UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

Silvia das Dores Rissino

Controle de Concorrência em Bancos de Dados Distribuídos Heterogêneos

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Murilo Silva de Camargo

Porto Velho, fevereiro de 2001

Controle de Concorrência em Bancos de Dados Distribuídos Heterogêneos

Silvia das Dores Rissino

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Fernando A. Ostuni Gauthier

Banca Examinadora

Prof. Dr. Murilo Silva de Camargo (orientador)

Profª Drª Darlene Figueiredo Borges Coelho

Prof. Dr. Luiz Fernando Jacintho Maia

Prof. Dr. João Bosco da Mota Alves

As coisas não são como as vemos,
mas como as recordamos.

Oferecimento

Ao Mario

Agradecimentos

A Universidade Federal de Rondônia, pelo apoio incentivo.

A Universidade Federal de Santa Catarina.

Ao professor Murilo Silva de Camargo, pela orientação.

A minha família - Mario, meus pais (Ivete e Jorge) e meus irmãos (Mª Ivete, Jorge, Luiz e Ana), por tudo.

Aos amigos da Ordem Quinta.

Ao amigos do prédio da Reitoria, em especial aos funcionários do CPD Romualdo, Francileide e os estagiários (Bruno, Carlos, Cícero, Nicolau, Manoel e Rodrigo), por toda paciência e compreensão.

A Ivanda da Silva Pinto, pela amizade e apoio.

Aos meus alunos do curso de Informática da UNIR, que compreenderam, em alguns momentos, minha ausência.

A Rosane Rangel, pela amizade e apoio.

A professora Darlene Coelho, pela amizade e ajuda neste trabalho.

Ao Nildo Carlos por todo apoio neste curso.

Ao meus amigos deste curso de mestrado, em especial a Elisângela e Laura, que incentivaram e apoiaram nas horas difíceis.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

GLOSSÁRIO

BDD	Bancos de dados distribuídos
BDHH	Bancos de dados distribuídos heterogêneos
DB2	SGBD desenvolvido pela IBM
GTL	Gerenciado de transações Locais
GTG	Gerenciador de transações globais
LAN	Rede local de Computadores (Local Network Area)
MAN	Rede metropolitana de computadores
SGBD	Sistema Gerenciador de Banco de Dados
SGBDD	Sistema Gerenciador de Banco de Dados Distribuídos
SGBDFs	Sistemas Gerenciador de bancos de dados federados
SGBDDHs	Sistema gerenciador de bancos de dados distribuído heterogêneos
SGBDM	Sistema gerenciador de bancos de dados múltiplos
Scheduler	Escalonador
Troughput	Taxa de dados úteis realmente entregues ao computador destinatário
WAN	Redes de computadores geograficamente distribuídas

RESUMO

Controle de concorrência é um dos graves problemas em bancos de dados distribuído e heterogêneo. Os mecanismos empregados para resolvê-los, baseiam-se em abordagens pessimistas e otimistas. Cada uma desses abordagens, emprega mecanismos *wait, timestamp ordering* e/ou *rollback*. Em função desses mecanismos, os protocolos propostos serão livres ou não de *deadlocks* globais. Neste trabalho, são identificados os tipos de autonomia e heterogeneidade dos bancos de dados distribuídos e como estes influenciam no projeto dos mecanismos de controle de concorrência. Identifica-se, também, os principais mecanismos de controle de concorrência utilizados comercialmente, além de se fazer uma análise comparativa dos mecanismos apresentados.

Palavras ou expressões chaves: heterogêneo, autonomia, controle de concorrência distribuída, *deadlock*s, sistemas de bancos de dados, processamento de transações.

ABSTRACT

Concurrency control is one of the most serious problems in heterogeneous and distributed databases. The implemented mechanism to solve the problem are based on pessimistic and optimistic approaches. Each of those strategies use waiting mechanisms, time markers, and callback. Based on those mechanisms, the proposed protocols are free or not of global *deadlock*s. In this work the types of autonomies and heterogeneities of distributed databases are identified and the way they influence the design of concurrency control mechanisms are investigated. The principal mechanism of concurrency control used in commercial applications are identified and a comparative analysis of them is presented.

CAPÍTULO 1

INTRODUÇÃO

1.0 CONSIDERAÇÕES GERAIS

As organizações do mundo de hoje têm disponíveis uma grande variedade de bancos de dados confiáveis para dar suporte aos seu negócios. Grandes organizações usam bancos de dados em uma variedade de plataformas, incluindo *mainframes*, *workstations* e servidores configurados para trabalharem em ambientes de *Intranet* e com acessos à *Internet*.

As instituições vêm tornado-se muito sofisticadas no que se refere ao compartilhamento de informação. Ao mesmo tempo, os avanços da computação distribuída e das redes, combinadas com o alto grau de conectividade disponível, estão colaborando para compartilhamento e confiabilidade dessas informações.

A tecnologia de bancos de dados distribuído tem seu desenvolvimento recente no campo global de bancos de dados. Bancos de dados distribuídos são aqueles tipicamente não armazenados em apenas uma localização física, estando dispersos através de uma rede de computadores geograficamente afastados e sua conexão é feita através de elos de comunicação. Normalmente são gerenciados por diferentes sistemas gerenciadores de bancos de dados (SGBDs), que rodam em plataformas computacionais heterogêneas.

O desafio desses ambientes é disponibilizar aos usuários os dados de uma forma que para estes, o local de armazenamento dos dados manipulados independa do local de onde estejam acessando. Nesses sistemas é necessário não somente o acesso para vários usuários, mas também permitir que os bancos de dados envolvidos possam realizar acessos uniformes para todos os dados e sincronização da execução de suas transações.

Os usuários de ambientes computacionais heterogêneos devem ser capazes de interagir com múltiplos sistemas e dispositivos de hardware, além de poderem coordenar essas interações. Esses sistemas devem executar de forma autônoma, em diferentes plataformas de *hardwares*, devem suportar diferentes sistemas operacionais, devem ser projetados para diferentes propósitos e usar diferentes formatos de dados.

A sincronização e o acesso compartilhado nas bases distribuídas é uma das

metas que sempre deve ser alcançada em projetos de bancos de dados [Date. 1991]. Porém, em se tratando de bancos de dados distribuídos heterogêneos, a complexidade da implementação e do gerenciamento do bancos de dados global é muito alta, o que constitui um dos principais problemas a ser enfrentado pelos profissionais da área.

1.1 OBJETIVOS DO TRABALHO

1.1.1 OBJETIVO GERAL

Este trabalho fará uma estudo de controle de concorrência (CC) em bancos de dados distribuído heterogêneo (BDDH). Quais os tipos de autonomia e heterogeneidade existente nesse tipos de banco dados e quais os principais mecanismos empregados para solucionar o problema da concorrência.

1.1.2 OBJETIVOS ESPECÍFICOS

a) Identificar os tipos de autonomia e heterogeneidade dos bancos de dados distribuídos.
b) Identificar e comparar os mecanismos empregados para resolver os problemas decorrentes do controle de concorrência.

1.2 METODOLOGIA E ORGANIZAÇÃO DESTE TRABALHO

Em síntese, as principais atividades deste trabalho foram: levantar os problemas existentes com relação ao controle de concorrência de bancos de dados [Papadimitriou. 1982]; como esses problemas ocorrem em bancos de dados distribuídos heterogêneos; quais os mecanismos utilizados para resolver os problemas de controle de concorrência em bancos de dados distribuídos heterogêneos.

Para a realização deste trabalho se fez necessário entender, de forma geral, o gerenciamento dos bancos de dados distribuídos, em especial a gerência de transações e a gerência do controlador de concorrência distribuída, pois é neste tópico que se concentra o enfoque principal do trabalho [Elmasri e Navathe. 1998]. As tarefas

compreendidas neste trabalho e organização são mostrados a seguir.

1.2.1 Fases do Desenvolvimento do Trabalho

✓ **Primeira fase**: levantamento bibliográfico amplo, com objetivo de situar e delimitar o estudo do problema em questão. Nesta fase foram identificados e definidos os aspectos gerais de controle de concorrência em bancos de dados distribuídos heterogêneos.

✓ **Segunda fase**: Nesta fase é feita a exposição dos principais mecanismos de controle de concorrência de forma geral.

✓ **Terceira fase**: Nesta fase é feita a discussão dos mecanismos específicos, utilizados pelo controle de concorrência de bancos de dados distribuídos heterogêneos

1.2.2 Organização do Trabalho

O trabalho está dividido em seis capítulos como a seguir:

✓ Capítulo 1 – Introdução

✓ Capítulo 2 – Bancos de dados

Apresentam-se os conceitos básicos de bancos de dados, com ênfase nos sistemas de gerência de bancos de dados.

✓ Capítulo 3 – Bancos de dados distribuídos

Apresentam-se os conceitos básicos de distribuição de dados, compartilhamento, gerência distribuída de transações, taxonomia de bancos de dados distribuído.

✓ Capítulo 4 – Arquitetura de Bancos de dados distribuídos heterogêneos

Discussão sobre os tipos de bancos de dados heterogêneos com ênfase para os bancos de dados federados e os bancos de dados múltiplos.

✓ Capítulo 5 – Controle de Concorrência

O controle de concorrência é abordado de uma forma geral, discutindo-se a arquitetura de controle de concorrência, as principais anomalias das transações concorrentes, a taxonomia dos principais mecanismos de controle de concorrência e, para finalizar, a teoria de controle de concorrência.

✓ Capítulo 6 – Macanismos de Controle de Concorrência para BDDHs

Neste capítulo são discutidos os mecanismos específicos utilizados para controle de concorrência de BDDHs.

✓ Capítulo 7 – Conclusões e Recomendações para trabalhos futuros

✓ Capítulo 8 – Bibliografia

CAPÍTULO 2

BANCOS DE DADOS

2.0 INTRODUÇÃO

O uso de gerência em bancos de dados tem se tornado cada vez mais necessário, em função da complexidade em relação ao acesso e atualização dos dados das instituições. As instituições precisam se atualizar para se manterem no mercado competitivo; para tanto, é fundamental que elas utilizem formas de trabalho simples e modernizadas, para que atendam a todas as suas necessidades e de seus clientes de forma otimizada.

Para acompanhar o ritmo das demandas de trabalho e atualizações das instituições, foi necessário o surgimento de uma nova forma de compartilhamento de dados descentralizados, que gerou o surgimento dos Gerenciadores de Bancos de dados Distribuído. Essa ferramenta tornou possível a comunicação e troca de informação entre instituições, facilitando e agilizando o trabalho que, anteriormente, exigia muito tempo, dependendo, às vezes, da distância entre as instituições.

Este trabalho trata de Bancos de dados distribuídos heterogêneos no que diz respeito a controle de concorrência dos dados, que é um problema enfrentado por todos os tipos de SGBDs, já que implica em integridade, consistência e performance dos Bancos de Dados.

Neste capítulo serão apresentados os conceitos fundamentais de bancos de dados (seção 2.1), além dos aspectos gerais da gerência de sistemas de bancos de dados (seção 2.2), modelos de dados (seção 2.3), conceito geral de controle de concorrência (seção 2.4), além de uma definição formal para *deadlock* (seção 2.5) e processo (seção 2.6).

2.1 BANCOS DE DADOS

Bancos de dados é uma coleção de dados operacionais, ou uma coleção de dados lógicos com algum significado ou relação entre os mesmos, sendo este um conjunto de dados integrados e organizados, constituindo assim uma representação natural de dados,

sem impor restrições ou modificações para que sejam adaptados ao computador [Date 1991].

2.2 Sistema de Gerência de Bancos de dados (SGBDs)

Os SGBDs são um conjunto de programas que possibilita auxílio aos usuários na manipulação do bancos de dados, sendo estes voltados para quaisquer aplicações. Os SGBDs foram criados para manipular grandes quantidades de dados e possuem as seguintes características: independência, compartilhamento, controle de redundância, integridade, privacidade, segurança, consistência, relacionamento e controle de espaço de armazenamento de dados.

1) Independência dos Dados

É o mais importante dos objetivos de um bancos de dados, pois permitirá a expansão das atividades da empresa, isto é, criar nova estrutura lógica decorrente de uma nova aplicação ou inclusão de um dado na estrutura existente.

Considerando a lógica dos dados, há três níveis de abstração de dados, que conjuntamente são chamados de arquitetura em três níveis (ANSI/PARC) [Date 1991], que são mostrados a seguir:

a) Nível Físico ou Interno

Este nível descreve o armazenamento físico dos dados, envolvendo estruturas de dados complexas utilizadas para sua representação.

b) Nível Conceitual

Este nível descreve os dados que estão armazenados no bancos de dados, e alguns aspectos relacionados à gerência desses dados como, por exemplo, as relações existentes entre os dados, questões de segurança de acesso.

c) Nível Externo ou Visão

Este nível descreve apenas o conjunto de visões de usuário ou esquemas externos. Ou seja, como os usuários não precisam enxergar todo conteúdo do bancos de

dados, então são definidas visões do mesmo, para que apenas tenham acesso ao que realmente necessitam.

Existem dois níveis de independência de dados; são eles:

Primeiro Nível - Independência Física dos Dados: É a facilidade de modificar o nível físico sem a necessidade de modificar os níveis conceituais e externos, nem os programas de aplicação que acessam os dados.

Segundo Nível - Independência Lógica dos Dados: É a facilidade de se modificar o nível conceitual sem a necessidade de reescrever os programas de aplicação, ou de realizar qualquer alteração no nível externo.

2) Compartilhamento dos Dados

Em muitos casos os dados devem ser armazenados em um bancos de dados de forma que possam ser compartilhados por toda a corporação. Dessa forma, evita-se que os dados sejam replicados, ou seja, que para cada usuário que deseje manipular um dado em um determinado instante exista uma cópia.

Os acessos compartilhados devem ser feitos de tal forma que não haja perda de integridade nem da consistência dos mesmos. Existem regras para o compartilhamento dos dados, ou seja, quando vários usuários estiverem acessando apenas para leitura, todos podem compartilhar o dado, mas se existir algum usuário tentando acessar para gravação, então o dado deve ser acessado de forma exclusiva, para que os outros usuários, mesmo acessando para leitura, não vejam valores diferentes para um mesmo dado.

3) Controle de Redundância de Dados

É uma maneira centralizada de compartilhamento de dados por várias aplicações, impedindo dessa forma a repetição dos mesmos, já que esta pode implicar em inconsistência do bancos de dados. A duplicidade de dados só é permitida em alguns casos para melhorar a performance.

4) Garantir a Integridade dos Dados

É a necessidade de manter dados de maneira íntegra, quando da concorrência do mesmo por dois ou mais usuários.

Quando usuários concorrentes tentam realizar uma atualização sobre o mesmo dado, pode ocorrer inconsistência na base de dados; para resolver esse problema o sistema deve ser capaz de evitar o acesso ao dado por um usuário quando o mesmo já estiver sendo acessado para alterações.

5) Privacidade de Dados

Deve haver controles para que o acesso aos dados seja feito somente por usuários autorizados. Esta preocupação é redobrada quando o bancos de dados utiliza conexão remota, pois dados confidenciais podem fluir através da rede para usuários não autorizados.

6) Segurança dos Dados

Segurança envolve procedimentos de validação, garantia de integridade e controle de acesso, que tem como principal objetivo resguardar o bancos de dados de possíveis perdas ou destruição de dados por falhas de programa, por falha no sistema ou equipamento.

7) Consistência dos Dados

No caso de duplicidade dos dados, deve-se garantir que as diversas cópias contenham os mesmos valores, ou seja, que no caso de atualização de uma cópia, a outra ou outras sejam atualizadas automaticamente, garantindo dessa forma a consistência dos dados.

8) Relacionamento entre Dados

O relacionamento entre dados existentes nos diferentes arquivos deve ser controlado de forma automática, isto é, o programador não deve se preocupar com a criação de procedimentos para controlar os relacionamentos.

9) Controle de Espaço de Armazenamento

O SGBD deve possuir técnicas ou mecanismos de controle de acesso do espaço

alocado, do espaço real utilizado e a disponibilidade do espaço do gerenciador do bancos de dados. Deve também possuir técnicas de compressão de dados e reaproveitamento automático dos espaços liberados pela atualização dos dados.

2.3 Modelo de Dados

2.3.1 Modelos Conceituais ou de Alto Nível

Estes modelos apresentam conceitos mais próximos à percepção do usuário, e utilizam conceitos como:

- **Entidade:** É um objeto do mundo real, que é representado dentro de um bancos de dados.

- **Atributo:** É uma propriedade pela qual são descritas as características do objeto.

- **Relacionamento:** É a interação ou associação entre vários objetos.

2.3.2 Modelos de Implementação

São modelos utilizados pelos usuários para facilitar a organização lógica das bases de dados. A seguir serão apresentados os modelos mais usados.

✓ Modelo em Rede ou Reticular

É um modelo formal para representação do relacionamento de atributos de um conjunto de entidades e associações entre conjuntos de entidades. Os gerenciadores mais comuns baseados neste modelo são:

- IDMS
- IDMS/R
- DMS 1100
- TOTAL
- IMF

O modelo em rede consiste num conjunto de tipos de registros e ligações entre esses tipos de registros. Suas principais características são:

- Formar listas ligadas;
- Não incorporar informações semânticas.

Este modelo pode ser considerado o primeiro Gerenciador de Bancos de dados que foi comercializado em larga escala no mercado.

✓ **Modelo Hierárquico**

Um bancos de dados hierárquico é uma coleção de árvores desmembradas com ocorrência de registros como nós. Cada árvore desmembrada é chamada de árvore de bancos de dados e consiste em um registro raiz e todos os seus registros dependentes de vários níveis.

Em um bancos de dados hierárquico identifica-se a relação de pai e filhos entre registros de acordo com as conexões entre os registros, além do que pode-se identificar relações de descendência e ascendência entre os registros. Um caminho hierárquico é uma seqüência de registro, iniciada pelo registro raiz, na qual os registros estão numa relação alternada pai e filho. Exemplos de sistemas gerenciaodres de bancos de dados hierárquicos: *IMS* da *IBM* e *o SYSTEM* 2000 da *Honeywell.*

✓ **Modelo Relacional**

O modelo realcional representa as inter-relações entre atributos de um conjunto de entidades e associações entre os conjuntos de entidades. A maioria dos produtos comercializados de gerenciadores de bancos de dados utiliza o modelo relacional com padrão. Como exemplo temos: *DB2, INTERBASE* entre outros.

✓ **Modelo Orientado a Objetos**

A orientação a objetos corresponde à organização de sistemas como uma coleção de objetos que integram estruturas de dados e comportamento. Além desta noção básica, a orientação a objetos inclui um certo número de conceitos, princípios e mecanismos que a diferenciam das demais, sendo estes: Abstração, Objeto, Identidade de Objeto, Objetos Complexos, etc.

Os modelos de dados orientados a objetos têm um papel importante nos SGBDs porque, em primeiro lugar, são mais adequados para o tratamento de objetos complexos (textos, gráficos, imagens) e dinâmicos (programas, simulações). Depois, por possuírem maior naturalidade conceitual e, finalmente, por estarem em consonância com fortes tendências em linguagens de programação e engenharia de software. O casamento entre

as linguagens de programação e bancos de dados é um dos problemas que estão sendo tratados de forma mais adequada no contexto de orientação a objetos.

2. 4 CONTROLE DE CONCORRÊNCIA

Controle de concorrência é a capacidade que o sistema possui, em um ambiente multi-usuário, de permitir que várias transações acessem simultaneamente dados em uma base de dados.

