gg (gravação-gravação). Suponha-se que as classes C_i e C_j são ligadas por uma aresta horizontal (w_i, w_j) indicando que existe interseção entre seus conjuntos de gravação. Se o escalonador E recebe uma operação de gravação proveniente de C_i, deve-se retardar a execução desta operação até receber as operações de gravação com "Timestamps" menores provenientes de C_i. Similarmente, uma aresta diagonal indica que S pode precisar de retardar operações para efeito de sincronização l_g (leitura-gravação).

Assim, a análise do grafo de conflitos otimiza a situação pela identificação de conflitos entre classes antes do início da execução das transações.

No SDD-1, o grafo de conflito é construído e analisado estatisticamente quando o banco de dados e as classes são definidos. Classes que não formam ciclos são anotadas e o GT correspondente não usará sincronização na execução das transações. Os GTs restantes devem sincronizar suas transações.

SDD-1 não usa a técnica de Bloqueio/Desbloqueio para sincronização, em vez disso usa Protocolos baseados sobre Time stamps que serão estudados a seguir.

Fig. 5.15 - CLASSES DE TRANSAÇÕES E GRAFOS DE CONFLITOS

. Uma classe é definida por um conjunto de leitura e um conjunto de gravação:

C_1: Conjunto de leitura = X_1, ; Conjunto de gravação = Y_1, Y_2

C_2: Conjunto de leitura = x_1, x_2 ; Conjunto de gravação Y_1, Y_2, Y_2, Y_3

C_3: Conjunto de leitura = Y_2, z_3 ; Conjunto de gravação x_1, z_2, z_3

. Uma transação pertence a uma classe se seu conjunto de leitura é um subconjunto do conjunto de leitura na classe e seu conjunto de gravação é um subconjunto do conjunto de gravação da classe.

T_1: Conjunto de leitura = x_1 , Conjunto de gravação = Y_1, Y_2

T_2: Conjunto de gravação = Y_2 , Conjunto de gravação = Z_2, Z_3

T_3: Conjunto de leitura = Z_3 , Conjunto de gravação = X_1

. T_1 pertence à classe C_1 e C_2

. T_2 pertence à classe C_2 e C_3

. T_3 pertence à classe C_3

OBS: NA FIG. 5.16 APRESENTA-SE O GRAFO DE CONFLITOS PARA ESTAS CLASSES DE TRANSAÇÕES.

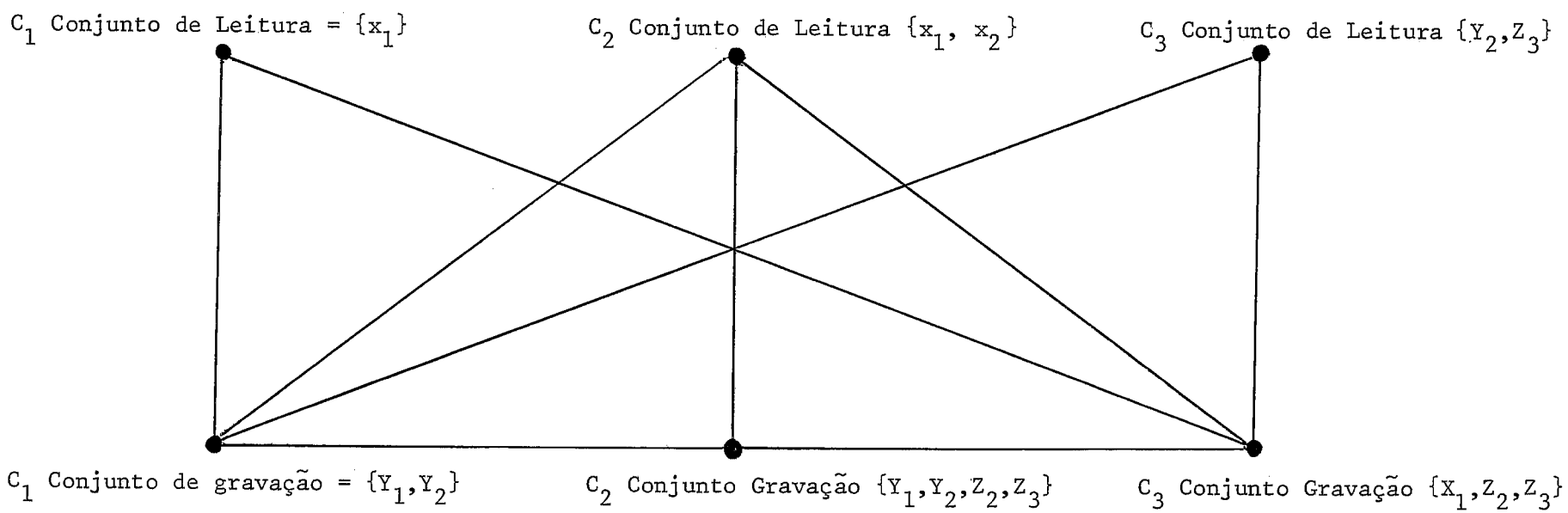


FIG. 5.16 - CLASSES DE TRANSAÇÕES E GRAFOS DE CONFLITOS

5.5.1.2 - PROTOCOLOS BASEADOS SOBRE "TIMESTAMPS"

Neste estudo se fará apenas um esboço da técnica de sincronização empregando protocolos baseados sobre time stamps.

Cada aresta proveniente do conjunto de leitura de uma classe ao conjunto de gravação de outra classe representa um conflito que deve ser sincronizado. Essa sincronização acontece durante o processamento dos comandos de leitura. Suponha-se que o conjunto de leitura de uma classe i está em conflito com o conjunto de gravação de uma classe i. Suponha-se também que T_i é uma transação na classe i. Para processar um comando de leitura para T_i no nó S, o controlador de concorrência espera até que o S cumpra as seguintes duas condições:

1- Tenha processado todos os comandos de gravação das transações na classe j que são mais antigas que T_i (SDD-1 nomea "timestamps" únicos às transações)

2- Não tenha processado os comandos de gravação das transações na classe j que são mais "novos" do que T_i.

Esse procedimento tem o mesmo efeito do que o bloqueio dos dados compartilhados pelo conjunto de leitura de i e o conjunto de gravação j.

VI - CONCLUSÕES

1- O critério principal de distribuição de dados num SGBDD é armazenar os dados onde eles são mais frequentemente usados, de modo que muitas consultas possam ser resolvidas localmente. Contudo, atingir um rendimento ótimo é uma tarefa muito complexa e a performance depende principalmente dos seguintes fatores:

- A percentagem de interrupções de acesso, i.e., acessos que não são resolvidos localmente, de modo que se tem que usar o diretório para fazer a transferência de acesso ao nó apropriado;
- Frequência de atualização;
- Número de nós;
- Quantidade de dados;
- Custos de operação e comunicação

2- O método mais apropriado de distribuir pequenas quantidades de dados é simplemente replicar se a frequência de atualização em tempo real é baixa, porque o custo de armazenamento é baixo.

Quando as atualizações (não frequentes) ocorrem, elas são enviadas a todos os nós que tem cópias de dados.

3- Particionamento e Replicação representam modelos "extremos" ou "puros". Na prática, um usuário deseja usar uma mistura arbitrária dos dois métodos, algumas vezes para o mesmo dado. Por exemplo, um banco de dados pode ser replicado inteiramente sobre dois nós e particionado sobre todos os outros

nós. Em outras situações, o banco de dados pode ser em sua grande parte particionado, mas em determinadas partes podem ser replicadas em nós selecionados para obter benefícios de performance e confiabilidade.

4- Chu [12] foi o pesquisador pioneiro na área de distribuição física de dados. Ele propôs uma solução em termos de programação linear para otimizar a alocação de arquivos na rede com as seguintes premissas:

1º) O número de cópias de arquivos é conhecido;

2º) Consultas são encaminhadas a todos os arquivos;

3º) Os modelos das consultas são conhecidos;

4º) Existem restrições para o tempo de resposta o que é uma contribuição importante do modelo.

A solução formulada é baseada no método de programação inteira zero-um. A solução só é válida para problemas pequenos, [13,48] visto que o número de variáveis e restrições aumenta rapidamente quando cresce o número de nós no problema.