Grau de concorrência é definido como sendo o volume de transações executadas pelo sistema durante um determinado intervalo de tempo. Por outro lado, a consistência integral dos dados, em um ambiente multi-usuário, implica que as alterações efetuadas por uma transação não devem afetar as outras transações em curso.

Controle de concorrência em sistema de bancos de dados distribuído é uma atividade responsável pela coordenação dos acessos concorrentes de diferentes usuários ou processos [Silberschatz e Galvin. 1998].

2.5 DEADLOCK

Deadlock é um estado em que um sistema computacional pode estar, desde que exista um conjunto de transações tal que toda transação no conjunto esteja esperando por uma transação ou por um recurso já alocado a outra transação, impossibilitando o processamento normal do sistema [Silberschatz e Galvin. 1998]; enquanto não houver liberação dos recursos necessários, as transações ficam impossibilitadas de continuarem suas execuções, havendo a paralisação do sistema. Segundo Tanenbaum [1995] existe quatro condições para ocorrer *deadlock*:

1. Condição de exclusão mútua: cada recurso ou está alocado a exatamente um processo ou está disponível.
2. Condição de posse e de espera: processos que estejam de posse de recursos obtidos anteriormente podem solicitar novos recursos.
3. Condição de não-preempção: recursos já alocados a processos podem ser tomados à "força". Eles precisam ser liberados explicitamente pelo processo que detém a posse.

4. Condição de espera circular: deve existir uma cadeia circular de dois ou mais processos, cada um dos quais esperando por um recurso que está com o próximo membro da cadeia.

As quatro condições acima devem estar presentes para ocorrer o *deadlock*, se uma delas estiver ausente, não há a menor possibilidade de ocorrer uma situação de *deadlock*.

2.6 PROCESSO

Um processo pode ser definido como um programa em execução, incluindo os valores correntes de todos os registradores do hardware e das variáveis, manipuladas pelo próprio processo, no curso da execução [Tanenbaum. 1995].

2.7 CONSIDERAÇÕES FINAIS

Foram apresentadas neste capítulo os conceitos básicos de bancos de dados, os principais tipos de modelos, conceito de controle de concorrência, *deadlock* e processo. Tais conceitos darão suporte à discussão, no próximo capítulo, sobre bancos de dados distribuído.

CAPÍTULO 3

BANCOS DE DADOS DISTRIBUÍDOS

3.0 INTRODUÇÃO

Um sistema distribuído é caracterizado por possuir diversos processadores e outros recursos que podem ser compartilhados por todos os usuários do sistema, sem ser necessário que estes saibam de onde estão usando tal recurso. A mesma idéia é usada para Bancos de Dados Distribuídos (BDDs), onde há dados espalhados por todos os nós da rede e os usuários podem acessar esses dados sem problemas e sem saber qual a localização dos dados acessados [Bhargava. 1999]

Em Bancos de Dados Distribuídos o acesso aos dados pelos usuários é feito da mesma forma que no sistema centralizado. Para o usuário, a forma como os dados são acessados é transparente, mas para o SGBD a forma de acesso e o controle aos dados distribuídos é muito mais complexa.

O Sistema de Bancos de dados Distribuído envolve vários locais com ambientes computacionais diferentes, cada qual com um ou mais processadores que estão conectados via rede de comunicação, na qual cada nó possui seu próprio SGBD local para gerência dos dados locais. Os usuários, em qualquer um desses nós, poderão acessar qualquer dado na rede, como se este dado estivesse armazenado no local de acesso.

Bancos de dados distribuído é um conjunto de dados logicamente integrados e compartilhados que se encontram fisicamente armazenados em diferentes computadores (nós) de uma rede, podendo esta rede ser local (LAN), metropolitana (MAN) ou redes geograficamente distribuídas (WAN) [Elmasri e Navathe. 1998].

A utilização de BDD ocorre para diminuir a sobrecarga em BDs centralizados, tempo de resposta mais rápido nas consultas, maior disponibilidade (tolerância a falhas), mais flexibilidade (diferentes SGBDs, diferentes modelos, etc.), além das redes de telecomunicações, atualmente, permitirem uma melhor distribuição e acesso aos dados. Mas há algumas dificuldades enfrentadas pelos BDDs:

✓ a malha de interconexão entre os nós do bancos de dados torna-se, muitas vezes, um problema para *throughput* do sistema;

✓ grande variedade de ambientes operacionais (transformação dos dados);

✓ manter a consistência do bancos de dados;

✓ alocação dos dados;

✓ balanceamento da carga de uso entre os nós; etc.

3.1 Sistema de Gerência de Bancos de Dados Distribuídos

Arquitetura de um SGBDD

A arquitetura básica de um SGBDD é constituída pelo Diretório de Dados Global (**DDG**), Gerente de Transações (**GT**) e o Gerente de Dados (**GD**).

DDG contém a descrição do bancos de dados distribuído;

GT é responsável pelo controle de acesso aos dados do BDD;

GD é a interface com o SGBD local, tendo como principal função as traduções entre bancos de dados heterogêneos.

Em cada nó da rede de comunicação será instalado um SGBD local em conjunto com uma cópia do SGBD global. Cada nó poderá ou não conter parte do diretório global dos dados, mas isto dependerá da estratégia de alocação de dados adotada. Na figura 3.1 é mostrado um esquema de uma bancos de dados distribuído, onde há vários bancos de dados locais interconectados através de uma rede de computadores.

a) Gerência de Transações Distribuídas

Uma transação identifica uma unidade elementar de trabalho que faz parte de uma aplicação, na qual são definidas características de confiabilidade e isolamento.

Transações consistem em uma seqüência de operações (na base de dados) que possui a característica de ser atômica em relação ao controle da concorrência e da recuperação. Desta forma, existem apenas dois estados: todas as operações presentes numa transação são realizadas como se fossem uma ou nenhuma delas é realizada.

Uma transação pode ser definida sintaticamente, independete da linguagem na qual é escrita, como estando dentro de dois comandos: *Begin_transaction* e *End_transaction*. Dentro dos comandos de uma transação há duas instruções que, em particular, podem aparecer: *Commit* e o *Rollback*.

A transação será finalizada com sucesso somente após o comando *Commit*, enquanto que um efeito contrário será mostrado no bancos de dados com a instrução de

Rollback, que é utilizada para que o bancos de dados retorne a um estado de consistência; esta instrução é muito poderosa, pois, através dela, o usuário dos bancos de dados pode cancelar os efeitos de uma transação, independentemente da sua complexidade.

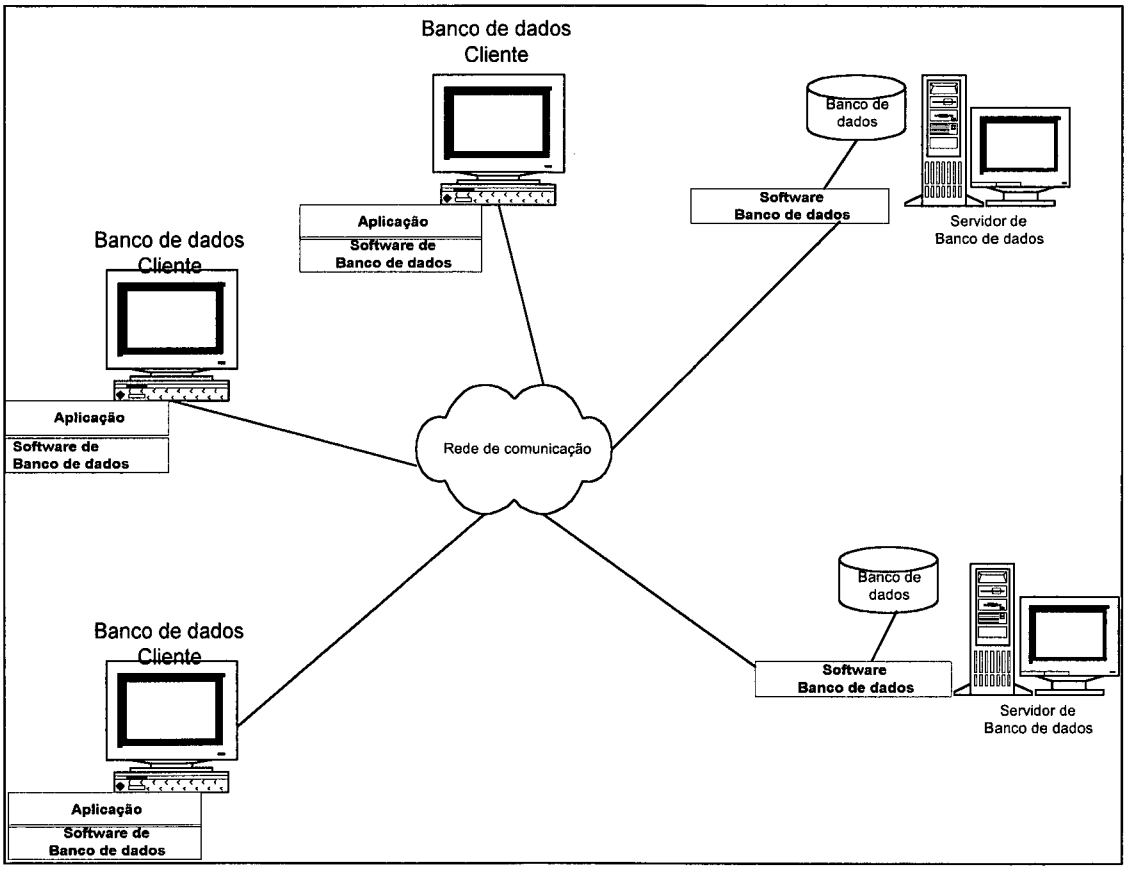

Figura 3.1: Esquema de uma rede com um BDD

PROPRIEDADES DAS TRANSAÇÕES

As transações possuem propriedades particulares tais como: Atomicidade, consistência, isolamento e durabilidade; essas propriedades apresentadas a seguir, são referenciadas como ACID das transações [Atzeni et al. 2000]:

✓Atomicidade

Como uma transação é uma unidade indivisível de execução, a atomicidade de uma transação garantirá que uma unidade do programa deve ser executada automaticamente, sem ser subdividida [Silberschatz e Galvin. 1998]. Na prática, não é possível haver alteração parcial no bancos de dados, isto é, ou a transação é executada

até o final, ou nenhuma das alterações provocadas por parte dos procedimentos desta transação serão consideradas.

A atomicidade tem conseqüências significantes em nível operacional: se durante a execução de uma operação ocorrer um erro, as operações da transação não poderão ser finalizadas, isto é, deve-se desfazer o trabalho realizado pelas primeiras operações da transação.

Para ocorrer a efetivação de uma transação em um bancos de dados, antes da execução do comando *commit*, o sistema deve se assegurar de que as transações levarão o bancos de dados a um estado final consistente. Sendo assim, o comando *commit* apenas deverá ser executado se a transação for finalizada com sucesso. Entretanto, se ocorrer alguma falha, deverá se provocar a eliminação de todos os efeitos da transação e o estado original deverá ser reconstituído (*rollback*).

✓Consistência

A consistência exige, na execução de uma transação ou de transações concorrentes, que nenhuma regra de integridade, definida para os respectivos bancos de dados, seja violada [Gardarin e Valduriez. 1989]. Quando uma transação viola uma regra de integridade, o sistema deve intervir para cancelar a transação ou para corrigir a transação no que se refere a violação da regra.

✓Isolamento

O isolamento demanda que a execução de uma transação seja independente da execução simultânea de outras transações [Bernstein. 1987].

A meta do isolamento é fazer com que o resultado de cada transação seja independente do resultado das outras transações. Isto é, se houver problema na execução de alguma transação, o sistema terá a capacidade de realizar um *rollback* em função de uma transação em particular, sem causar o mesmo efeito em outras transações.

✓Durabilidade

A durabilidade demanda que o efeito de uma transação que foi corretamente efetivada, execute o comando *commit* para que a alteração no bancos de dados se torne permanente.

De modo geral, a Gerência de Transações tem como objetivo a eficiência, segurança e execução concorrente de transações, sendo meta em bancos de dados distribuído o controle da execução de transações, ou seja, a gerência de transações deve garantir:

1. Atomicidade, durabilidade, seriabilidade e propriedades de isolamento;
2. Custos menores em se tratando de memória principal, CPU e número de mensagens de controle transmitida e seus tipos de respostas;
3. Maximização da disponibilidade dos dados.

ESTADOS DAS TRANSAÇÕES

A transação é uma unidade de trabalho atômica [Elmasri e Navathe. 1998], isto é, ou é finalizada com êxito ou é desconsiderado qualquer resultado de uma operação que faz parte da transação, pois para situações em que há pretensões de recuperação, o sistema necessita saber o local exato em que a transação iniciou, terminou, realizou um *commit* ou um *abort*; para isso, o sistema utiliza as seguintes operações [Atzeni et al. 2000]:

✓*Begin_trasaction*: marca o início da execução da transação.

✓*Read ou Write*: Especificam as operações de leitura e escrita em um item do banco de dados em que é executada parte da transação.

✓*End_transaction:* Especifica que as operações de leitura e escrita foram finalizadas, marcando desta forma o limite da execução. Contudo, este ponto é necessário para checar as trocas introduzidas pela transação, se as mesmas serão aplicadas ao bancos de dados (*commited*) ou se, ao contrário, serão rejeitadas (*abort*), pois violam o controle de concorrência.

✓*Commit_transaction:* Esta operação assinala um final com sucesso para a transação. Isto é, as atualizações realizadas pela transação foram efetivadas no bancos de dados, sem correr o risco de serem desfeitas.

✓*Rollback (or Abort):* Assinala que a transação foi finalizada sem sucesso. Isto

é, as alterações ou os efeitos das transações, que eram para ter sido efetivos foram desfeitos ou encerrados antes da finalização.

Para os procedimentos de recuperação do bancos de dados, algumas técnicas utilizam-se de operações como:

✓*Undo*: Similar ao *rollback*, desfaz operações.

✓*Redo*: Especifica que certas operações da transação devem ser refeitas para garantir que todas as operações de *commiteted* da transação serão aplicadas com sucesso no bancos de dados.

A seguir, tem-se a figura 3.2, onde é exibido um diagrama de transição de estados de uma transação. Neste esquema, há uma transação que se move através dos seus estados durante a sua execução. A transação entra no estado ativo, imediatamente antes de começar a execução, quando pode ocorrer uma operação de leitura ou de escrita.

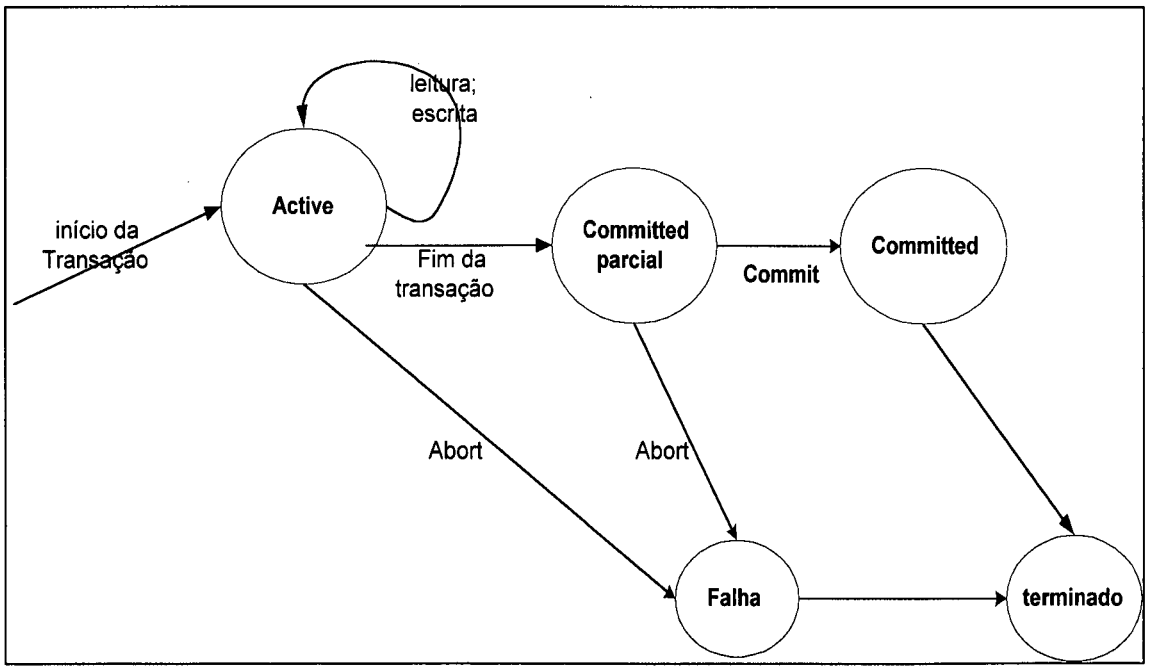

Figura 3.2 – Diagrama de transição de estados para execução de uma transação

Quando a transação finaliza, move-se para um estado de comprometimento parcial (*committed* parcial). Neste ponto, algumas técnicas de controle de concorrência

requerem que sejam feitas checagens para assegurar que a transação não interfira com outras transações que estão sendo executadas. Além disso, alguns protocolos de recuperação precisam assegurar que uma falha no sistema não resultará na incapacidade de registrar as mudanças de uma transação permanentemente (usualmente registrando mudanças no log do sistema). Uma vez feitas ambas as checagens, e as mesmas são bem sucedidos, se diz que a transação alcançou o ponto comprometimento e entra no estado de comprometimento (*commited*), e completa sua execução com sucesso.

Porém, uma transação pode entrar em um estado de falha se uma das checagens falhar, ou se é abortada durante seu estado de atividade. A transação pode então voltar para desfazer os efeitos de suas operações de escrita no bancos de dados. Um estado terminado corresponde a uma transação deixando o sistema. Transações que falharam ou foram abortadas podem serem reiniciadas mais tarde, tanto automaticamente como depois de serem novamente submetidas, como se fossem uma nova transação.

TRANSAÇÕES ATÔMICAS CONCORRENTES

Como cada transação necessariamente deve ser atômica, a execução concorrente de transações deve ser equivalente a que cada transação execute serialmente em alguma ordem arbitrária. Esta propriedade é conhecida como seriabilidade, podendo ser mantida através da execução de cada transação em uma seção crítica [Silberschatz e Galvin. 1998].

Recuperação de Transações Distribuída

Para que haja atomicidade na execução de uma transação distribuída, são necessárias duas condições [Elmasri e Navathe. 1998]:

1) Em cada nó, da rede de comunicação, todas as ações são efetivadas ou não;
2) Cada nó, da rede de comunicação, tem que ter a mesma decisão com respeito à confirmação ou cancelamento de sua subtransação.

O relacionamento entre a gerência de transação distribuída e a gerência de transações locais está representado no modelo de transações distribuída da figura 3.3.

O Gerente de Transações Locais (GTL) implementa a interface 1: *Local_begin, local_commit e local_abort.* O GTL pode também criar processos agentes (local_create).

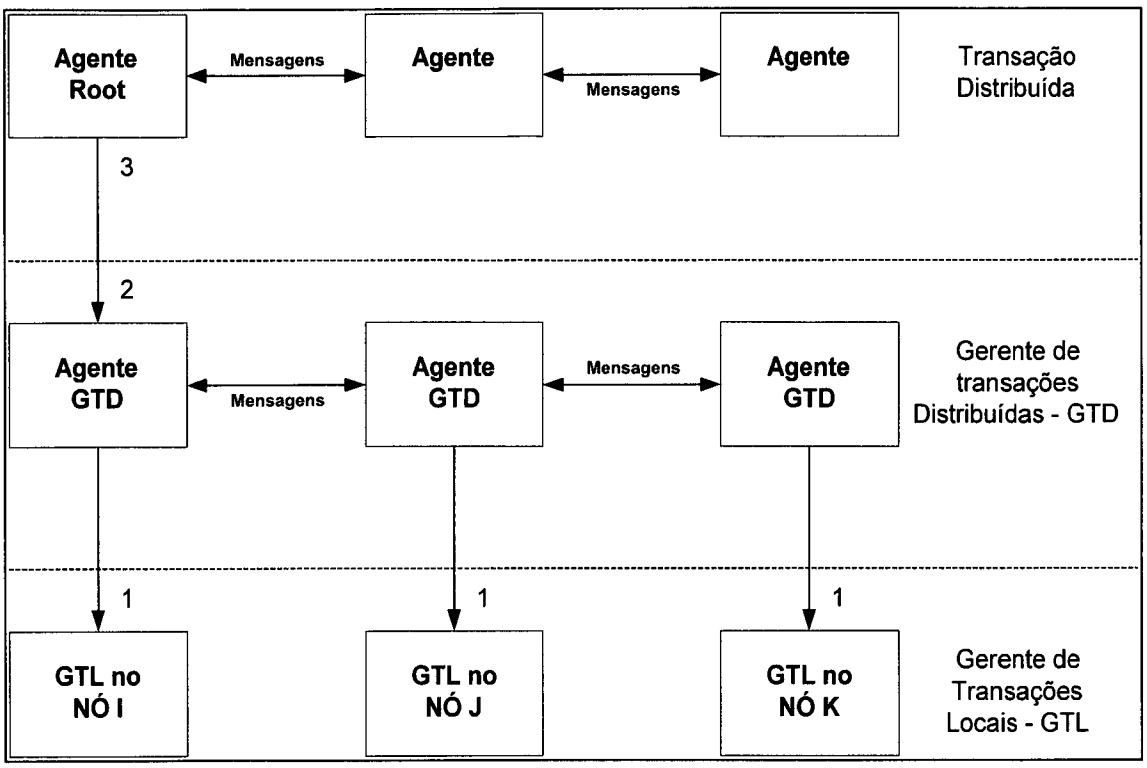

Figura 3.3. Modelo Referencial de Recuperação de transação distribuída.

Interface 1: *Local_begin, local_commit, local_abort, local_create*

Interface 2: *Begin_transaction, commit, abort, create*

O Gerente de Transações Distribuída (GTD) é uma camada distribuída, implementada por um conjunto de agentes GTD locais trocando mensagens entre si. O GTD implementa a interface 2: *begin_transaction, commit, abort* e *create agent* (remoto).

No nível 3 temos as transações distribuídas, constituídas pelo agente *root* e outros agentes. Apenas o agente *root* pode usar as primitivas *begin_transaction*, *commit* e *abort*. A interface 2 é usada apenas pelo agente root.

A interface 2 é implementada pelo gerente de transações do seguinte modo:

Begin_transaction: Quando o comando *begin_transaction* é usado pelo agente root, a GTD usará a primitiva *local_begin* para a GTL do nó de origem e para todos os nós nos quais já existem agentes ativos para a mesma aplicação, transformando todos os

agentes em subtransações.

Abort: Quando o comando *Abort* é usado pelo agente *root*, todas as subtransações devem ser canceladas e recuperadas. Isto é feito usando-se o comando *local_abort* para os nós onde existe uma subtransação ativa.

Commit: Quando o comando *commit* é usado pelo agente root, todas as subtransações devem ser confirmadas. Caso alguma subtransação fique impossibilitada de confirmar suas alterações, esta e todas as outras subtransações participantes devem recuperar as alterações realizadas no banco.

b) GERÊNCIA DE CONTROLE DE CONCORRÊNCIA

O módulo responsável pelo controle de concorrência em um SGBDD é conhecido como escalonador (*scheduler*). Sua principal função é evitar interferências entre as transações que são executadas concorrentemente.

A comunicação entre o programa de aplicação e o escalonador é realizada pelo Gerente de Transações (coordena operações dos bancos de dados na execução de aplicações). Qualquer técnica de controle de concorrência adotada pelo sistema deve garantir que toda execução concorrente de um conjunto de transações seja serializável. Isto é, em um conjunto de transações, cada transação deve ser executada completamente antes que a próxima transação, envolvendo o mesmo grupo de dados, inicie. Há basicamente três técnicas de controle de concorrência o bloqueio, *timesatamp* e o método otimista [Elmasri e Navathe. 1998]. A seguir, é mostrado um exemplo de transações que são submetidas ao SGBD e como podem ser executadas. Em conjunto, são apresentados os conceitos de escalonamentos seriais e não seriais; no capítulo 5 será feito detalhamento da teoria de serialização.

Exemplo

Supondo que dois usuários submetem ao SGBD as transações T_1 e T_2 aproximadamente ao mesmo tempo (mas não é permitido a mesclagem das transações), como mostra a tabela 3.1, há duas possíveis maneiras de ordenar as operações das transações para execução:

1. Executar todas as operações da transação T_1 em seqüência, acompanhada por todas as operações da transação T_2 também em seqüência.

2. Executar todas as operações da transação T_2 em seqüência, acompanhada por todas as operações da transação T_1 também em seqüência.

Se o intercalamento das operações é permitido, haverá muitas possíveis ordens na qual o sistema pode executar operações individuais da transação.

Transação T_1	**Transação T_2**
Read_item(X);	Read_item(X);
X:= X – N;	X:= X + M;
Write_item (X);	Write_item (X);
Read_item (Y);	
Y := Y + N	
Write_item (Y)	

Tabela 3.1. Duas transações simples. Transação T_1 e T_2.

Nas tabelas 3.2a e 3.2b são mostradas as possibilidades para escalonamento das transações T_1 e T_2 da tabela 3.1, os escalonamentos A e B da tabela 3.2 são chamados **escalonamentos seriais**, pois as operações são executadas consecutivamente, sem nenhum intercalamento de operações de outra transação.

ESCALONAMENTO A

Transação T_1	**Transação T_2**
Read_item (X);	
X:= X- N;	
Write_item (X);	
Read_item (X);	
Y = Y + N;	
Write_item (Y);	
	Read_item (X);
	X:= X + M;
	Write_Item (X);

Tabela 3.2a. Escalonamentos Seriais no qual T_1 é seguida de T_2.

ESCALONAMENTO B

TRANSAÇÃO T_1	TRANSAÇÃO T_2
	Read_item (X);
	X:= X + M;
	Write_Item (X);
Read_item (X);	
X:= X- N;	
Write_item (X);	
Read_item (X);	
Y = Y + N;	
Write_item (Y);	

Tabela 3.2b. Escalonamentos Seriais no qual T_1 é seguida de T_2.

Em um escalonamento serial, transações conjuntas são realizadas em ordem serial: T_1 e então T_2 , no Escalonamento A, e T_2 seguido de T_1 no Escolanamento B. Os escalonamentos seriais consistem de uma seqüência de instruções de várias transações, na qual as instruções pertencentes a uma única transação aparecem juntas naquele escalonamento. Assim, para um conjunto de n transações existe n! diferentes escalonamentos seriais válidos [Korth e Silberschatz. 1998]. Quando diversas transações são executadas concorrentemente, o escalonamento correspondente não precisa ser serial, isto é, o **escalonamento é do tipo serializável**. Logo, o número de escalonamentos possíveis para o conjunto de n transações é bem maior que n!. Se examinarmos os escalonamentos das Tabelas 3.3a e 3.3b , onde duas transações são executadas concorrentemente, há possibilidade de haver diversas seqüências de execuções, uma vez que várias instruções das duas transações podem ser intercaladas.

ESCALONAMENTO A

TRANSAÇÃO T_1	TRANSAÇÃO T_2
Read_item (X);	
X:= X- N;	
	Read_item (X);
	X:= X + M;
Write_item (X);	
Read_item (Y);	Write_Item (X);
Y = Y + N;	
Write_item (Y);	

Tabela 3.3a . Escalonamento Concorrente Serializável

Escalonamento B

Transação T_1	Transação T_2
Read_item (X);	
X:= X- N;	
Write_item (X);	
	Read_item (X);
	X:= X + M;
	Write_Item (X);
Read_item (Y);	
Y = Y + N;	
Write_item (Y);	

Tabela 3.3b. Escalonamento Concorrente Serializável

Nem todas as execuções concorrentes resultam em um estado correto. Na tabela 3.4 pode-se observar um exemplo de escalonamento não serial. Considere a variável A com um valor inicial de 1000 e a variável B com um valor inicial de 2000.

Após a execução do escalonamento exibido na tabela 3.4, tem-se um estado cujos os valores finais das varáveis são: A = 950 e B = 2.100. Esse estado final é um estado inconsistente, já que, se acresceu 50 no processo de execução concorrente. A soma (A+B) não é preservada pela execução das duas transações T_1 e T_2.

Transação T_1	Transação T_2
Read(A)	
A:= A – 50	
	Read(A)
	Temp:= A * 0.1
	A:= A – temp
	Write(A)
	Read(B)
Write(A)	
Read(B)	
B:= B + 50	
Write(B)	
	B:= B + temp
	Write(B)

Tabela 3.4. Escalonamento Não Serializável

Há necessidade de que uma transação seja um programa que preserve a consistência, isto é, cada transação, quando executada sozinha, transfira o sistema de um

estado consistente para um novo estado consistente.

Durante a execução de uma transação, no entanto, o sistema pode temporariamente entrar num estado inconsistente. Uma inconsistência temporária cria a possibilidade de inconsistência em escalonamentos não seriais, o que pode acarretar em inconsistência no bancos de dados. Mas, após a execução de transações, um escalonamento deve deixar o bancos de dados em um estado consistente.

Escalonamento de Conflito Serializável

Suponha-se que há um escalonamento S no qual existem duas instruções consecutivas I_i e I_j, das transações T_i e T_j respectivamente. Se I_i e I_j referem-se a itens de dados diferentes, então podemos trocar I_i e I_j sem afetar os resultados de qualquer instrução no escalonamento. Entretanto, se I_i e I_j referem-se ao mesmo item de dado X, então a ordem desses passos pode importar, já que se trata de instruções de leitura (*read*) e escrita (*write*). Há quatro situações que precisam ser consideradas.

1. Se I_i = *Read*(X), I_j = *Read*(X). A ordem de I_i e I_j não importa uma vez que o mesmo valor de X é lido por I_i e I_j independente da ordem.

2. Se I_i = *Read*(X), I_j = *Write*(X). Se I_i vem antes de I_j , então T_i não lê o valor de X que é gravado por T_i na instrução I_j . Se I_j vem antes I_i , então T_i lê o valor de X que é gravado por T_j . Assim, a ordem de I_i e I_J importa.

3. Se I_i = *Write*(X), I_j = *Read*(X). A ordem de I_i e I_j importa por razões similares a anterior.

4. Se I_i = *Write*(X), I_j = *Write*(X). Uma vez ambas instruções são operações *Write*, a ordem dessas instruções não afeta T_i e nem T_j . Porém, o valor obtido pela próxima instrução *Read* (X) de S é afetado, já que somente o resultado das duas últimas instruções *Write* é preservado no bancos de dados. Se não existir nenhuma outra instrução Write(X) depois de I_i e I_j em S, então a ordem de I_i e I_j afeta diretamente o valor final de X no estado do bancos de dados que resulta no escalonamento X.

Logo, somente no caso em que I_i e I_j são instruções *Read*, a ordem relativa de

suas execuções não interessa. Por isso I_i e I_j se conflitam, caso elas sejam operações de transações diferentes no mesmo item de dado, onde pelo menos uma das instruções seja uma operação *Write*. Para mostrar o conceito de operações conflitantes, considere o escalonamento da tabela 3.5.

A instrução *Write*(A) de T_i conflita com a instrução *Write*(A) de T_j . Porém, a instrução *Read*(A) de T_j não conflita com a instrução *Read*(B) de T_i , pois as duas acessam itens de dados diferentes.

Se I_i e I_j forem instruções consecutivas de um escalonamento S, e se I_i é diferente de I_j e não são conflitantes, a troca da ordem de I_i e I_j produzirá um escalonamento S'.

Espera-se que S seja igual a S', já que todas as instruções aparecem na mesma ordem nos dois escalonamentos, exceto por I_i e I_j cuja ordem não importa.

T_i	T_j
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Tabela 3.5. Instruções conflitantes

Quando a instrução *Write*(A) de T_i no escalonamento da tabela 3.5 não conflita com a instrução *Read*(B) de T_j, pode-se trocar as instruções, o que terá como conseqüência um escalonamento equivalente, que é exibido na tabela 3.6. Independentemente do estado inicial do sistema, os escalonamentos das tabelas 3.8 e 3.9 produzem o mesmo estado final.

T_i	T_j
Read(A)	
Write(A)	
	Read(A)
Read(B)	
	Write(A)
Write(B)	
	Read(B)
	Write(B)

Tabela 3.6. Escalonamento depois da troca de instruções

Trocando-se de lugar as instruções não conflitantes, teremos:

1. Instrução *Read*(B) de T_i com a instrução *Read*(A) de T_j.
2. Instrução *Write*(B) de T_i com a instrução *Write*(A) de T_j.
3. Instrução *Write*(B) de T_i com a instrução *Read*(A) de T_j.

No final, essas trocas resultaram no escalonamento serial da tabela 3.7. Logo, fica claro que, o escalonamento da tabela 3.4 é equivalente a um escalonamento serial da tabela 3.7. Isto é, ao considerar o escalonamento 3.4, o mesmo produzirá um estado final igual ao que o escalonamento serial da tabela 3.7.

Se um escalonamento S pode ser transformado em um escalonamento S' por uma série de trocas nas posições das instruções não-conflitantes, então S e S' são equivalentes quanto ao conflito.

T_0	T_1
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Tabela 3.7. Escalonamento serial equivalente ao escalonamento da tabela 3.4

Considerando-se o escalonamento da tabela 3.8, pode-se construir um grafo direcionado para este escalonamento, chamado grafo de precedência S.
O grafo S consiste em um par G= (V, A), no qual V é conjunto de vértices e A é um conjunto de arestas. O conjunto V consiste de todas as transações participantes do escalonamento. O conjunto A são todas as arestas $T_i \rightarrow T_j$. Para qualquer uma das três condições abaixo é válido:

1. T_i executa *Write*(Q) antes que T_j execute *Read*(Q).

2. T_i executa *Write*(Q) antes que T_j execute *Read*(Q).

3. T_i executa *Write*(Q) antes que T_j execute *Write(*Q).

Quando há um grafo de precedência, uma aresta $T_I \rightarrow T_j$. Logo, para qualquer escalonamento serial S' equivalente a S, T_i precisa aparecer antes de T_j . Na figura 3.4 é exibido o grafo de precedência para o escalonamento da tabela 3.8.

T_i	T_j
Read (A)	
A:= A – Y	
Write(A)	
Read(B)	
B: = B + Y	
Write(B)	
	Read(A)
	Temp := A * 0,1
	A:= A – temp
	Write(A)
	Read(B)
	B:= B + temp
	Write(B)

Tabela 3.8. Escalonamento Serial, onde T_i é seguido por T_j.

O grafo de precedência da figura 3.4 contém uma única aresta $T_I \rightarrow T_j$, uma vez que todas as instruções de T_I são executadas antes de T_j.

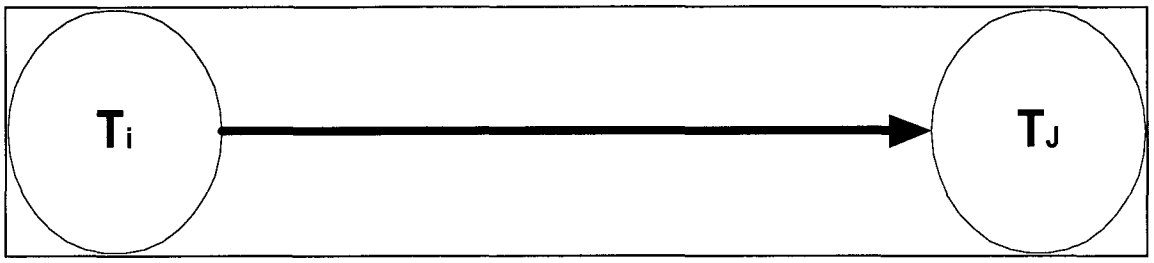

Figura 3.4. Grafo de precedência para T_I e T_J

Como escalonamento da tabela 3.4 é não-serializável, apresentamos na figura 3.5 um grafo para este tipo de escalonamento.

A aresta $T_0 \rightarrow T_1$, pois T_0 executa *Read*(A) antes que T_1 execute *Write*(A). O grafo contem também a aresta $T_1 \rightarrow T_0$, pois T_1 execute *Read*(B) antes que T_0 execute *Write*(B).

Se o grafo de precedência para um escalonamento S possuir um ciclo, então o escalonamento S não é serializável quanto ao conflito. Se o grafo de precedência não tiver ciclos, então o escalonamento é serializável quanto ao conflito. A ordem de seriabilidade pode ser obtida por meio de ordenação topológica que determina uma ordem linear consistente com a ordem parcial do grafo de procedência.

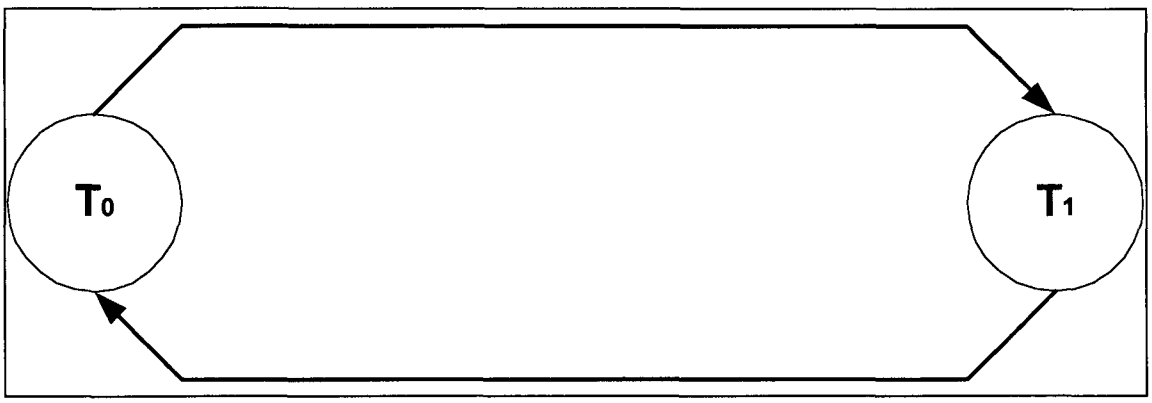

Figura 3.5. grafo de procedência para um escalonamento não-serializável

c) GERÊNCIA DE RECUPERAÇÃO

A gerência de recuperação tem como principal função, nos caso de falha do sistema, identificar quais transações devem ser desfeitas e quais têm que ser refeitas e, em seguida, efetuar as alterações necessárias no bancos de dados. O arquivo *log*,

também chamado de *journal* ou *audit trail*, é um elemento fundamental nesta função, pois é neste arquivo que são registradas todas as operações realizadas por todas as transações do bancos de dados.