5- Casey [9] não leva em conta as limitações impostas por Chu [12] e demonstra que assumindo-se que os custos de comunicação e atualização são iguais, então pode estabelecer um limite superior sobre o número de cópias do arquivo. Por exemplo, se o arquivo é tão atualizado ou mais do que é consultado então uma cópia simples do arquivo é ótima [12]. A principal vantagem do modelo de Casey é a simplicidade mas as maiores desvantagens estão nas suas premissas:

1º) Só um arquivo é considerado;

2º) Os tempos de espera não são considerados;

3º) Não apresenta restrições de capacidades;

4º) Limites sobre o número ótimo de cópias de arquivos são disponíveis só quando os custos de comunicação das consultas e atualizações são iguais.

Casey, pesquisa o problema das condições ótimas para criar cópias múltiplas de um arquivo. RAMAMOORTHY C. et alii [40] considera o algoritmo de Casey [9] muito mais eficiente e simples do que o algoritmo de Chu [12].

6- Os modelos anteriormente citados estudam o problema de distribuição de dados com uma função de custos que não consideram o fato de que os acessos aos arquivos de dados são feitos através de programas. Morgan et alii [37], pela primeira vez no estudo do problema de distribuição de arquivos, considera que os usuários nos vários nós obtêm acessos aos arquivos de dados através de programas de atualização e programas de consulta. Uma desvantagem deste modelo é proibir restrições na capacidade de armazenamento já que uma de suas premissas é de que a alocação do arquivo de dados i é independente da alocação do arquivo de dados j.

7- As desvantagens principais dos métodos heurísticos são:

1) Sempre acha-se um ótimo local em vez de um ótimo global;

2) A validade do algoritmo é muito difícil. Além do mais, é possível determinar os valores que satisfazem uma boa execução, mas é difícil di

zer quando e em que condições tem-se um comportamento ruím;

3) Não permite formular dependências entre programas e arquivos de dados e ignora capacidades de armazenamento o qual leva a resultados indesejáveis [18].

8- A vantagem do modelo de MAHMOUD-RIORDON [34] é que permite a simultânea alocação de arquivos e capacidade de comunicação, restringido pela demora nas mensagens e pela disponibilidade de arquivos.

A aplicação do método é baseada na premissa de que, com alta probabilidade, uma ou mais das soluções ótimas achadas estarão perto, em termos do custo, do ótimo global.

9- O modelo de CHANDY & HEWES [18] é especialmente interessante por sua simplicidade na aplicação de procedimentos iterativos, mas sua desvantagem é assumir que todos os arquivos são independentes o que impede controlar a performance do sistema.

10- Os métodos heurísticos precisam de maior pesquisa já que eles fornecem soluções práticas para o problema de alocação de arquivos. Estas soluções têm que considerar, além do mais, considerações para o problema de migração de arquivos ("o problema dinâmico de alocação de dados") no qual os arquivos são permitidos mudar no percurso do tempo com a finalidade de adaptar-se às mudanças dos requerimentos de acesso.

11- Considerando a premissa de que o custo de transmissão é muito mais alto do que o custo de armazenamento, para baixos níveis de atualização do diretório, o diretório distribuído deve ter custos operacionais menores do que o diretório centralizado. Por outro lado, no sistema de diretórios locais, devido ao grande custo de comunicação associado quando um arquivo não está armazenado no diretório local do usuário correspondente, o custo de operação deve ser maior do que o diretório centralizado.

12- Nas técnicas de processamento de consultas do BDD o critério de otimização das estratégias não é único e constitui o problema central no desenvolvimento de projetos de BDD:

- Cellary et alii [2] considera o tempo de resposta como critério de otimização. Segundo suas pesquisas, este tempo é o elemento principal na avaliação da eficiência do sistema. Este tempo consiste de tempos de execução local das subconsultas e operações relacionais mais os tempos de transmissão.
- No SDD-1 [1] o critério de otimização é baseado na minimização da quantidade de transferência dos dados entre nós e a maximização do processamento em paralelo. O primeiro é o fator dominante dos custos de processamento das consultas.
- No INGRES [45] considera-se a minimização do tempo de resposta e do tráfego na rede como dois critérios principais de otimização. Esses

dois critérios estão relacionados: Um aumento no tráfego da rede aumentará o tempo de resposta se resulta num processamento em paralelo maior.