Integridade e Segurança

✓ **INTEGRIDADE**

A integridade de um bancos de dados está relacionada à consistência, exatidão, validade e precisão dos seus dados. A violação de integridade ocorre quando inserimos uma chave inválida em uma tabela, falha do algoritmo de controle de concorrência, falha na gerência de recuperação, etc.

✓ **SEGURANÇA**

Os SGBDs locais possuem a responsabilidade da segurança de seus dados. A violação de segurança ocorre quando, de forma deliberada, há tentativa de acesso não autorizado a determinados dados.

Identificação e Autenticação: Para se ter acesso aos dados de um determinado bancos de dados, o usuário/programa de aplicação deve fornecer sua identificação seguida da senha correspondente. Para conexões remotas, pode-se usar o usuário ativo da presente sessão, informando-se o usuário/ senha.

Distribuição das Regras de Autorização de Acesso: Em um ambiente de bancos de dados distribuído, as regras de autorização de acesso aos dados definidos para um nó devem ser replicadas aos outros nós participantes.

Encriptação: A encriptação dos dados é uma forma de impedir acessos não autorizados, além de resolver o problema do acesso por usuários não autorizados que tentam infringir as regras de segurança do SGBD ou que tenham acesso aos *frames* do protocolo da rede de comunicação.

A encriptação dos dados pode ser realizada através de vários métodos, tais como data *Encryption Standart* (DES) ou *Public Key Crystosystems* (PCK).

Visão Global: Através de definições de visões, os dados são disponibilizados de forma restritiva, ou seja, limita-se o número de colunas acessadas como também o

número de linhas em uma tabela. Através dos comandos *GRANT* e *ROVOKE* pode-se limitar-se o acesso ou liberá-lo.

3.2 Taxonomia de Bancos de dados Distribuído

Nesta seção, será apresentada uma taxonomia de bancos de dados distribuído. Num primeiro momento, são apresentados critérios para a classificação dos bancos de dados distribuídos e, após, uma breve descrição de cada táxon. Os critérios utilizados para a classificação estão relacionados aos objetivos do trabalho. Para que se possa contextualizar os problemas de bancos de dados heterogêneos, dentro de bancos de dados distribuído, de qualquer forma, se faz necessário definir que parâmetros serão utilizados.

No caso de bancos de dados distribuídos, a classificação envolve vários aspectos; neste trabalho apresentaremos apenas as relacionadas à autonomia do bancos de dados, sua distribuição, heterogeneidade e interoperabilidade. Na tabela 3.9, podemos observar um exemplo de classificação para bancos de dados distribuído em função da heterogeneidade e do tipo de rede de computadores, levando em consideração o tipo de aplicação.

Tipo de SGBD	Tipo de rede de computadores	
	LAN	WAN
Homogêneos	Gerenciamento de dados para aplicações financeiras.	Gerenciamento de dados para aplicações financeiras e passagens aéreas.
Heterogêneos	Sistemas de informações entre empresas filiais e matrizes	Integração de Bancos e sistemas entre agências bancárias

Tabela 3.9. Classificação para SGBDs com exemplos de tipos de aplicações.

3.2.1 Heterogeneidade

A heterogeneidade em sistemas distribuídos pode ocorrer de diferentes formas: heterogeneidade de hardware, de protocolos de redes e de gerenciadores de dados [Özsu e Valduriez. 1999]

É importante ressaltar que a heterogeneidade é independente da distribuição física dos dados. Sistemas de informações ou apenas bancos de dados podem estar localizados em nós geograficamente diferentes e serem homogêneos. Um sistema é considerado homogêneo se o software que manipula e cria os dados é o mesmo em todos os nós do bancos de dados e todos os dados têm a mesma estrutura e modelo de dados. Já em bancos de dados heterogêneo, o modelo de dados e estrutura é totalmente diferente ou parcialmente diferente. Isto é, heterogeneidade pode acontecer em todos os níveis do sistema de bancos de dados. Diferentes nós da base de dados podem usar diferentes linguagens de programação para escrever diferentes aplicações, diferentes linguagens de consulta, diferentes modelos, diferentes SGBDs, diferentes sistemas, etc.

Considerando-se apenas a base de dados, podemos ter:

1. Bancos de dados locais baseados no mesmo modelo

Nestes bancos de dados há total homogeneidade em relação aos modelos utilizados. Porém a complexidade para a execução das transações globais é menor em relação aos bancos de dados heterogêneos.

2. Bancos de dados locais baseados em modelos diferentes

São os banco de dados que se baseam modelos de dados diferentes, isto é heterogeneidade de modelos. Neste banco de dados têm-se duas duas abordagens para trabalhar:

✓Modelo Canônico (ou global)

✓Modelo que fornece tradutores entre os diferentes modelos

3. Heterogeneidade semântica

Ocorre quando há discordância sobre o significado, interpretação ou intenção de utilizar os mesmos dados ou dados relacionados [Larson e Sheth. 1990]

3.2.2 AUTONOMIA

A autonomia está relacionada com a distribuição do controle do sistema, isto é, controle global versus controle local. Os sistemas homogêneos tem menos autonomia do que os heterogêneos. As principais formas de autonomia são:

- **Autonomia de projeto**

Individualmente os sistemas gerenciadores de bancos de dados são livres para usar o modelo de dados, linguagens de consulta, interpretação semântica dos dados, técnicas de gerenciamento de transação e etc., que forem mais adequados.

- **Autonomia de comunicação**

Cada sistema gerenciador de bancos de dados local é livre para tomar as suas próprias decisões e também que tipo de informação será permitido aos outros sistemas gerenciadores de bancos de dados distribuído acessarem.

- **Autonomia de execução**

Cada sistema gerenciador de bancos de dados distribuído pode executar as transações que lhe são submetidas e da forma que lhes convêm. O componente do bancos de dados não distingue se a operação é local ou global, apenas as executa.

- **Autonomia de Associação**

Bancos de dados locais podem decidir como as funções/operações e os dados serão compartilhados com uma certa classe de usuários. As informações estatísticas como custo, eficiência, velocidade de execução do processamento da informação, são também, determinadas individualmente pelo bancos de dados, assim como, o custo de processamento de consultas globais e a otimização.

O bancos de dados local tem a habilidade de associar ou desassociar, por si mesmo, a rede do bancos de dados.

3.2.3 DISTRIBUIÇÃO DOS DADOS

Em muitos ambientes e/ou aplicações, os dados são distribuídos em vários bancos de dados. Esses bancos de dados podem estar armazenados em um ou vários computadores que são tanto localmente centralizados ou geograficamente distribuído, mas interconectados através de elos de comunicação.

Os dados podem ser distribuídos em múltiplas bases de dados através de várias maneiras. Isto inclui, em termos relacionais, partições horizontais e verticais. Ao distribuir os dados pelos computadores em um ambiente distribuído, deve-se considerar

dois objetivos:

✓ Maximizar o paralelismo "natural" existente em um ambiente distribuído;

✓ Minimizar o tráfego de dados entre os nós

As principais técnicas utilizadas para a distribuição dos dados são:

- **REPLICAÇÃO**

O SGBDD mantém várias réplicas da relação, armazenadas em nós diferentes. Consequentemente teremos maior disponibilidade, maior paralelismo, menos tráfego entre nós, atualizações mais lentas, recuperação e controle de concorrência mais complexos.

A replicação pode ser de duas formas:

✓**Total**, isto é, o bancos de dados terá cópias completas em todos os nós do sistema.

✓**Parcial**, isto é, o bancos de dados terá partes replicadas pelos nós do sistema.

- **FRAGMENTAÇÃO**

Consiste em dividir a relação r em um conjunto de fragmentos r_1, r_2, ... , r_n , de modo que seja possível reconstruir a relação original a partir deles. Há três formas de fragmentação: Horizontal, Vertical e Mista

1) Fragmentação Horizontal

Divide a relação r levando em consideração as tuplas -- Seleção.

Cada tupla precisa ser destinada a um dos fragmentos horizontais

$r = r_1 \cup r_2 \cup ... \cup r_n$

2) Fragmentação Vertical

Divide a relação segundo seus atributos, isto é, decompondo o esquema R da relação -- Projeção.

$R = R_1 \cup R_2 \cup ... \cup R_n$

$r = r_1$ JOIN r_2 JOIN ... JOIN r_n

É preciso adicionar uma chave primária da relação a cada tupla.

3) Fragmentação Mista

A relação r é dividida em uma série de fragmentos, obtidos por sucessivas seleções ou projeções sobre a relação r ou outros fragmentos. Replicação e fragmentação podem ser aplicadas em conjunto.

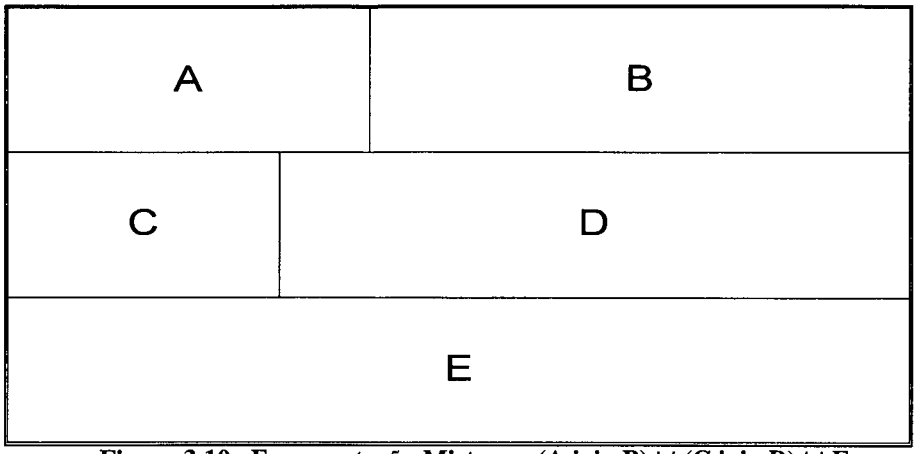

Figura 3.10. Fragmentação Mista - r=(A join B) ∪ (C join D) ∪ E

- **TRANSPARÊNCIA**

Transparência, neste caso, significa a capacidade que o bancos de dados tem de esconder a distribuição dos dados dos usuários do sistema, podendo ser de dois tipo:

1) Transparência de Replicação e Fragmentação

Um usuário não deve se referir a uma réplica específica de um item de dado, nem se preocupar com a forma pela qual um dado é armazenado.

Em um "read", o sistema deve determinar qual réplica deve ser lida e, em um "write", atualizar todas as réplicas existentes.

Garantir a transparência ⇔ Manutenção de visões

2) Transparência de Acesso e Localização

Os comandos para fazer acesso a um item de dados devem ser sempre os mesmos, quer este seja remoto ou local.

O nome não deve fornecer informações acerca da localização do dado, isto é deve possuir tanto trasnparência de migração como de Rede.

- **RECUPERAÇÃO À FALHAS**

Em BD Distribuídos, o problema de recuperação de falhas é bem mais complexo,

pois uma transação global só terá sucesso se cada transação local individual também for concluída sem problemas.

Em um sistema distribuído, surgem alguns tipos de falhas, tais como: falha de um nó; falha de uma conexão com um nó; perda de mensagens na rede; interrupção da rede.

Os sistemas robustos precisam detectar essas falhas e reconfigurar-se rapidamente, a fim de continuar a atender seus usuários de forma satisfatória. Se existirem dados replicados armazenados no nó que falhou, as consultas não mais devem se referenciar àquele nó. Deve-se abortar imediatamente as transações que estavam ativas naquele nó no momento da falha. Se o nó que falhou era uma espécie de servidor para algum subsistema, deve-se eleger um novo. Exemplo: servidor de nomes, de concorrência, etc.

3.2.4 INTEROPERABILIDADE

Interoperaperabilidade de um sistema é a capacidade de habilitar a solicitação e o recebimento de serviços entre os sistemas interoperantes, além de utilizar as funcionalidades dos sistemas envolvidos. Uma forma limitada de interoperação é quando há intercâmbio de dados por meio de um sistema que é capaz de enviar dados periodicamente a um outro sistema recipiente.

Interdepência de sistemas implica que os dados e as funções de diferentes sistemas são relacionados ou dependentes um do outro, mesmo que para os usuários ou aplicações esta situação não seja aparente. Logo, o gerenciamento de dados interdependente implica no reforço das regras de consistência do bancos de dados múltiplos.

Geralmente, são consideradas sistemas de informação interoperáveis aqueles que obedecem às seguintes condições:

✓Podem intercambiar mensagens e solicitações;

✓Podem receber serviços e operar como uma unidade na resolução de problema comum.

As condições acima sugerem que para um sistema de informação ser interoperável devem ter as seguintes características:

✓Uso das funcionalidades dos sistemas envolvidos;

✓Habilidades cliente – servidor;

✓Comunicação apesar de haver incompatibilidades de detalhes internos dos componentes;

✓Distribuição;

✓Extensibilidade e fácil evolução.

A tabela 3.11, exibe um resumo da taxonomia apresentada.

Hogeneidade	Autonomia	Distribuição
Homogêneos	Autônomos	Fragmentado
Heterogênos	Não Autônomos	Replicado

Tabela 3.11. Resumo da taxonomia apresentada

3.3 Considerações Finais

Neste capítulo foram apresentados os conceitos básicos necessários para o desenvolvimento do tema da dissertação. Foram abordados, com mais ênfase, os assuntos relacionados a gerência e serialização de transações distribuída, além da taxonomia para bancos de dados distribuídos.

Através desse conceitos pode-se contextualizar o tema da dissertação, pois, criam-se condições básicas para a discussão sobre controle de concorrência em ambientes distribuídos heterogêneos.

CAPÍTULO 4

ARQUITETURA DE BANCOS DE DADOS DISTRIBUÍDOS HETEROGÊNEOS

4.0 INTRODUÇÃO

Sistemas de informação que permitem interoperação e vários graus de integração sobre múltiplas base de dados estão sendo definidos como sistemas de bancos de dados múltiplos, bancos de dados federados e, mais genericamente de bancos de dados distribuídos heterogêneos (BDDHs). A forma para relacionar os termos mais freqüentemente usados vem da fundamental dimensão heterogeneidade e autonomia, conforme apresentado por James A. Larson e Amit P. Sheth [Larson e Sheth. 1990].

As dimensões mais utilizadas, inicialmente, em sistemas de bancos de dados heterogêneos eram o tipo de heterogeneidade e funcionalidade [Sheth. 1987]. A primeira dimensão é mostrada na tabela 4.1, onde são exibidos os tipos mais comuns de de heterogeneidade que eram manipulados pelos sistema de bancos de dados heterogêneos. Há vários anos, pesquisadores e fabricantes de sistemas gerenciadores de bancos de dados distribuídos heterogêneos (SGBDDHs) têm trabalhado no desenvolvimento de interfaces que garantam a integração sobre a heterogeneidade de hardware/sistema, comunicação e sistema operacional.

<table>
<tr><td colspan="2">Banco de Dados

SGBDs
Modelos de dados
Tipos de dados</td></tr>
<tr><td>Sistemas Operacionais

Sistemas de Arquivos
Tipos de Arquivos e operações
Suporte a Transações
Comunicação Interprocessos</td><td rowspan="2">Comunicação</td></tr>
<tr><td>Hardware/Sistema

Conjunto de Instruções
Formato dos Dados e Representação
Configuração</td></tr>
</table>

Tabela 4.1 - Tipos mais comuns de Heterogeneidade.

A Segunda dimensão de um sistema de bancos de dados heterogêneos era a funcionalidade. A primeira perspectiva é que os sistemas de bancos de dados heterogêneos deveriam prover as mesmas funcionalidades que são tipicamente esperadas de um sistema de bancos de dados homogêneo. Isto é, os sistemas de bancos de dados heterogêneos deveriam permitir, também, transparência de localização, controle de concorrência para dados replicados ou fragmentados e tolerância a falhas.

No contexto atual, a integração de dados implica acesso uniforme e transparência pelo gerenciador de dados das múltiplas base de dados. O mecanismo de arquivamento utilizado é um esquema de integração que envolve todas as partes do sistema

Em um Sistema gerenciador de bancos de dados distribuídos heterogêneos (SGBDDHs), não é necessário ter um simples (global) esquema de integração do sistema. Larson e Sheth [1990] mostram de forma clara a possibilidade de se ter múltiplos esquemas de integração, também chamados de esquemas federados, além de obter exemplos de sistemas que suportam múltiplos esquemas integrados.

Neste capítulo são apresentados o esquema de integração global, o bancos de dados federados e o sistema de bancos de dados múltiplos, em função do alto grau de integração dos componentes do sistema.

Historicamente, há várias maneiras para se negociar entre o compartilhamento e a autonomia do bancos de dados. Esta negociação pode ser resumida da seguinte forma: Quanto mais compartilhados os dados do bancos de dados, teremos menor autonomia. Neste caso, o uso de esquema integrado aumenta o compartilhamento de dados enquanto reduz a autonomia do bancos de dados.

Outra classificação é a arquitetura de referência proposta por [Larson e Sheth. 1990] e [Roussoupoulos et al. 1990], onde é mostrada a divisão da arquitetura de sistemas multibase de dados em cinco: bancos de dados distribuído, esquema global de bancos de dados múltiplos, bancos de dados federados, sistema de linguagens de banco de dados múltiplos e bancos de dados interoperáveis. Na seção 4.1 é apresentada a Integração de esquema Global, na seção 4.2 Bancos de dados Federados, na seção 4.3 Sistemas de bancos de dados múltiplos.

4.1 ESQUEMA DE INTEGRAÇÃO GLOBAL

O esquema de integração foi definido como a primeira tentativa de compartilhamento de dados sobre um sistema gerenciador de bancos de dados distribuídos heterogêneos (SGBDDHs), baseado na completa integração de múltiplas base de dados para produzir um esquema global.

A vantagem deste tipo de integração, é que os usuários têm uma consistente e uniforme visão do acesso aos dados. Os usuários não percebem que o bancos de dados em uso é distribuído e heterogêneo, pois múltiplas bases de dados que aparecem logicamente, são, na verdade, simples bancos de dados. Entretanto, o esquema de integração global tem várias desvantagens:

✓ **Complexidade para automatizar:** é difícil identificar os relacionamentos entre atributos de dois esquemas, relacionamento entre tipos de entidades e tipos de relacionamentos. O problema de integração de esquemas relacionais tem mostrado não ter uma solução geral. O entendimento humano tem sido necessário para solucionar muitos tipos de conflitos semânticos, estrutural ou comportamental.

✓**Autonomia:** especialmente a autonomia de associação, é sempre sacrificada para solucionar os conflitos semânticos. Todos os bancos de dados envolvidos precisam revelar as informações sobre o seu esquema conceitual ou dicionário de dados, assim como o processo de integração requer total conhecimento prévio da semântica. Algumas vezes, estes processos requererem uma igualdade ao bancos de dados local para alterar o esquema e facilitar a integração.

✓ **Múltiplos métodos de integração:** se há mais de dois bancos de dados, haverá vários métodos de integração. Pode-se considerar todos de uma vez, ou dois ao mesmo tempo e então combiná-los no final. Dependendo da ordem na qual os esquemas são integrados, somente a semântica de conhecimento incompleta é usada em cada passo. Como resultado, algumas semânticas de conhecimento podem ser perdidas no final do esquema global sem integração.

4.2 BANCOS DE DADOS FEDERADOS - BDFS

Sistema de Bancos de dados Federados é uma coleção de componentes de bancos de dados cooperantes, autônomos e possivelmente heterogêneos. A figura 4.1 exibe a arquitetura de um SBDFs.

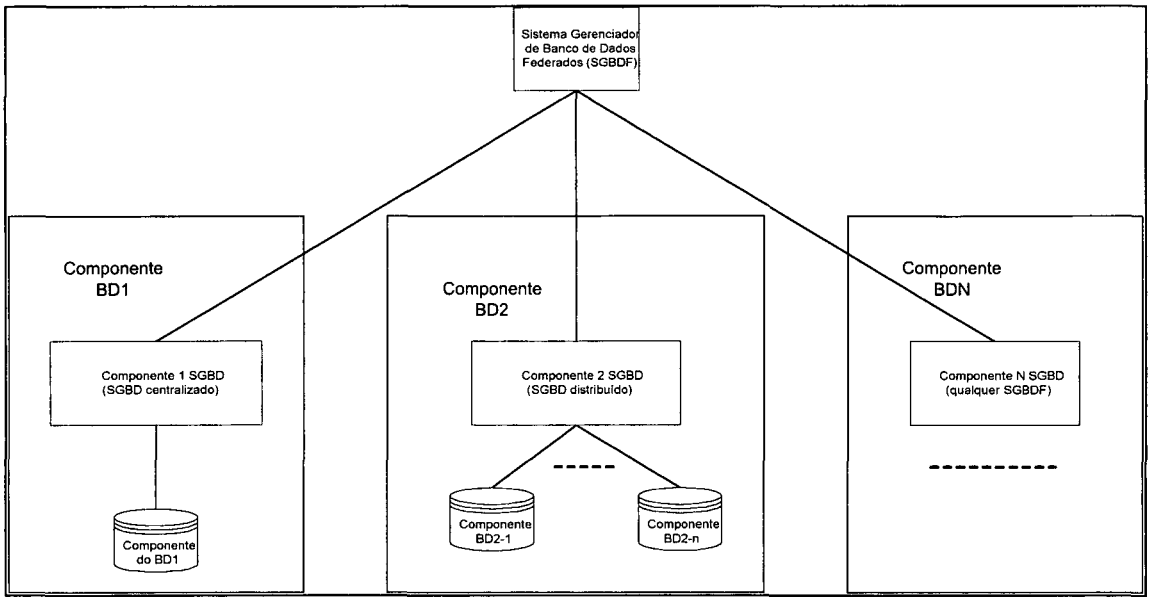

Figura 4.1. Bancos de dados Federados e seus componentes

O sistema de bancos de dados federados (SBDFs) permite acesso a um conjunto de sistemas gerenciadores de bancos de dados (SGBDs) locais pré-existentes. A heterogeneidade e a autonomia são os dois principais aspectos de um BDFs. A heterogeneidade de um BDFs refere-se aos vários tipos de SGBDs locais participantes, os quais podem empregar diferentes interfaces, modelos de dados, linguagens de consultas e mecanismos de gerenciamento de transações.

A autonomia assegura que os SGBDs local não devem ser alterados por um SGBDFs global e um SGBDs local tem o direito de decidir que tipo de informação interna deve produzir para o BDFs e executar consultas e transações de acordo com as sua próprias regras [Larson e Sheth. 1990]. Devido à autonomia, há dois tipos de transações em um ambiente de SGBDFs chamadas de transações globais e transações locais.

Uma transação local, é aquela que acessa dados controlados por um simples SGBDs, é submetida e executada por um SGBDs local sem o controle do SGBDFs global. Já a transação global, que acessa dados controlados por mais de um SGBDs, é

submetida ao SGBDFs global sendo decomposta em várias subtransações para serem executadas em diferentes SGBDs. O objetivo da gerência de transações em BDFs é garantir a execução serializável das transações globais e locais [Kung e Robison. 1981].

A arquitetura de bancos de dados federados tem por objetivo eliminar a necessidade de um esquema de integração estático global, isto é, permite mais controle sobre a informação compartilhada, pois se tem mais autonomia de associação, sobre bancos de dados independentes, em ambientes cooperativo. O controle, consequentemente, é descentralizado em um BDFs.

O grau de integração, nos BDFs, não deve ser completo em um esquema global de integração, pois depende da necessidade dos usuários. Os BDFs podem ser também sistemas rigidamente ou levemente acoplados, pois os BDFs são, na verdade, um compromisso entre a não integração e a total integração [Elmagarmid et al. 1999]. A seguir serão apresentados os tipos de bancos de dados federados.

4.2.1 Bancos de dados Federados Fracamente Acoplados

São bancos de dados nos quais os usuários têm a responsabilidade de criar um esquema de federação fracamente acoplado ao SGBDFs, isto é, não possuem esquema global.

Os sistemas federados fracamente acoplados são altamente autômos, somente realizam leituras no bancos de dados federado, não suportam operações de atualizações, sendo suas principais vantagens:

> Diferentes classes de usuários da federação têm flexibilidade para mapear diferentes ou múltiplas semânticas do mesmo conjunto de objetos.

> Sistemas Fracamente Acoplados permitem alterações dinâmicas de componentes ou esquemas de exportações melhores que os sistemas fortemente acoplados, pois são mais fáceis de construir visões do que criar esquemas globais desde o início. Contudo, a detecção de alterações dinâmicas em um esquema de exportação para bancos de dados remoto pode dificultar o arquivamento e uma sobrecarga da rede, assim como as *triggers* podem introduzir também muitas mensagens de *broadcast*.

Algumas da desvantagens dos BDF Fracamente Acoplados são:

Se dois ou mais usuários independentes acessarem informações similares de um mesmo componente do bancos de dados, eles criarão suas visões/mapeamento, sem saber se outros usuários possuem as mesmas visões/mapeamento anteriormente criadas. Isto cria um grande potencial de trabalho duplicado na criação de visões e no entendimento do esquema de exportação. Outro problema é a dificuldade de entender o esquema de exportação, quando o número de esquemas é grande.

Devido aos múltiplos mapeamentos semânticos entre objetos, visões de atualizações podem não ser bem suportadas, podendo criar dificuldades.

4.2.2 BANCOS DE DADOS FORTEMENTE ACOPLADOS

Os sistemas gerenciadores de bancos de dados federados fortemente acoplados são compostos por um conjunto de SGBDs locais, integrados de forma que a localização e o caminho de acesso aos dados são transparentes ao usuários [Özsu e Valdurez. 1999].

Nestes sistemas, os administradores da federação têm total controle na criação e manutenção do esquema federado. O objetivo é prover localização, replicação e transparência de distribuição.

Os BDF Fortemente Acoplados suportam um ou mais esquemas federados, pois um simples esquema ajuda a manter a uniformidade e a semântica interpretativa dos múltiplos componentes de dados integrados. As principais desvantagens dos BDF Fortemente Acoplados são:

Os administradores do SGBDFs e os administradores (DBAs) dos componentes dos bancos de dados deverão negociar para fazerem o esquema de exportação. Porém, durante a negociação, os administradores dos SGBDFs podem permitir a leitura dos componentes do esquema sem acesso a nenhum dado. Esta situação viola a autonomia.

Quando um esquema federado é criado pela primeira vez, raramente ocorrerá mudanças, isto é, o esquema é estático. Quando forem necessárias alterações no

componente/esquema de exportação, será necessário fazer integrações desde o início para cada esquema federado.

4.3 Sistemas de Bancos de Dados Múltiplos - SBDM

Os sistemas de bancos de dados múltiplos são uma coleção coerente, integrada de vários bancos de dados, cada qual controlado por um sistema gerenciador de bancos de dados (SGBDs), mas que logicamente aparenta ser um único bancos de dados, sendo implementado em vários bancos de dados. O SBDM é um tipo especial de bancos de dados distribuído, em que cada bancos de dados local participante é um sistema gerenciador de bancos de dados múltiplos (SGBDM) autônomo.

Um SBDMs cria a ilusão de um simples bancos de dados; isto permite aos usuários manipularem dados contidos em vários bancos de dados sem modificações nas aplicações do bancos de dados corrente e sem migração do dado para o bancos de dados corrente [Breitbart et al. 1988].

Os SGBDM escondem dos usuários as interfaces de diferentes SGBDs e diferentes métodos de acesso, o que permite acesso uniforme aos bancos de dados preexistentes, sem requerer que o usuário saiba a localização ou a característica de diferentes base de dados e seus correspondentes SBDMs.

As consultas e as linguagens de manipulações de dados, nos SGBDM, permitem aos usuários acessar múltiplos bancos de dados preexistentes com uma simples consulta de uma aplicação.

Os bancos de dados que participam do SGBDM são geralmente heterogêneos, e os usuários não necessitam saber como e onde os dados estão armazenados e como são acessados.

4.3.1 Componentes de um SGMBD

Um SGMBD consiste de componentes globais e locais, como descrito na figura 4.2. Interações como o bancos de dados global/local são conduzidas por programas de usuário chamados de transações globais/locais. Uma transação global é uma transação que é submetida ao SGBDM sendo executada sob o controle do SGBDM.

A transação local, de forma oposta à global, é submetida ao SGBDs local, aparentemente controlada pelo SGBDM.

O gerenciador de transações globais (GTG) controla a execução das transações globais. Para cada transação global, o GTG aloca um gerenciador de transações local (GTL), para cada nó do bancos de dados referenciado pela transação. Cada GTL é responsável pela tradução da operação de leitura e escrita global dentro da linguagem do SGBDs, transferindo e recuperando dados para o GTG. Sempre que um gerenciador de transações locais é alocado, este não poderá ser desalocado até que a transação tenha realizado *commits* ou *aborts* [Breitbart et al. 1988].

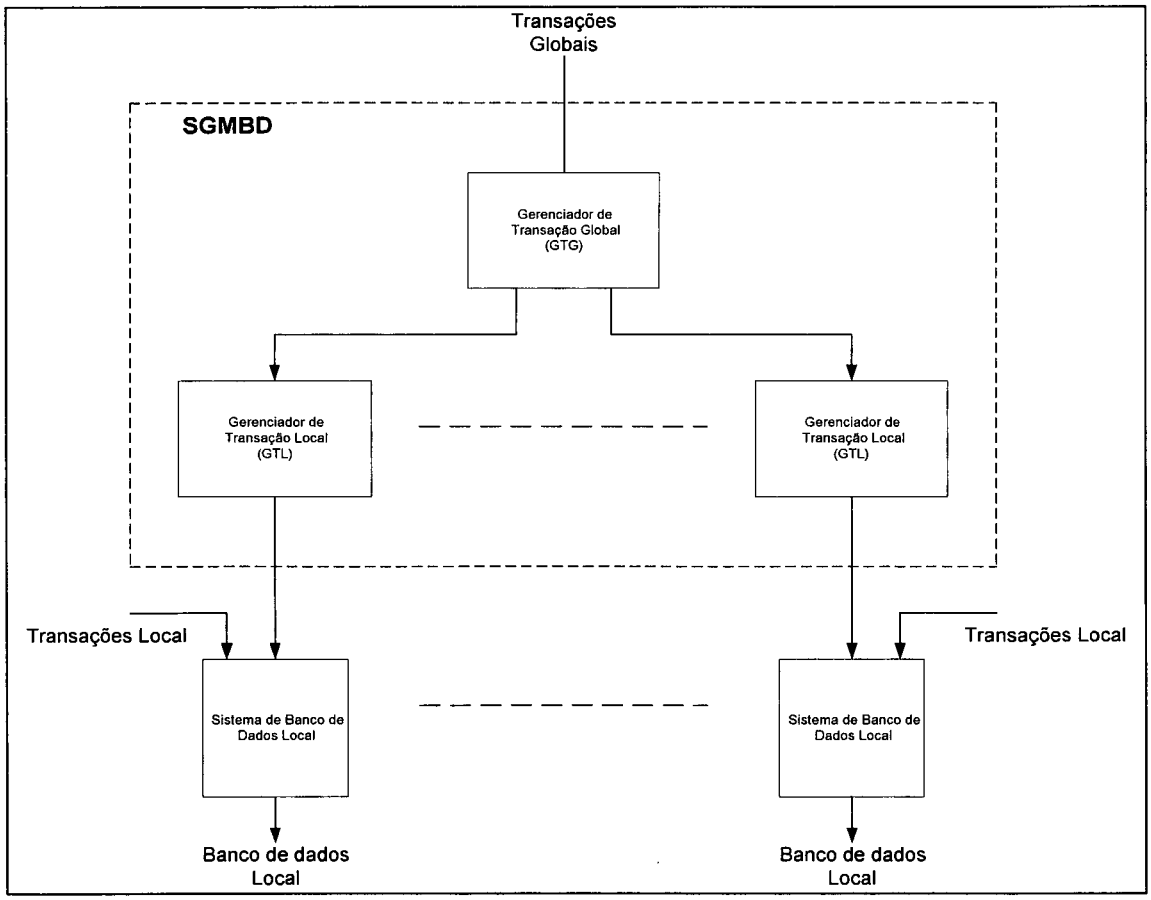

Figura 4.2. Modelo para Sistemas de bancos de dados múltiplos

4.3.2 ABORDAGEM À LINGUAGEM DE BANCOS DE DADOS MÚLTIPLOS

A abordagem para uma linguagem de um sistema de bancos de dados múltiplos é compreendida pelo usuário que não usa um esquema global ou parcial pré-definido. Preexistindo um SGBD heterogêneo local é geralmente integrado sem modificações.

As informações armazenadas em diferentes bancos de dados podem ser redundantes, heterogêneas e inconsistentes. Esses problemas ocorrem quando os componentes do sistema são fortemente autônomos.

O objetivo de uma linguagem para bancos de dados múltiplos é fornecer construtores que realizam consultas envolvendo vários bancos de dados ao mesmo tempo. Uma linguagem tal, tem as características que não são suportadas nas linguagens tradicionais. Por exemplo, um nome global pode ser usado para identificar uma coleção de bancos de dados. As consultas podem especificar dados de qualquer bancos de dados local participante.

A maior censura à abordagem de linguagens de bancos de dados múltiplos é a falta de transparência, para os usuários, de localização e distribuição dos dados, pois os usuários têm de achar, *a priori*, a informação correta em uma potencialmente grande rede de bancos de dados. Os usuários são responsáveis por entender esquemas detectando e resolvendo conflitos de semânticas.

As linguagens para bancos de dados múltiplos fornecem operadores adequados e construtores de expressões para os usuários efetuarem as resoluções dos conflitos de semântica em vários níveis de abstração. Uns dos interessantes aspectos da linguagem são a nomeação global, dependência e consultas inter-bancos de dados.

No geral, nesta abordagem, usuários estão de frente com algumas questões, como: encontrar informações relevantes em múltiplos bancos de dados, entender cada esquema individual de bancos de dados, detectar e resolver conflitos de semântica criando uma visão de integração.

4.3.3 Esquema de Tradução

Um aspecto importante dos sistemas de bancos de dados múltiplos é o suporte a tradução entre modelos de dados globais e locais. Vários modelos de dados têm sido usados para projetar um universo de discurso: modelo hierárquico, modelo em rede, modelo relacional, modelo semântico e modelo orientado-objeto [Korth e Silberschatz. 1998].

Em geral, quando se integra bancos de dados heterogêneo, os esquemas locais são traduzidos para um modelo comum de dados, o que permite resolver a

heterogeneidade semântica resultante do uso de diferentes modelos de dados. Por exemplo, em um sistema de bancos de dados múltiplos, tem-se como componentes um bancos de dados relacional e um bancos de dados em rede; o modelo comum de dados usa o modelo funcional e isso é geralmente esperado, pois o poder da modelagem do modelo comum de dados é mais rico que o modelo usado pelos componentes do bancos de dados individual.

O modelo relacional vem sendo freqüentemente usado como modelo comum de dados nos sistemas de bancos de dados múltiplos. Desde de que o Modelo de Entidada-Relacionamento (ER) passou a ser uma ferramenta irresistível para modelagem conceitual [Roussoupoulos et al. 1990], esforços recentes nas pesquisas de tradução de modelagem de dados enfocam a transformação do modelo ER. Recentemente, o modelo ER vem sendo substituído pelo modelo orientado-objeto (MOO), pois o mesmo é visto, geralmente, como um modelo padrão mais apropriado [Stonebraker e Brown. 1999].

4.3.4 Esquema de Integração

Uma instituição tem múltiplos sistemas gerenciadores de bancos de dados. Diferentes setores ou departamentos, dentro da instituição, requisitam diferentes tipos de informações, podendo selecioná-las de diferentes SGBDs; este o cenário mais comum encontrado, atualmente, nos ambientes de trabalho. Como os SGDBs, são adquiridos em vários períodos, podem ser diferentes, além de utilizarem tecnologias variadas [Elmagarmid et al. 1999].

No cenário descrito acima, a necessidade de integração dos bancos de dados é grande. Se os modelos de dados utilizados pelos bancos de dados forem homogêneos, não há necessidade da fase de tradução do modelo, pois não haverá heterogeneidade entre os modelos utilizados.

O esquema de integração é realizado após o processo de tradução, gerando o esquema conceitual global para integrar os esquemas intermediários. O esquema de integração é um processo de identificação dos componentes do bancos de dados ao qual são relacionados uma ao outro, selecionando a melhor representação do esquema conceitual global e, finalmente, integrando os componentes de cada esquema intermediário. Há algumas metodologias de integração, classificadas como mecanismos

binários ou n-ário [Batini et al. 1986], como mostra a figura 4.3.

O mecanismos de integração binários envolvem a manipulação de dois esquemas ao mesmo tempo. Isto ocorre criando-se pares de esquemas intermediários, que são integrados em uma nova etapa subsequente.

Os mecanismos de integração n-ários são implementado através do método de integração em um passo e método interativo. Quando todos os esquemas são integrados de uma vez, produzem o esquema conceitual global; como utilizou uma única interação, é conhecido como *one pass integration* [Özsu e Valdurez. 1999] (figura 4.3 – mecanismo de integração n-ária exibe esta situação).

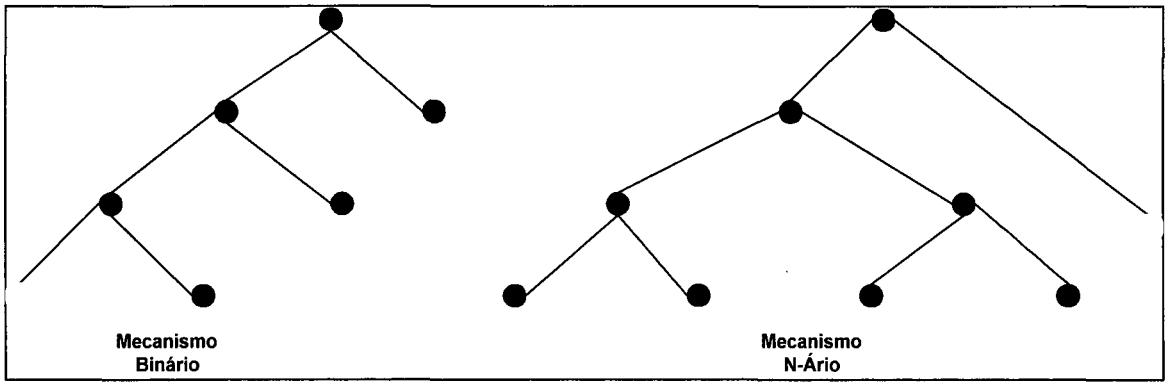

Figura 4.3 Mecanismos de Integração Binário e N-Ário

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada a arquitetura de bancos de dados distribuídos heterogêneos, através do esquema de integração global, bancos de dados federados e sistemas de bancos de dados múltiplos.