- HEVNER e alii [26] considera o tempo de resposta e o tempo total de execução como dois critérios de avaliação da estratégia do algoritmo. Ele apresenta um algoritmo geral [26] para toda classe de consultas para atingir uma estratégia de ótima qualidade.

As técnicas de otimização na estratégia de processamento de consultas usam dois modelos de otimização: Estáticas e Dinâmicas. As soluções estáticas usam como base de otimização, informação estatística concernente ao dado armazenado e são avaliadas com antecedência. Por outro lado nas soluções dinâmicas a avaliação da execução e tempos de transmissão são feitas passo a passo durante o processamento da consulta (Ex. INGRES distribuído).

13- As estratégias centralizadas não são as melhores em muitos casos práticos devido à diminuição do paralelismo de processamento no sistema. Por outro lado, as estratégias distribuídas, tomam maior vantagem do alto paralelismo do sistema. Frente a esta afirmação os implementadores do SDD-1 [4] fazem a seguinte análise: "Os programas construídos por nosso procedimento de otimização consistem de uma fase de redução seguido por uma fase de processamento final. A fase de redução executa todas as semijunções que têm uma relação custo-benefício

proveitosa permitida pela consulta numa forma distribuída, enquanto a fase final executa todas as junções precisadas para resolver a consulta numa forma centralizada. Contudo, é algumas vezes melhor executar as junções numa forma distribuída também, e assim a execução de algumas junções podem ser mais vantajosas. Isso sugere uma estratégia alternada de processamento de consultas com a seguinte estrutura:

DO WHILE A consulta não tem solução

i) Executar todas as semijunções rentáveis ou vantajosas.

ii) Executar uma ou mais junções

END

Evidentemente, o novo fator crítico de otimização é selecionar as junções que são processadas em cada uma das iterações.

14- Uma característica comum das estratégias desenvolvidas é que elas consideram a otimização do processamento da consulta como se o SGBDD estivesse dedicado a resolver os requerimentos de um simples usuário. Não considera-se o fato do processo simultâneo de consultas múltiplas em múltiplos nós. Cellary et alii [10] apresentam um modelo que considera esses fatos.

15- Neste estudo apresenta-se uma estrutura básica para projetar e analisar algoritmos de controle de concorrência. A estrutura tem dois componentes principais:

a) Uma terminologia comum e conceitos básicos para descrever duas técnicas de sincronização:Bloqueio bifásico e ordenação por "timestamps". A

ra transações executadas não frequentemente, que substitui uma sincronização mais rígida sobre essas transações a fim de reduzir os requisitos de sincronização para transações comuns. Em adição, todos os protocolos usam "timestamps" para resolver conflitos, assim os "bloqueios perpétuos" são previnidos sem necessidade de se gastar tempo com um algoritmo de deteção. Logo, pode-se afirmar que os algoritmos baseados na exclusão mútua são suscetíveis ao "bloqueio perpétuo" enquanto que os algoritmos baseados na ordem dos "timestamps" não apresentam esse problema.

18- As principais desvantagens dos algoritmos de controle de concorrência baseados na ordem de escalonamento são as seguintes:

- Precisa-se muita memória para armazenar os "timestamps". Uma solução a este problema é o esquecimento dos "timestamps" antigos, armazenando-os em tabelas pequenas as quais são periodicamente atualizadas [3].
- O tratamento das "transações abortadas" podem criar conflitos repetidos e o adiamento indefinido das transações.
- Altos custos de comunicação pela frequência de bloqueios (i.e., no caso de algoritmos conservativos) e "overhead" no caso dos algoritmos que usam processos de reinícios ("restarts").
- Muitas vezes as transações tem que esperar (i.e. aumenta o tempo de resposta) com a finalidade

análise destas duas técnicas é o foco principal desta estrutura.

b) Uma separação do algoritmo de controle de concorrência em sub-algoritmos de sincronização leitura-gravação e gravação-gravação.