Apresentou-se a necessidade de um esquema global de integração, em sistemas de bancos de dados heterogêneos, além dos esforços dos pesquisadores em criar uma interface para atenuar as dificuldades em face da heterogeneidade dos sistemas implantados nos ambientes de trabalho.

Há autores como Bharat Bhargava [Bhargava.1987] e [Bhargava.1999], consideram os bancos de dados federados como um subtipo dos bancos de dados múltiplos. Porém, em função das características dos bancos de dados federados como: autonomia, controle descentralizado, entre outras, baseando-se nas afirmações de alguns autores como San-Yih Hwang [Hwang et al. 1981], James A. Larson e Amit P. Sheth

[Larson e Sheth.1990] e Csaba J. Egyhazy [Egyhazy.1991]. Consideram-se, neste trabalho, os bancos de dados federados como um tipo de bancos de dados distribuídos heterogêneos, em função da forte autonomia apresentada nos banco da dados participantes desse tipo de esquema, como também da não interferência na execução das transações de cada bancos de dados individualmente.

CAPÍTULO 5

CONTROLE DE CONCORRÊNCIA

5.0 INTRODUÇÃO

Um SGBD deve, com freqüência, servir a várias aplicações, além de responder a requisições de muitos usuários. A carga de aplicações de um SGBD pode ser mensurada através do número de transações por segundo (tps), mas que são gerenciadas pelo próprio SGBD. Além disso, deve satisfazer as necessidades das aplicações.

É essencial que as transações executadas simultaneamente possam ser realizadas em seqüência [Thomasin. 1996], pois apenas o controle de concorrência das transações permite uma eficiente operação em um SGBD, em que há necessidade de maximizar o número de transações realizadas por segundo e minimizado os tempos de resposta.

Apresentaremos a seguir: a arquitetura do controle de concorrência, as anomalias encontradas na execução de transações concorrentes, taxanomia dos mecanismos de controle de concorrência e teoria do controle de concorrência.

5.1 ARQUITETURA DO CONTROLE DE CONCORRÊNCIA

O sistema de controle de concorrência refere-se ao mais baixo nível da arquitetura de um SGBD [Atzeni et al. 2000] e está relacionado as operações de entrada/saída, onde é realizada a transferência de blocos da memória secundária para a memória principal e vice-versa. Na figura 5.1 é exibido o esquema simplificado da arquitetura do controle de concorrência.

Considerem-se uma operação de escrita e uma de leitura. Cada operação de leitura consiste de transferência de um bloco da memória secundária para a memória principal e a escrita consiste da operação oposta. Normalmente, os blocos são carregados dentro da memória como páginas.

As operações de leitura e de escrita são gerenciadas pelo módulo do sistema conhecido como escalonador (*scheduler*), o qual determina se a requisição pode ser satisfeita.

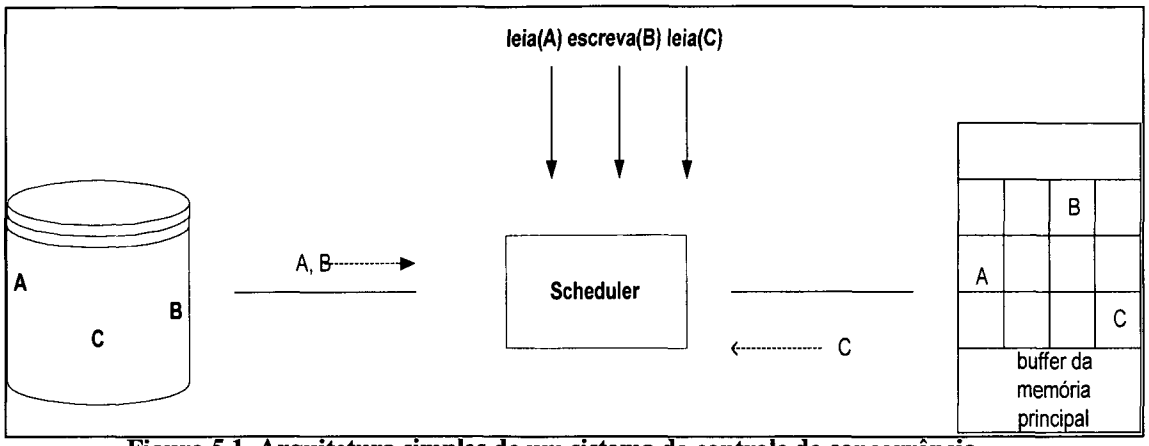

Figura 5.1. Arquitetura simples de um sistema de controle de concorrência

5.2 ANOMALIAS DE TRANSAÇÕES CONCORRENTES

A execução simultânea de várias transações pode causar problemas, conhecidos como anomalias; é necessário um sistema de controle de concorrência para gerenciar tais problemas. A seguir serão analisadas as situações de operações não efetivadas/perdidas e de inconsistência no bancos de dados:

a) OPERAÇÕES NÃO EFETIVADAS OU PERDIDAS

Caso 1

Supondo-se que há duas transações iguais e que estão operando no mesmo objeto do bancos de dados:

T_1: r(a), a = a + 1, w(a)

T_2: r(a), a = a + 1, w(a)

onde **r(a)** é uma representação de uma leitura de um objeto genérico a, e **w(a)** representa a escrita do mesmo objeto. A troca do valor no objeto **a** é realizada por um programa da aplicação. Suponha-se que o valor inicial de **a** é 2 (a = 2). Se forem realizadas duas transações T_1 e T_2 em seqüência. Supondo-se que as ações aconteceram instantaneamente, como mostra a tabela 5.1, no final, o valor de **a é 3**, porque ambas as transações T_1 e T_2 leram o valor de a, somam 1 a este valor. Porém, a transação que vai ser verdadeiramente escrita é T_1 . Esta anomalia é chamada de perda de atualização, pois o efeito da transação $\mathbf{T_2}$ (a primeira para escrever o novo valor para **a)** é perdida.

Transação T_1	Transação T_2
$r_1(a)$ $r_1(a)$	
a = a + 1	
	$R_2(a)$
	a = a + 1
	Commit
$w_1(a)$	
commit	

Tabela 5.1. Execução da Transação T_1 e T_2 em seqüência

Caso 2

Tem-se uma transação T_1 que é primeiramente executada, porém é cancelada. Nesta transação têm-se as operações r (leitura) e w(escrita). A tabela 5.2 mostra o esquema da execução. O valor de **a** no final da execução é **4**, mas deveria ser **5.** O problema crítico nesta execução é a leitura da transação $\mathbf{T_2}$, na qual aparece como estado intermediário gerado pela transação $\mathbf{T_1}$.

A transação $\mathbf{T_2}$, contudo, não deveria aparecer neste estado, pois é produzido pela transação $\mathbf{T_1}$, a qual é subseqüentemente executada e abortada. Esta anomalia é conhecida como leitura suja (*dirty read*), isto é, o pedaço do dado que é lido representa um estado intermediário no processamento da transação. Nota-se que somente a maneira de restaurar a consistência seguida do abort $\mathbf{T_1}$ será imposto o aborto de $\mathbf{T_2}$. Esta situação é extremamente danosa para o gerenciamento, sendo conhecida como efeito dominó.

Transação T_1	Transação T_2
$r_1(a)$	
a = a + 1	
$w_1(a)$	
	$r_2(a)$
	a = a + 1
	commi
Abort	

Tabela 5.2: Leitura de "lixo"

b) INCONSISTÊNCIA NO BANCOS DE DADOS

Caso 1

Supondo-se que a transação T_1 executa somente operações de leitura, mas que as leituras são executadas repetidas vezes no dado **a** em instantes sucessivos, como é descrito na tabela 5.3. Neste caso, **a** assume o valor de 2 antes da primeira operação de leitura e o valor de 3 antes da segunda operação de leitura. Em vez disso, é conveniente que a transação que acessou o bancos de dados duas vezes ache exatamente o mesmo valor para cada pedaço do dado lido e não é efetuada por outra transação.

Transação T_1	Transação T_2
$r_1(a)$	
	$r_2(a)$
	a = a + 1
	$w_2(a)$
	commit
$r_1(a)$	
Commit	

Tabela 5.3. Inconsistência de leitura

Caso 2

Supondo-se que um bancos de dados com três objetos **a, b** e **c**, o qual satisfaz as regras de integridade, como **a + b + c = 1000**; assume-se que a execução da transação é como é exibido na tabela 5.4.

Transação T_1	Transação T_1
$r_1(a)$	
	$r_2(c)$
$r_1(c)$	
	c = c – 1000
	$r_2(b)$
	b = b + 1000
	$w_2(c)$
$w_2(b)$	
	Commit
$r_1(b)$	
s = a + b+ c	
Commit	

Tabela 5.4. Falsa Atualização

A transação T_2 não altera a soma dos valores e também não viola a integridade. Contudo, no final da evolução da transação $\mathbf{T_1}$ a variável s, na qual contem a soma de a, b e c, encontra o valor de 1100. Isto é, $\mathbf{T_1}$ observa apenas os efeitos da transação $\mathbf{T_2}$, e observa também o estado que não satisfaz a integridade. Esta anomalia é conhecida como falsa atualização.

5.3 TAXONOMIA DO CONTROLE DE CONCORRÊNCIA

Segundo M. Tamer Özsu e Patrick Valdurez em [Özsu e Valdurez.1999], um dos critérios de classificação mais utilizados para determinar os mecanismos de controle de concorrência está no modo de distribuição do bancos de dados. Alguns dos algoritmos que vêm sendo propostos necessitam de uma completa replicação do bancos de dados, enquanto que outros podem operar com bancos de dados parcialmente replicados ou particionados.

O critério de classificação mais comumente utilizado é a primitiva de sincronização. O ponto de separação dos algoritmos de controle de concorrência resultam em duas classes [Özsu e Valdurez. 1999]: algoritmos que são baseados em exclusão mútua, isto é, acesso a dados compartilhados usando *locking*, e os que se baseiam em uma ordem de execução para as transações de acordo com um conjunto de regras (protocolos). Todavia, as primitivas podem ser utilizadas em algoritmos com diferentes visões: a visão pessimista, na qual muitas transações entram em conflitos; a visão otimista, na qual não haverá muitas transações em conflito.

As visões pessimista e otimista geram duas classes de mecanismos de controle de concorrência, como mostrado na figura 5.2.

Os algoritmos pessimistas sincronizam a execução concorrente das transações, no começo do seu ciclo de vida, enquanto que os algoritmos otimistas postergam a sincronização da transação até seu final.

O grupo de algoritmos pessimistas consiste de algoritmos baseados em *locking* (bloqueio), algoritmos baseados em *ordering* (transações ordenadas) e algoritmos híbridos.

O grupo de algoritmos otimista, pode, de forma similar ao pessimista, ser classificado como *locking* ou baseado em *timestamp ordering*.

Em algoritmos baseados em *locking*, a sincronização das transações é realizada empregando-se *locks* físicos ou lógicos, em algumas porções ou grânulos do bancos de dados, usualmente chamados de *locking granulariry* [Beeri e Bernstein. 1989]. Esta classe de algoritmo subdivide-se de acordo com as atividades do gerenciamento dos *Locks* que são realizadas:

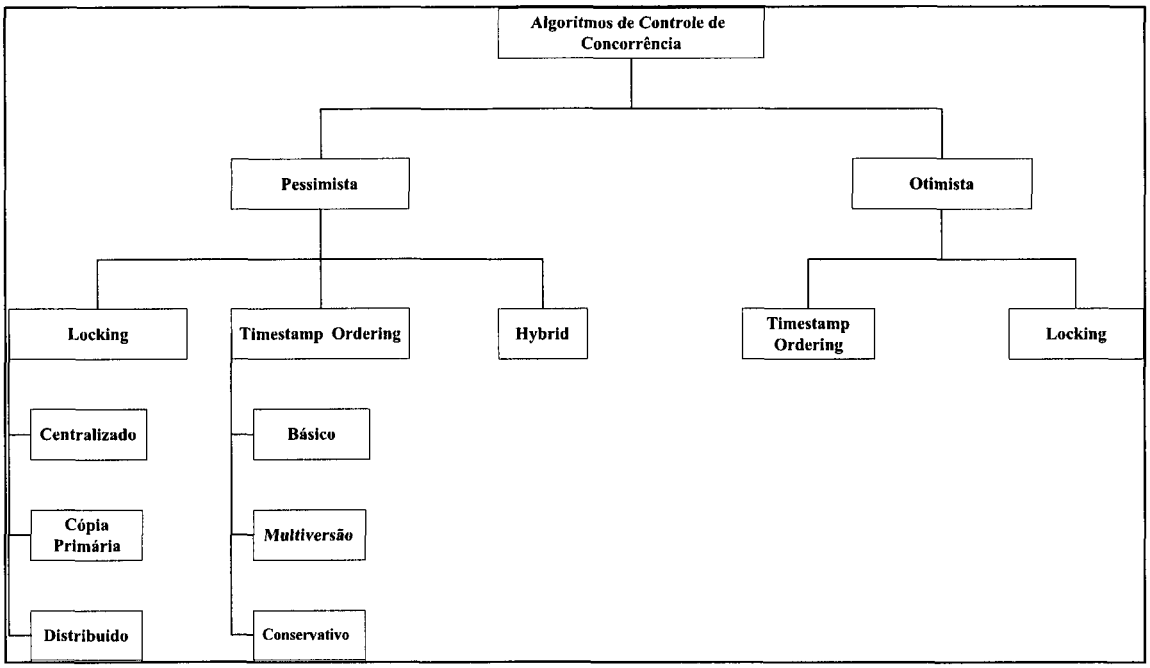

Figura 5.2. Classificação dos algoritmos de controle de concorrência

1.*Locking* Centralizado: um nó da rede é designado como nó primário, onde as tabelas dos locks para todo o bancos de dados são armazenadas, com a responsabilidade de garantir os *locks* das transações.

2.*Locking* em cópia primária (podendo ser em uma das cópias, quando há múltiplas cópias): Para cada unidade de lock é designada uma cópia primária, a qual será bloqueada com o propósito de acesso a esta unidade particular. Um exemplo desta situação pode ser apresentado: Se for feito um lock da unidade X e este for replicado pelos nós 1, 2 e 3, um desses nós, que pode ser o nó 1, é selecionado como cópia primária para X. Todas as transações que desejarem acessar X obtêm seus locks no nó 1, antes de poderem acessar a cópia de X. Se o bancos de dados não for replicado (isto é, há somente uma cópia de cada unidade bloqueada), o mecanismo de lock da cópia primária distribui a responsabilidade do gerenciamento do lock sobre os nós.

3. ***Locking*** **Descentralizado:** O gerenciamento do lock obrigatório é compartilhado por todos os nós da rede. Neste caso, a execução da transação envolve a participação e coordenação de *scheduler* em mais de um nó. Cada *scheduler* local é responsável pelo lock da unidades locais.

A classe de algoritmos *Timestamp Ordering* (TSO) inclui organização da ordem de execução das transações e dessa forma mantêm a interconsistência. Esta ordem é mantida pela determinação do *timestamp* para ambos, transação e item de dados que são armazenados no bancos de dados.

Os algoritmos híbridos são obtidos através da utilização em conjunto dos algoritmos baseados em *lock* e os *timestamp*. O algoritmo híbrido é um recurso utilizado para melhorar a eficiência a nível concorrência [Özsu e Valdurez. 1999].

5.4 TEORIA DO CONTROLE DE CONCORRÊNCIA

O principal objetivo em um projeto de algoritmos para controle de concorrência (CC) é corrigir o processamento de transações que estejam em conflito [Bhargava. 1999].

Cada transação tem um conjunto de operações de leitura e um conjunto de operações de escrita; se esses conjuntos de escritas e leituras interceptarem-se ocorrerá um conflito, como é mostrado no gráfico da figura 5.3.

Sendo CL o conjunto de operações de leituras e CE o conjunto de operações de escritas realizadas pela transações T_1 e T_2, em uma entidade comum dos bancos de dados, diz-se que o conjunto de leitura de T_1 conflita com o conjunto de escritas de T_2, representada pelas diagonais na figura 5.3. Não necessariamente há problemas de conflito entre o conjunto de operações de leitura de duas transações, pois a ação de ler não altera valores nas entidades dos bancos de dados.

Pode-se notar que haverá conflito se as duas transações forem executadas ao mesmo tempo. Por exemplo: T_1 foi finalizada depois de T_2 ser submetida ao sistema, logo ocorrerá uma interseção entre o conjunto de leitura e escrita, mas neste caso não haverá conflito, pois apenas uma das operações (escrita) causam alterações na entidade, visto que não ocorreram no mesmo instante.

Uma questão importante, a ser lembrada, é a serialização de transações concorrentes em bancos de dados distribuídos, pois a dificuldade desta operação é muito maior em bases distribuídas do que nas centralizadas. Isto porque os bancos de dados distribuídos estão operando sobre uma rede de computadores espalhados em áreas diferentes e há necessidade de envio e recepção de mensagens durante o processamento das transações; por essas razões, os mecanismos de controle de concorrência distribuído deve ter, entre outras coisas:

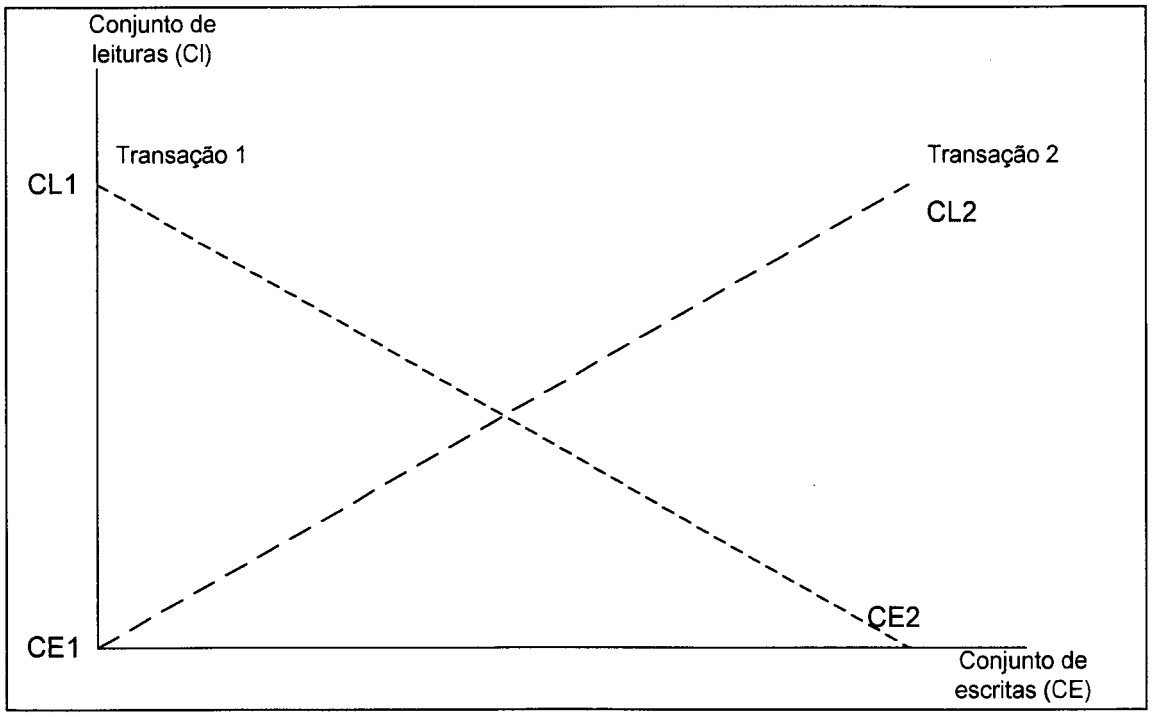

Figura 5.3. Gráfico exibindo o conflito de duas transações

✓**Capacidade de resolver conflitos independentes**: todo nó participante do sistema, deve ser capaz de resolver os conflitos com o mínimo ou nenhuma ajuda dos outros nós onde a transação concorrente tenha passado.