Nos algoritmos baseados sobre a técnica de "timestamp" a ordenação pode ser controlada pelo usuário. As transações podem ser pré-analisadas antes de sua execução (como no SDD-1).

Por outro lado, nos algoritmos baseados na técnica de bloqueio/desbloqueio a ordenação é imposta pelo sistema e não pode ser influenciada pelos usuários. Se requer o máximo bloqueio antes do desbloqueio que define a ordem do processamento em série.

16- Dado um conjunto de transações a ser executadas concorrentemente:

a) A técnica do bloqueio bifásico garante que a execução concorrente é equivalente a alguma execução em série imprecisa dessas transações.

b) A técnica de "timestamps" garante que a execução concorrente é equivalente a uma execução em série específica dessas transações — i.e., definida pela ordem dos "timestamps".

17- A vantagem dos protocolos baseados nos "timestamps" gira em torno dos protocolos que podem ser usados, ou seja, há apenas um protocolo especial de transações de leitura que é mais eficiente que a exclusão mútua. Há um protocolo especial pa

de conhecer que transações com "timestamps" menores têm sido executadas. Assim, uma transação baseada nesta técnica pode esperar o acesso ao banco de dados de qualquer outra transação com um "timestamp" menor proveniente de qualquer nó: No SDD-1 [2] existe um método de restrição do número de nós que podem causar conflitos para superar essa desvantagem.

19- Uma pré-análise das transações no SDD-1 pode melhorar o seu desempenho. Se muitas transações executam-se em classes que não requerem sincronização, então o sistema requerirá poucas mensagens. As classes podem (em princípio) ser projetadas de modo que as mais frequentemente acessadas não formem ciclos (formem grafos acíclicos) para assim evitar conflitos.

20- Com o objetivo de avaliar o sistema SDD-1, BERSTEIN et alii [1] definem o custo de uma operação como a quantidade de dados (i.e., número de bytes) que devem ser enviados entre nós para executar a operação; enquanto benefício é a quantidade de dados que se eliminam.

Restrições e projeções podem ser calculadas com transferência de dados inter-nós igual a zero (custo de operação = 0) e benefício não negativo. Assim esses operandos produzem uma relação custo-benefício ótima. Por outro lado, semijunções requerem movimentos inter-nós sempre que seus operandos estejam armazenados em diferentes nós. Assim, o custo de semijunção depende do estado do Banco de Dados. Mas, semijunções podem

ser calculadas com menor custo do que junções na maioria das vezes [1]. Para calcular $R_i \langle A = B] R_j$, só precisa-se transmitir a projeção de uma relação (i.e. R_j [B]) enquanto para calcular R_i [A = B]R_j deve-se transmitir a relação toda.

Baseado em modelos estatísticos do Banco de Dados, demonstrou-se [1] mediante um procedimento de redução de custos e benefícios que o operador de semijunção é o mais efetivo operador de redução no processamento de consultas em SDD-1.

21- Com a finalidade de avaliar o desempenho dos algoritmos de controle de concorrência, alguns pesquisadores [36] afirmam que os mecanismos de controle de concorrência baseados na técnica de bloqueio são chamados pessimistas dado que os recursos do Banco de Dados são bloqueados mesmo na ausência de conflitos entre transações em execução. Enquanto que os algoritmos baseados na ordem de escalonamento são chamados otimistas, porque eles consideram os conflitos, situações infrequentes e por esta razão, ações apropriadas para preservar integridade devem ser executados só quando os conflitos aparecem.

Segundo essa classificação, Menacé D. e Nakanischi T., [36] construíram um modelo de simulação para comparar os dois tipos de algoritmos tomando em conta o tempo médio de resposta e os diferêntes parâmetros tais como o número de recursos por transação e o tamanho do Banco de Dados.

Em todas as situações analizadas os algoritmos pessimistas têm uma melhor performance que os otimistas. Contudo, tem-se situações na qual os algoritmos otimistas podem ser tão bons quanto os pessimistas.

Isto acontece em situações onde o grau de confli

to entre transações é baixo e quando o sistema de cômputo e tal que a carga extra devido ao reinício de transações podem ser facilmente absolvido pelos recursos disponíveis de CPU e entrada/saída.

22- Na literatura existem poucas pesquisas acerca do desempenho dos algoritmos de controle de concorrência. É difícil saber qual é o melhor.

Berstein [3,5] explica que não se têm modelos de avaliação do desempenho ("performance") baseados sobre uma análise teórica detalhada e extensiva experimentação. Além disso, muitos algoritmos são apropriados para certas aplicações [27] e parâmetros do sistema enquanto outros são mais apropriados para outras aplicações. Contudo, a maioria de pesquisadores definem os principais critérios de avaliação dos algoritmos de concorrência [3,5,22,31] e são os seguintes:

a) "Throughput" do sistema: É definido pelo número de transações por unidade de tempo que podem ser processadas. As técnicas que não aproveitam da vantagem do paralelismo oferecido pelo sistema de BDD atingem um menor throughput [31]

b) Tempo de Resposta;

c) Custos de Comunicação inter-nós;

d) Custos de Processamento Local;

e) Custos de Restauração de Transações;

f) Custos de Bloqueio de Transações;

g) Confiabilidade do sistema ante falhas eventuais de hardware e/ou software e facilidades de

recuperação.

Para outros pesquisadoeres [4,27] um critério importante de avaliação é o GRAU DE CONCORRÊNCIA, i.e., a cardinalidade do conjunto de todos os escalonamentos consistentes que um mecanismo de controle de concorrência pode produzir. Kohler [29] afirma que o Bloqueio Bifásico e a Ordenação "timestamp" são métodos corretos, garantem execuções em série e oferecem poucas diferenças quanto ao grau de concorrência.

Contudo, o objetivo principal de mecanismo de controle de concorrência é permitir o aproveitamento máximo das facilidades do paralelismo oferecido pelos sistemas de gerenciamento de BDD, i.e., permitir a execução concorrente do máximo de transações enquanto se garente a consistência interna e externa (cópias múltiplas) do Banco de Dados.

REFERÊNCIAS BIBLIOGRÁFICAS

1 - ADIBA, M. et alii. Issues in distributed data base management systems: a technical overview. IN: CONFERENCE ON VERY LARGE DATA BASE. Berlin, Oct. 1978, Proceedings on VLDB, 1978.

2 - BERNSTEIN, P.A. et alii. Concurrency control in a system for distributed databases (SDD-1). ACM Transactions on Database Systems, New York, 5(1): 18-51, Mar. 1980.

3 - BERNSTEIN, P.A. & GOODMAN, N. Concurrency control in distributed data base systems. ACM Computing Surveys, New York, 13(2): 185-221, Jun. 1981

4 - BERNSTEIN, P.A. et alii. Query processing in a system for distributed data base (SDD-1). ACM Transactions on Data base Systems, New York, 6(4): 602-25, Dec. 1981

5 - BERSTEIN, P.A. & GOODMAN, N. Approaches to concurrency control in distributed database systems. Cambridge, Mass. Harvard University, Center for Research in Computing Technology, 1978. 50 p. (Technical Report, 26).

6 - BLASGEN, M.W. & ESWARAN, K.P. Storage and acess in relational data bases. IBM Systems Journal, Armonk, NY, 16 (4): 363-97, 1977.

7 - BRACHI, C. et alii. Distributed query processing. IN: DRAFFAN, I.W. & POOLE, F. Distributed data bases Cambridge, Cambridge, University Press, 1980, p. 83-101.

8 - CARLESI, C. et alii. Data distribution criteria and query processing strategy in a small distributed environment. IN: DELOBEL C. & LITWIN, W. Distributed data bases. Amsterdam, North-Holland, 1980, p. 121-136.

9 - CASEY, R.G. Allocation of copies of a file in an information network. IN: AFIPS CONFERENCE PROCEEDINGS. Atlantic N.J., May 16-18, 1972. Spring joint computer conference. Montvale N.J., AFIPS PRESS, 1972, v. 40, p. 617-25

10 - CELLARY, W. et alii. A multiquery approach to distributed processing in a relational distributed data base management systems. IN: DELOBEL, C. & LITWIN, W. Distributed data bases. Amsterdam, North-Holland, 1980. p. 99-119.