✓**Livre de *deadlocks***: em sistemas distribuídos, dois tipos de *deadlocks* podem ocorrer. Um é o *deadlock* local, o qual é restrito a um nó do sistema, o outro é o *deadlock* global, o qual envolve mais de um nó do sistema. A detecção e solução do *deadlock* local é mais simples para resolver, pois é quase inexpressivo para o sistema. O

problema está na detecção e solução do *deadlock* global, pois este ocorre entre nós do sistema e, dependendo da quantidade de nós envolvidos, pode levar a um colapso todo o bancos de dados.

✓ **Robustez**: Os mecanismos de controle de concorrência devem ser robustos, isto é, as operações de comprometimento (commit) devem ter boa performance não terem muito custo, isto é, não devem requerer muitas mensagens.

Com objetivo de resolver os problemas causados pelos conflitos de transações, foram propostas por Bharat Bhargava [Bhargava. 1999] três abordagens que podem ser usadas para projetar algoritmos de controle de concorrência:

✓ *Wait:* se duas transações entram em conflito, as ações conflitantes de uma das transações deve esperar até que a ação da outra transação seja finalizada.

✓ ***TimeStamp***: a ordem na qual as transações são executadas é selecionada baseada em uma marca (*stamp*) de tempo. Cada transação é assinalada pelo sistema por uma única marca de tempo (*timestamp*). Quando uma ação de uma transação conflita com outra ação de outra transação, a ordem de execução será determinada pelo *timestamp* designado inicialmente pelo sistema. A marca de tempo pode assinalar início, meio e fim da execução.

✓ ***Roolback***: Se duas transações estão em conflito, e se algumas ações de uma das transações é desfeita, retornar ou reinicializar, é necessário realizar um *rollback*, ou seja, retornar ao estado inicial as transações. Esta abordagem é conhecida como Otimista, pois espera-se que poucas transações realizem um *rollback*.

5.4.1 ALGORITMOS BASEADOS EM MECANISMO DE ESPERA (*WAIT*)

Quando duas transações são executadas ao mesmo tempo e entram em conflito, uma solução pode ser: uma das transações deve esperar que a outra libere a entidade (ou recurso) comum necessária a ambas. Para implementar esta solução, o sistema pode produzir um bloqueio (*lock*) na entidade do bancos de dados.

Quando uma transação quer realizar uma operação de escrita ou leitura, deve

esperar até que o sistema garanta o bloqueio da entidade necessária para a execução daquela operação. Para reduzir o tempo de espera, quando uma transação quer realizar uma das operações citadas, pode-se empregar os seguintes bloqueios:

✓***Readlock*** **(Bloqueio de Leitura)**: a transação bloqueia a entidade em modo compartilhado. Qualquer outra transação que esteja esperando para ler a mesma entidade, pode também obter o readlock.

✓***Writelock*** **(Bloqueio de Escrita)**: a transação bloqueia a entidade em modo exclusivo. Se uma transação quiser escrever em uma entidade, nenhuma outra entidade, naquele momento, poderá obter um *readlock* ou um *writelock.*

Quando uma transação finaliza uma operação em uma entidade, a transação pode realizar a operação de desbloqueio (*unlock*), após a qual outros tipos de bloqueios são permitidos e a entidade pode ser utilizada por outras transações que estavam esperando liberação.

Um aspecto importante é que as operações de bloqueio e desbloqueio podem ser embutidas em uma transação pelo usuário ou serem transparentes para a transação. Há casos em que o sistema assume a responsabilidade, com objetivo de garantir a exatidão, de forçar o bloqueio e o desbloqueio das operações de cada transação.

O bloqueio em entidades/objetos acarreta dois problemas: *livelock* e o *deadlock.* O livelock ocorre quando uma transação repetidamente falha na obtenção de um bloqueio (*lock*); o *deadlock* ocorre quando várias transações tentam bloquear várias entidades simultaneamente[Makki e Pissinou. 1995]: uma transação X bloqueia uma entidade A, que está sendo esperada pela transação Y, mas a transação Y está com a entidade B bloqueada, porém a entidade B está sendo esperada pela transação X; como não há liberação das entidades pelas transações, uma transação fica indefinidamente esperando a liberação da entidade pela outra, causando um colapso no sistema e, consequentemente, sua paralisação.

O problema do *deadlock* pode ser resolvido com as seguintes abordagens [Bhargava. 1999], entre outras:

✓ cada transação bloqueia todas as entidades de um vez. Se algum desses referidos bloqueios for mantido por outra transação, então esta transação libera-o para que seja obtido.

✓ assinalar uma ordem linear arbitrária para os itens, e solicitar que todas as transações requisitem os bloqueios nesta ordem.

Desde que o critério para o processamento concorrente de várias transações seja a seriabilização, o bloqueio pode ser feito corretamente para assegurar as propriedades anteriores. Um simples protocolo que toda transação pode acatar para resultar na seriabilização é o protocolo Bloqueio de duas fazes (***Two-phase locking* – 2PL**).

O protocolo simplesmente exige que, nas transação, todos os *locks* sejam precedidos de um *unlock*. A transação opera em duas fazes: na primeira fase realiza-se o *locking*, e na segunda realiza-se o *unlocking*.

A primeira fase pode ser considerada a fase de crescimento, na qual a transação obtém mais e mais *locks* sem liberá-los. Se a transação liberar algum *lock*, então entra na fase de diminuição (*shrinking*).

Na fase de *shrink* a transação somente libera *locks*, sendo proibido obter *locks* adicionais. Quando a transação termina, todos os locks restantes, que não foram liberados, serão liberados automaticamente. Neste caso, antes da liberação do primeiro *lock*, este ponto é chamado de lockpoint; após o lockpoint entra-se na fase de *shrink*. Na figura 5.4 é ilustrado o 2PL com o *lock point*.

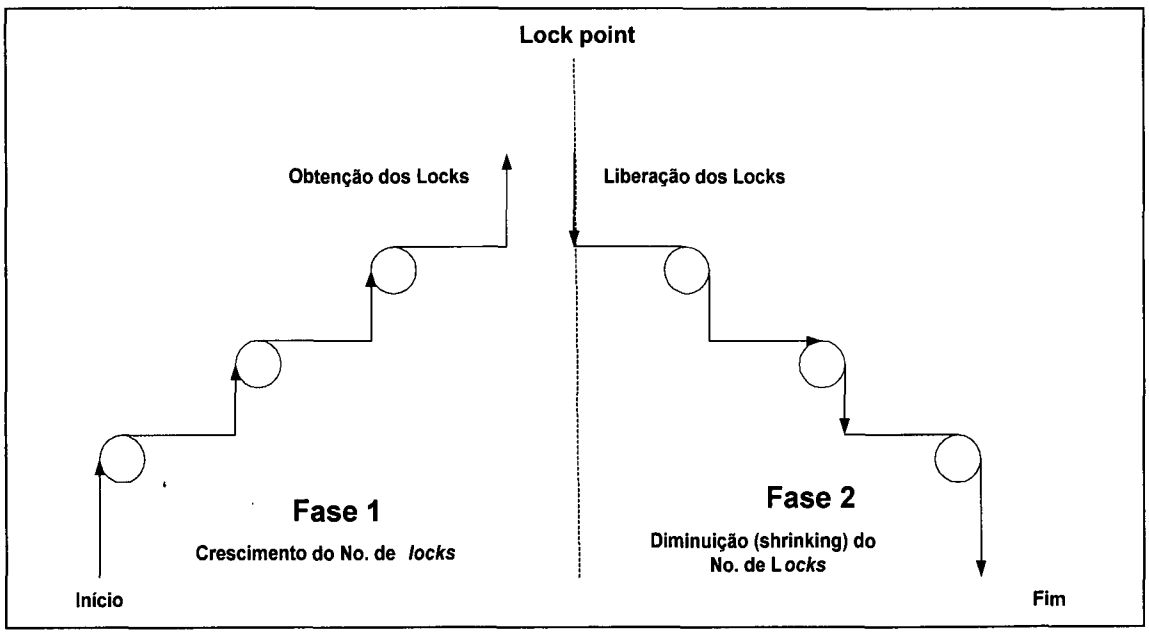

Figura 5.4. Bloqueio (*Locking*) de duas Fases
Fase de crescimento - ponto (lock point) de mudança – fase de diminuição

Em sistema reais, ter uma base de dados totalmente replicada torna o sistema muito ineficiente, na prática, quando os bancos de dados são replicados, realizam mais operações de leitura [Thomasin. 1998].

A maior razão para o estudo do controle de concorrência está na realização de operações de escrita em um bancos de dados com múltiplas cópias, pois é neste tipo de ambiente que podem ocorrer as anomalias citadas na seção 5.2 deste capítulo.

A seguir, será exibido um mecanismo simples para controle de concorrência, baseado em um algoritmo que utiliza bloqueios (*locks)* centralizados em um sistema de bancos de dados distribuído. Por motivos de simplicidade, considera-se que todas as transações de escritas dentro do bancos de dados e a própria base de dados seja totalmente replicada.

Algoritmo de controle de concorrência usando *locking* centralizado

Quando uma transação T_1 alcança o nó X de um sistema distribuído, os passos a seguir são realizados:

1. O nó X requisita do nó central o bloqueio para todas as entidades referenciadas na transação.

2. O nó central checa todas os bloqueios requisitados. Se alguma entidade é realmente bloqueada por outra transação, então a requisição é enfileirada. Há uma fila para cada entidade; a requisição espera em uma fila por vez.

3. Quando a transação obtém todos os locks, ela é executada no nó central, pois se ocorrer em outro nó ocorretará muita troca de mensagens. Então realizam-se as leituras necessárias, executa-se a computação relativa às leituras e, após todo esse processo, escrevem-se os resultados no bancos de dados do nó central.

4. Os valores obtidos (conjunto de dados da atualização realizada) no passo anterior são transmitidos para todos os outros nós, pois o bancos de dados é totalmente replicado.

5. Cada nó recebe os novos conjuntos de atualizações repassando-as para o bancos de dados local; então o sinal de recebido "OK" é enviado para o bancos de dados central, assinalando que a operação foi efetivada.

6. Quando o nó central recebe o sinal de "OK" de todos os bancos de dados locais participantes do sistema, isto significa que a transação T_1 foi complemente realizada em todos os nós. Após, o nó central libera os bloqueios (*locks*) e inicia o processamento da próxima transação.

Há algumas variações do algoritmo de *locking* centralizado, como as seguintes:

✓ **Bloqueio (*lock*) no nó Central - Execução em todos os nós**: Em vez da execução da transação ser no nó central, pode se designar o bloqueio (lock) no nó central e enviar a transação T_1 para o nó X. A transação T_1 é executada no nó X; logo, o conjunto de dados para leitura e escrita são obtidos do nó X. O nó X envia o conjunto de dados de atualização, obtendo o sinal de "OK" de todos os outros nós, então conclui-se que a transação T_1 foi finalizada. O nó X envia a mensagem para o desbloqueio (*unlock*) das entidades referenciadas por T_1. Após receber esta mensagem, o nó central libera os bloqueios (*locks*) e inicia a designação dos bloqueios para espera das transações.

✓ **Evitar reconhecimento – Designar uma seqüência de números**: No algoritmo de controle centralizado, o reconhecimento é necessário pelo nó central, para verificar se as atualizações foram efetivadas em todos os nós do bancos de dados. Mas é necessário que o nó central espere que esse reconhecimento aconteça, pois esta condição é suficiente para garantir que as atualizações foram realizadas em todos os nós e na mesma ordem em que foram realizadas no nó central. Para concluir a operação com êxito, o nó central pode designar um incremento monotônico na seqüência de números de cada transação. A seqüência numérica é adicionada ao conjunto de atualizações da transação, que é usada para ordenar a atualização dos novos valores dentro do bancos de dados de cada nó. Isto é, o nó central não precisa

esperar pelo aviso de reconhecimento enviado por todos os nós, garantindo a atualização. Após a execução dos passos acima, relativos à variação do algoritmo centralizado, a efetivação não precisa esperar pelo reconhecimento, mas o equivalente realizado com êxito, tornando o algoritmo mais eficiente.

- ✓ **Bloqueio Global de duas fases**: É uma simples variação do mecanismo de bloqueio centralizado. Em vez de obter todos os bloqueios para a transação no início e liberá-los no final, emprega-se a política do algoritmo 2PL. Cada transação obtém os bloqueios necessários para sua execução, e então libera os bloqueios que foram necessários por muito tempo, isto é, os primeiros bloqueios. A transação não poderá obter mais bloqueios depois que começar a liberá-los. Logo, se a transação necessitar de mais bloqueios, no futuro, deverá manter todos os locks presentes. A parte final deste algoritmo tem o mesmo comportamento do algoritmo visto anteriormente.

- ✓ **Bloqueio de Cópia Primária**: Nesta variação, em vez de selecionar o nó para o controlador central, uma cópia de cada entidade é designada como cópia primária da entidade. Cada transação deve obter o bloqueio na cópia primária de todas as entidades referenciadas. Em qualquer momento, nesta variação do algoritmo, a cópia primária tem o valor mais atualizado dos dados.

5.4.2 ALGORITMOS BASEADOS EM MECANISMO DE *TIMESTAMP*

No mecanismo de marcadores de tempo (*Timestamp*), a ordem da serialização é selecionada a priori. A execução das transações segue, obrigatoriamente, esta ordem preestabelecida.

No *timestamp ordering* (TSO), cada transação é assinalada com um único *timestamp* pelo escalonador do controlador de concorrência. Claro que, para se ter êxito, o *timesatamp* de cada transação deve alcançar diferentes nós do sistema distribuído e todos os clocks de todos os nós devem estar sincronizados, ou então terá que se escolher entre dois *timestamp*.

Lamport [1979] descreveu o algoritmo para sincronização de *clocks* distribuído via passagem de mensagens, onde as mensagens para alcançarem o nó local do nó remoto com *timestamp* alto, terão que assumir que o *clock* local é lento ou atrasado. O *clock* local é incrementado para o *timestamp* na mensagem recentemente recebida. Desta forma, todos os clocks são antecipados até serem sincronizados.

Um outro esquema, onde dois *timestamp* idênticos não podem ser designados para duas transações, cada nó designa um *timesatamp* para somente uma transação e para cada toque do relógio. Em conseqüência, o clock local é armazenado em *bits* de alta ordem e o nós identificados são diferentes. Este procedimento assegura um único *timestamp*.

Quando as operações de duas transações conflitam, elas são requeridas para serem processadas na ordem do marcador de tempo. Pode-se ter algumas variações desta abordagem (*TimeStamp*). Estas variações foram proposta no trabalho de Thomas [1979]:

- ✓ **Ordenação dos marcadores de tempo por classes de transações**: Nesta abordagem, assume-se que o conjunto de leituras e de escritas é conhecido antecipadamente. Esta informação é usada para agrupar as transações dentro de classes pré-definidas. A classe de uma transação é definida pelos conjuntos de leitura e de escrita. A transação T é um membro da classe C, se o conjunto de leitura de T é um subconjunto do conjunto de leitura de C e o conjunto de escrita de T também seja subconjunto do conjunto de escritas de C. A definição da classe é usada para produzir controle de concorrência. Este mecanismo foi usado no desenvolvimento do sistema SDD-1 [Rothnie et al. 1980].

- ✓ **Algoritmo de Eleição Distribuída**: Este algoritmo usa controle distribuído para decidir qual transação pode ser aceita e executada. Os nós do sistema de bancos de dados distribuído, comunicam-se entre eles e votam em cada transação. Se a transação obtiver a maioria dos votos, ela é aceita para execução e conclusão. A transação pode ser rejeitada por maioria dos votos, neste caso, deve ser reinicializada. Os nós podem defender ou postergar a votação para uma determinada transação. Os marcadores de tempo

(*timesatamp*) são mantidos nas entidades dos bancos de dados, e significam o tempo em que uma transação foi atualizada.

5.4.3 ALGORITMOS BASEADOS EM MECANISMO DE *ROLLBACK*

Nas abordagens realizadas anteriormente (mecanismos de espera e o de marcador de tempo), os algoritmos utilizam o bloqueio (lock) e o mecanismo de espera (*wait*). São, ambas, abordagens de controle de concorrência pessimista, pois subentende-se que todas as transações realizem um *lock/wait*, enquanto que na abordagem otimista, espera-se que poucas transações realizem um *rollback*.

A família de algoritmos de controle de concorrência otimista ou de não bloqueio é apresentada por H. T. Kung e J. T. Robison [Kung e Robison.1981]. Nesta abordagem, a idéia é validar uma transação junto com um conjunto de transações previamente comprometidas. Se a validação falhar, o conjunto de leituras da transação é atualizado e a transação repete sua computação e novamente tenta sua validação.

A fase de validação usará conflitos entre o conjunto de leituras e o conjunto de escritas juntamente com certas informações de marcadores de tempo. O procedimento de validação inicia quando uma transação finaliza sua execução sob o propósito otimista, e que outras transações não entrem em conflito com a mesma.

A abordagem otimista maximiza a utilização de informação sintática e tenta fazer uso da mesma semântica de informação sobre cada transação. Maximizar a informação e disponibilizá-la quando a transação finalizou seu processamento.

O controlador de concorrência pode tomar decisões a respeito de qual transação deve ser "abortada" enquanto outras transações podem ser processadas. Esta decisão pode ser tomada no momento da chegada da transação, durante a execução e até no final do processamento. Decisões tomadas no momento da chegada tendem a ser pessimistas e decisões tomadas no final podem invalidar o processamento da transação e necessitar um *rollback* [Bhargava. 1987]. Os efeitos da transação são mantidos em um espaço privado, não sendo conhecidos por outras transações, até que o controlador de concorrência garanta seu êxito.

Um projeto pode empregar mecanismos de controle de concorrência que empreguem o máximo de informação a um custo de reinicializar (fazer um *rollback*)

algumas transações. A extensão da reinicialização pode ser proporcional ao grau de conflitos sobre transações concorrentes.

Há quatro fases na execução de uma transação, na abordagem otimista do controle de concorrência:

1. **Leia**: desde de que a leitura de um valor de uma entidade não cause a perda da integridade, as leituras são completamente irrestritas. A transação lê os valores do conjunto de entidades e designa-as para o conjunto de variáveis locais. Os nomes das variáveis locais têm correspondência biunívoca com os nomes das entidades dos bancos de dados, mas o valor das variáveis locais é uma instância do estado anterior do bancos de dados, sendo apenas conhecida pela transação. Desde que os valores lidos pela transação podem ser alterados por uma operação de escrita de outra transação, fazendo a leitura do valor incorreto, a conjunto de leitura é submetido para validação.