11 - CHAMPINE, G. A. Currents trends in data base systems. IEEE Computer, Piscataway, N.J. 12(5): 27-41, May 1979

12 - CHU, Wesley W. Multiple file allocation in a multiple computer system. IEEE Transactions on Computers, Piscataway, N.J., 18(10): 885-9, Oct. 1969.

13 - ———Performance of file directory systems for data bases in start and distributed networks. IN: AFIPS CONFERENCE PROCEEDINGS, New York, N.Y., June 577-10, 1976.

National computer conference. Montvale, NJ, AFIPS Press, 1976, v. 45, p. 577-87.

14 - COFFMAN, M. Systems deadlocks, ACM Computing Surveys, New York, 3(2): 67-78, Jun. 1971.

15 - DATE, C. J. An Introduction to database systems, 3. ed., Reading, Mass., Addison-Wesley, 1981. (The Systems Programming Series)

16 - ———An Introduction to database systems. Reading, Mass., Addison-Westey, 1983. v. 2 (The Systems Programming Series).

17 - DAVENPORT, R. A. Desing of distributed database systems. The Computer Journal, London, Heyden & Son, 24(1): 31-41, Feb. 19 81

18 - DOWDY,W. L. & FOSTER,D. V. Comparative models of the file assignment problem. ACM Computing Surveys, New York, 14 (2): 287-313, Jun. 1982

19 - DRAFFAN, I.W. & POOLE,F. The Classification of distributed data base management systems. IN: —— Distributed data bases. Cambridge,University Press, 1980. chap. 3. p. 57-81.

20 - ESWARAN,K. P. et alii. The Notion of consistency and predicate locks in a database systems. ACM Communications,

New York, 9(11): 624-33, Nov. 1976

21 - GARCIA MOLINA,H. Reliability issues for fully replicated data bases, IEEE Computer Piscataway, N.J., 15(9): 34-42, Sept. 1982

22 - GARDARIM,G. et alii. Concurrency control principles in distributed and centralized data bases.Le Chesnay,INRIA/ Centre de Rocquencourt, jan. 1982. 91p. (Rapports de Recherche, 113).

23 - GHOSH, S. P. Distributing a data base with logical associations on a computer network for parallel searching. IEEE Transactions on Software Engineering. Piscataway, N.J. SE-2(2): 106-13, June 1976.

24 - GRAPA, Enrique & BELFORD, G. G. Some theorems to aid in solving the file allocation problem.ACM Communications, New York, NY, 20(11): 878-82, Nov. 1977

25 - HELD, G. D. et alii. INGRES - A relational data base management system.IN: AFIPS CONFERENCE PROCEEDINGS. Anaheim, Ca., May. 19-22, 1975. Spring join computer conference - Montvale NJ, AFIPS Press, 1975, v. 44, p. 409-16.

26 - HEVNER, A. et alii. Query processing in distributed data base system.IEEE Transactions on Software Engineering. Piscataway, N.J. SE-5 (3): 177-87, May 1979.

27 - HOLT, R. C. Some deadlock properties of computer systems. ACM Computing Surveys, New York, 4(3): 179-96, Set.1972

28 - KATZAN JR., H. An Introduction to distributed data processing. New York, Petrocelli Book, 1978. 242 p.

29 - KOHLER,W.N. A survey of techniques for sincronization in decentralized computer systems.ACM Computing Surveys, New York, 13(2): 149-83, Jun. 1976

30 - LAMPORT, L. Time, chocks and the ordering of events in a distributed system. ACM Communications,New York, 21(7): 558-65, Jul. 1978

31 - LAN,G. Consistency, sinchronization and concurrency control. IN: DRAFFAN, I. W. & POOLE, F. Distributed data bases. Cambridge, Cambridge University Press, 1980.Cap. 7: p. 195.220

32 - LINDSAY,B. Simple and multisite recovery facilities. IN: DRAFFAN, I.W. & POOLE,F. Distributed data bases Cambridge, Cambridge University Press, 1980. Cap. 10: p. 247-84.

33 - MAHMOUD,S. A. & RIORDON, J. S. Optimal allocations of resources in distributed information networks. ACM Transactions on Data Base Systems, New York, 1(1): 66-78, Mar. 1976

34 - MAHMOUD, S. A.et alii. Distributed data base partitroning and query processing. IN: BRACCHI, G. & NIJSSEN, G. M. Data base architecture. Amsterdam, North-Holland, 1979. p. 35-54.

35 - MARTELLA, G. & SCHREIBER, F. A. A Data dictionary for distributed databases. IN: DELOBEL, C. & LITWIN, W. Distributed data bases. Amsterdam, North-Holland, 1980. p. 17-33

36 - MENASCÉ, D. A. & NAKANISHI, T. Optimistic versus pessimistic concurrency control mechanisms in database management systems. Information Systems, Oxford, Eng., Pergamon Press, 7(1): 13-27, 1982.

37 - MORGAN, M. L. & LEVIN, K. D. Optmal program and data locations in computer networks. ACM Communications, New York, 20(5): 315-22, May. 1977

38 - OHANA, I. Arquivos e diretórios em banco de dados distribuidos. IN: CONGRESSO NACIONAL DE PROCESSAMENTO DE DADOS, 13. Rio de Janeiro, Out. 1980. Anais do XIII CNPD, Rio de Janeiro, SUCESU, 1980. p. 57-65

39 - PRICE, R. T. & OHANA, I. Alocação de arquivos em bancos de dados distribuídos. IN: CONGRESSO NACIONAL DE PROCESSAMENTO DE DADOS, 12. São Paulo, 8-12 Out. 1979. Anais do XII CNPD, São Paulo, SUCESU, 1979. p. 273-81.

40 - RAMAMOORTHY, C. V. & WAH, B. W. Data management in distri buted data bases. IN: AFIPS CONFERENCE PROCEEDINGS. New York, June 4-7, 1979. National Computer Conference, Mon tvale, N.J., AFIPS Press, 1979. v. 48. p. 667-80

41 - RIES, D. et alii. Effects of locking granularity in data base management system. ACM Transactions on Database Systems, New York, 2(3): 233-46, Sept. 1977.

42 - RIORDON,J. S. Resource allocation and query processing in distributed databases. IN: VAN DE RIET,R.P. & LITWIN, W. Distributed data sharing systems, Amsterdam, North-Holland Publishing Company, 1982, p. 96-9

43 - ROTHNIE,J. B. et alii. Introduction to a system for dis tributed databases (SDD-1). ACM Transactions on Data-base Systems, New York, 5(5): 1-17, March 1980

44 - SCHREIBER, F. A.et alii. Distributed data base applica tions: an overview. IN: DRAFFAN, I. W. & POOLE, F. Dis-tributed data bases. Cambridge, Cambridge University Press, 1980. Chap. 12. p. 323-36.

45 - STONEBRAKER, M. et alii. The design and implementation de INGRES. ACM Transactions on Database Systems, New York, 1(3): 189-222, Sept. 1976

46 - STONEBRAKER, M. Homogeneous distributed data base systems. IN: DRAFFAN, I. W. & POOLE, F. Distributed data bases.

Cambridge, Cambridge University Press, 1980. p. 103-154.

47 - SPACCAPIETRA, S. Heterogeneous data base distribution IN: DRAFFAN, I. W. & POOLE, F. Distributed data bases. Cambridge, Cambrigge University Press, 1980. Chap. 6 p. 155-93

48 - TANENBAUM, A. S. Computer Networks, Englewood Cliffs, N. J., Prentice - Hall, 1981

49 - TRAIGER, I. L. et alii. Transactions and consistency in distributed database systems. ACM Transactions on Databases Systema, 7(3): 323-42, Sept. 1982

50 - VETTER, M. & MEDDISON, R. N. Database design methodology. Englewood Cliffs, N. J., Prentice-Hall, 1981

51 - WONG, E. et alii. Decomposition - A strategy for query processing. ACM Transactions on Database Systems, New York, 1(3): 223-42, Sept. 1976