2. **Computa:** A transação computa o conjunto de valores para entidade de dados chamadas de conjunto de escrita. Esses valores são designados para um conjunto de variáveis locais. Desse modo, todos escrevem, após a computação acontecer, em uma cópia da transação das entidades do bancos de dados.

3. **Valida:** O controlador de concorrência pode usar informações semânticas, informações sintáticas ou a combinação das duas. Neste trabalho será discutido o uso de informações sintáticas no contexto de validação e em apenas um nó [Bhargava. 1987]. No trabalho de Christos H Papadimitriou [Papadimitriou.1982], foi mostrado que, quando apenas a informação sintática é disponível para o controlador de concorrência, a serialização é a melhor forma de concluir com êxito, sem os critérios de precisão. Esta fase inicia após a transação ter passado e finalizado a fase de computação. A transação que entra na fase de validação antes que qualquer outra transação é automaticamente validada e comprometida (commited). Isto ocorre porque, inicialmente, o conjunto de transações comprometidas está vazio, logo, esta transação escreverá valores atualizados em um bancos de dados. Desde que

esta transação talvez seja exigida para validar junto com futuras transações, uma cópia do conjunto de leituras e escritas é mantida para o sistema. Qualquer transação que entra na fase de validação, valida junto o conjunto de transações comprometidas que foram concorrentes com ela. Com uma extensão, o procedimento de validação pode incluir validação junto com outras transações durante a fase de validação.

Condições para Validação: Não há um ciclos no grafo (Ciclo, em um grafo, é uma cadeia simples e fechada [Boaventura Netto. 1996]) de transações comprometidas (*commited transactions*) porque eles são serializáveis (ver figura 3.2 do capítulo 3). Se a validação de uma transação cria ciclos em um grafo, esta deve reinicializar ou realizar um *rollback*. Caso contrário, é incluída em um conjunto de transações comprometidas. A validação de uma transação está em uma seção crítica, isto é, o conjunto de transações comprometidas não serão alteradas, enquanto a transação é ativamente validada.

No caso da transação falhar na validação, o controlador de concorrência pode reinicializar a transação desde o início até a fase de leitura. Isto ocorre porque a falta de validação têm como conseqüência a leitura de um conjunto incorreto de transações falhas, pois a falha na validação faz com que o conjunto de leitura, da transação que falhou seja incorreto. Desde que o conjunto de escritas das transações comprometidas encontre o conjunto de leituras da transação que falhou, durante a validação, é talvez possível atualizar os valores do conjunto de leituras da transação, no momento da validação. É possível, na transação que falhou, dar início à fase de computação antes da fase de leitura.

4. **Compromete e escreve** (***Commit e Write***): Se a transação obtiver sucesso na validação, é considerada comprometida (*commited*) no sistema, sendo designado um marcador de tempo (*timestamp*). Caso contrário, a transação é desfeita (*rollback*) ou reinicializada até outra fase de computação ou fase de leitura. Se a transação tem sucesso na fase de validação, então o conjunto de

escrita é feito global e os valores do conjunto de escrita tornam-se valores das entidades no bancos de dados de cada nó.

Todas as fases do processamento corrente de transações são intercalado, mas a fase de leitura deve preceder a fase de computação e validação [Bernstein e Goodman. 1984].

5.5 CONSIDERAÇÕES FINAIS

Foram discutidas a arquitetura do controle de concorrência e as principais anomalias das transações concorrentes. Apresentou-se a taxonomia dos algoritmos de controle de concorrência proposto por M. Tamer Özsu e Patrick Valdurez em [Özsu e Valdurez.1999] e a abordagem proposta por Bharat Bhargava [Bhargava. 1999]. A primeira (proposta por Özsu e Valdurez) é baseada na forma como os dados estão distribuídos; a segunda (proposta por Bhargava) é baseada em três algoritmos que são *wait*, *timesatamp* e roolback.

Os algoritmos de controle de concorrência distribuída discutidos garantem as propriedades de isolamento e consistência das transações. Os mecanismos de controle de concorrência de um SGBD, asseguram que a consistência do bancos de dados seja mantida. Por essa razão, o controlador de concorrência é um dos componentes principais do SGBD distribuído.

Todos os mecanismos apresentados, apenas alguns ou a combinação de dois ou mais podem resultar em um mecanismo confiável para bancos de dados distribuídos heterogêneos.

CAPÍTULO 6

MACANISMOS DE CONTROLE DE CONCORRÊNCIA PARA BANCOS DE DADOS DISTRIBUÍDOS HETEROGÊNEOS

6.0 INTRODUÇÃO

Controle de concorrência, em sistemas de bancos de dados, tem sido foco de mais de 20 ano de pesquisa. Os problemas e soluções foram proposto por Christos H. Papadimitriuo [Papadimitriou. 1979]. Os bancos de dados têm a característica de poderem ser interconectados, resultando em um sistema de bancos de dados heterogêneo, onde cada nó da rede de computadores, de que este sistema faz parte, pode usar diferentes estratégias para o controle de concorrência. A figura 6.1 exibe um sistema de bancos de dados heterogêneos.

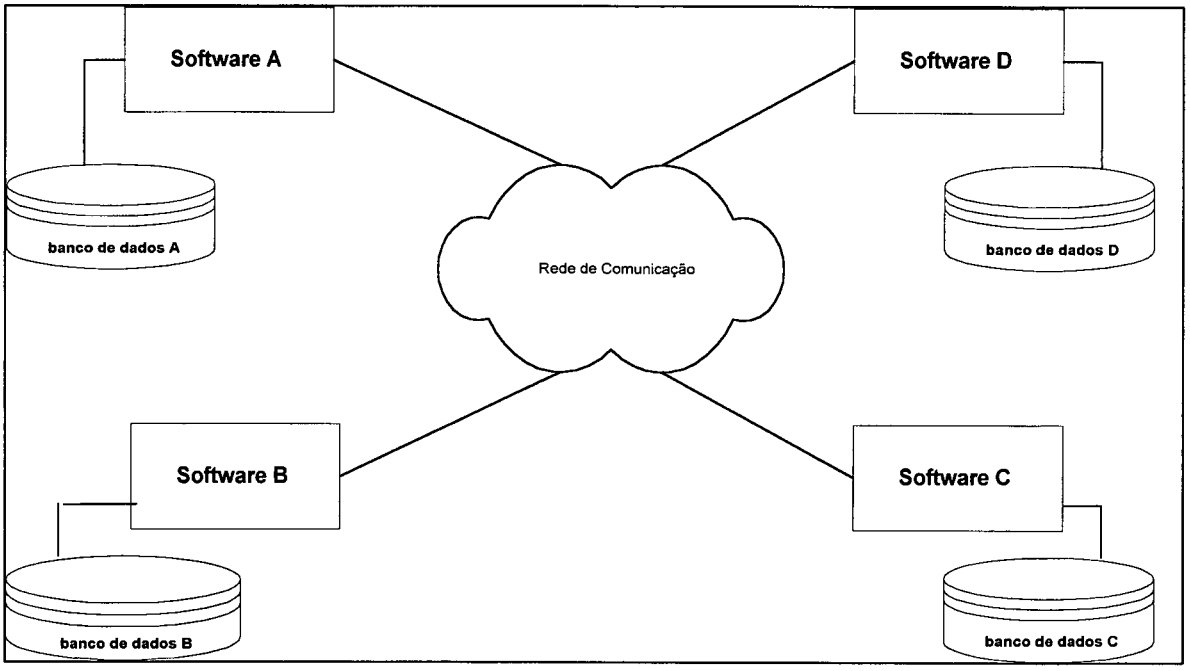

Figura 6.1. Sistema de bancos de dados distribuídos heterogêneos

Em sistemas de bancos de dados distribuídos, vários usuários podem ler e atualizar informações concorrentemente; nestas condições, podem ocorrer situações indesejáveis, se as operações das transações dos vários usuários são intercaladas de

forma imprópria. Para resolver esses problemas, os sistemas de bancos de dados possuem um módulo que é responsável pelo controle de concorrência das transações.

O controle de concorrência é um importante aspecto no processamento de transações distribuídas. Virtualmente, todos os sistemas gerenciadores de bancos de dados usam *Two-Phase Locking* (2LP) para sincronizar o acesso aos bancos de dados [Thomasin. 1998]. Desde de que foram descritos por H. T. Kung e J. T. Robison [Kung e Robison 1981], os métodos de controle de concorrência, tanto para bancos de dados centralizados como distribuído, vêm sendo implementados em vários protótipos, principalmente em ambientes distribuídos heterogêneos.

Projetar estratégias/mecanismos de controle de concorrência para bancos de dados heterogêneo é mais difícil do que para ambiente de bancos de dados homogêneos, pois deve-se negociar não apenas a distribuição dos dados, mas também a heterogeneidade e a autonomia implícitas nestes sistemas.

Neste capítulo serão apresentados mecanismos para controle de concorrência específicos para bancos de dados distribuídos heterogêneos. Será discutida a dinâmica do controle de concorrência em bancos de dados múltiplos e o proposto por San-Yih Hwang [Hwang et al.1993] para bancos de dados federados.

Outro método a ser discutido é o método híbrido de controle de concorrência otimista com alta performance no processamento de transações. Este método foi proposto por A. Thomasin [Thomasin. 1998], sendo este, atualmente, o mais utilizado comercialmente por produtos como INGRES, INFORMIX [Stonebraker e Hellerstein. 1998] entre outros, em função do seu alto desempenho no processamento de transações, o que faz com que as atualizações sejam confiáveis e produzam estados de consistências nos bancos de dados envolvidos.

6.1 Dinâmica do Controlador de Concorrência nos BDDHs

Em sistemas de bancos de dados fortemente acoplados, há somente um controlador de concorrência que coordena, certifica e/ou produz escalonamentos. O controlador de concorrência tem acesso a todas as informações necessárias para ordenar e produzir e/ou certificar os escalonamentos; normalmente tem o controle sobre todas as transações que estão sendo executadas no sistema. Os sistemas de bancos de dados

múltiplos devem negociar os problemas causados pela autonomia dos múltiplos sistemas locais.

A princípio, os controladores de concorrência locais são designados de uma forma que desconhecem totalmente o local dos outros sistemas de bancos de dados, além de desconhecer o processo de integração (autonomia de projeto). Em segundo lugar, o controlador de concorrência global precisa de informações sobre a ordem das execuções locais, para manter a consistência global do bancos de dados. De qualquer modo, o controlador de concorrência geral não tem acesso direto às informações e não pode forçar o controlador de concorrência local a fornecê-las (autonomia de comunicação).

Outro problema é que os controladores de concorrência local podem decidir sobre o comprometimento (*commitments*) de transações, baseados inteiramente nas suas próprias considerações. Os controladores de concorrência locais não sabem ou não se preocupam com o comprometimento de uma determinada transação, introduzindo, desta forma, inconsistência no bancos de dados global.

Um Controlador de concorrência global não têm o controle de todos os controladores de concorrência locais. O controlador de concorrência global não pode forçar um controlador de concorrência local a reinicializar uma transação, igualando o comprometimento (*commitment*) das transações locais, pois isto introduzirá inconsistência no bancos de dados global (autonomia de execução).

6.2 Controle de Concorrência em SBDM

Para iniciar a discussão do controle de concorrência em bancos de dados múltiplos, deve-se considerar a arquitetura da figura 6.2. Os componentes da arquitetura são descritos a seguir:

SBDL - Sistema bancos de dados local: São bancos de dados preexistentes, geralmente heterogêneos. Há dois níveis de heterogeneidade:

1º nível - heterogeneidade a nível de sistema, onde SGBDLs suportam diferentes modelos de dados, empregam-se diferentes estratégias de otimização de consultas e diferentes protocolos de controle de concorrência.

2º nível – heterogeneidade a nível de dados, onde informações podem ser representadas de forma diferente, isto é, explicitamente os itens de dados são de um SBDL ou implicitamente como uma relação entre dois itens de dados em outro SBDL.

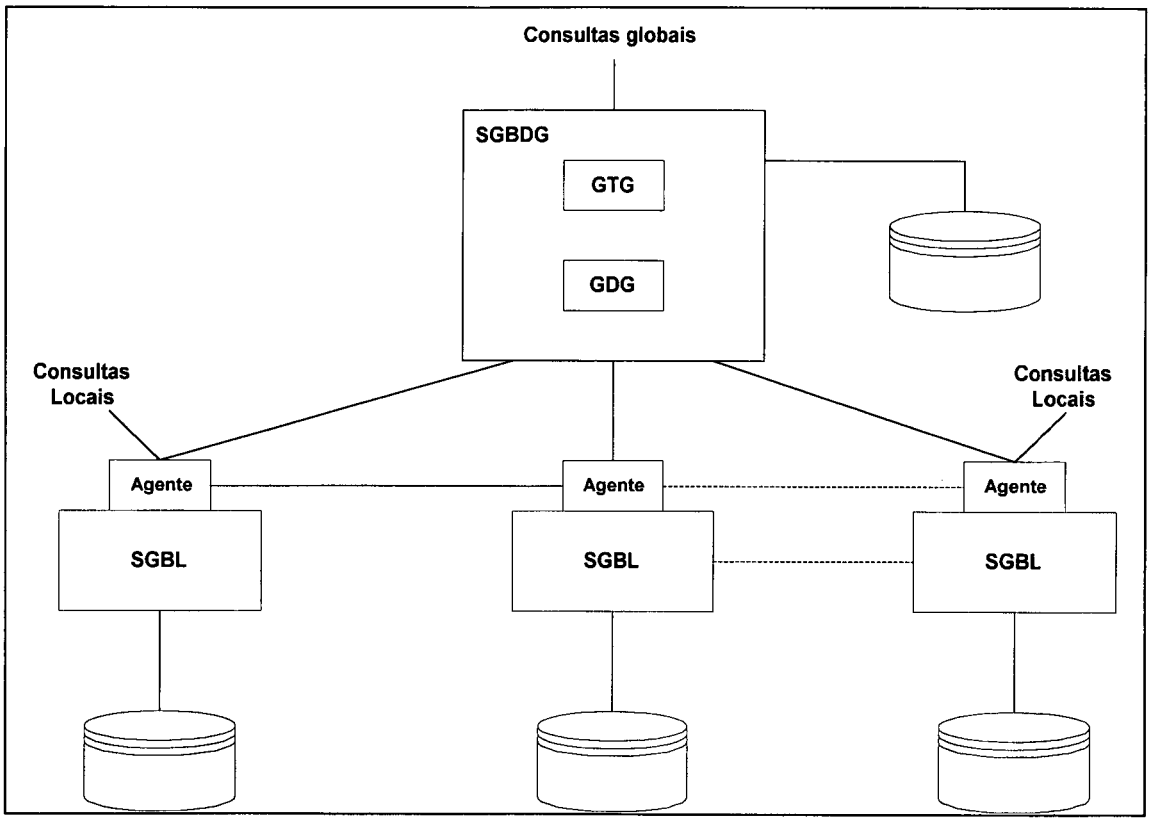

Figura 6.2 Modelo de sistema de bancos de dados múltiplos

SGBDG – Sistema gerenciador de bancos de dados global – Geralmente tem recursos limitados. Isto pode ser devido à necessidade de acesso a outro sistema de informação ou sistema operacional de outros SGBDL que são essenciais para o SGBDG executar as consultas globais de forma correta e eficiente.

Consultas Locais: Podem ser independentemente submetidas para SGBDL sem informar ao SGBDG. As consultas locais podem interagir com subconsultas submetidas pelo SGBDG e, de alguma forma permitidas pelo SGBDL, fazendo a coordenação a execução de subconsultas.

6.2.1 PROCESSAMENTO DA TRANSAÇÃO EM SBDM

Os sistemas gerenciadores de bancos de dados globais –SGBDG consistem de dois componentes: um gerenciador de dados globais (GDG) e um gerenciador de transações globais. Uma transação é primeiramente decomposta em subtransações pelo gerenciador de dados globais – GDG, de acordo com a disponibilidade dos dados.

O gerenciador de dados globais – GDG, junto com o gerenciador de bancos de dados locais – GBDL, é também responsável por checar autorizações, o qual impede acessos ilegais no bancos de dados local.

O gerenciador de transações locais - GTL é responsável por submeter e executar subtransações globais. Para preservar a autonomia local, um agente é adicionado no topo de cada sistema gerenciador de bancos de dados local.

Um agente é uma interface entre o SGBDG e um SGBDL e controla a execução de subtransações globais nos nós.

O controlador de concorrência global – CCG é um componente do GTG que coordena e executa subtransações globais em diferentes nós, sendo que a consistência global é mantida.

A abordagem convencional para controle de concorrência executa as transações da mesma forma da serialização. A execução de um conjunto de transações é dita serializável se é equivalente a uma execução seqüencial de transações.

A dificuldade do controle de concorrência global em SGBDM é sobretudo devida à falta de informação sobre o controle das execuções locais.

No controle de concorrência global, há possibilidades de se aplicar tanto uma abordagem pessimista como uma otimista. Em geral, a abordagem pessimista evita possíveis inconsistências pelo atraso da submissão das subtransaçãoes globais, por essa razão resultando em uma baixa concorrência.

A abordagem otimista resolve interações indesejáveis posteriormente, por isso pode haver abortos de muitas transações. Outra importante questão do controle de concorrência é o conhecimento da execução pelo escalonador do sistema gerencioador de bancos de dados local. Do ponto de vista do gerenciador de transações globais, os cinco tipos a seguir de escalonamento são diferentes [Breitbart et al. 1990]:

✓ **Serializável:** um escalonamento é dito serializável quando possui uma seqüência de instruções de várias transações na qual as intruções petencentes a uma

única transação aparecem juntas.

✓ **Fortemente Serializável**: Um *schedule* (escalonamento) é fortemente serializável se é serializável e a ordem da serialização de qualquer de suas transações não sobrepostas é consistente com a sua atual ordem de execução.

✓ ***Sp-schedule*:** Um *sp-schedule* é uma execução fortemente serializável e a ordem da serialização de uma transação, pode ser determinada pela designação de uma operação da transação (lockpoint).

✓ **Fortemente Recuperável**: Um *schedule* é fortemente recuperável se um *sp-schedule* e uma transação não realizam um *commit* até que todas as transações, que estão em conflito com ela e a precedem na ordem da serialização, fazem um *commit*.

✓ **Rigoroso**: Um *schedule* é rigoroso se é fortemente recuperável em uma operação e será executado somente se não conflitar com nenhuma operação precedente de uma transação descomprometida (*uncommitted*).

Muitos dos protocolos de controle de concorrência que têm sido propostos geram *schedules* serializáveis (*two-fase locking*, time-stamp ordering, otimista e *serialization graph* [Hadzilacos. 1988]). A maior parte deles gera somente *sp-schedule*, execeto a abordagem de grafo de serialização, que pode ser facilmente modificada, gerando somente *schedules* fortemente recuperáveis [Fakete et al. 1990].

Os protocolos pessimista e otimista podem determinar que não haja violação da autonomia local. Por questões de performance, a autonomia local pode ser intencionalmente violada [Elmagarmid et al. 1999]. Há muitos protocolos propostos para ambiente heterogêneos de controle de concorrência globais; a seguir, os mesmos serão discutidos, dando-se ênfase à suposição de que eles fazem *schedules* locais e permitem controle de concorrência global. A discussão será dividida por abordagem.

6.2.2 ABORDAGEM PESSIMISTA

A abordagem pessimista confia nas propriedades dos escalonamentos (*schedules*) de subtransações globais dos SGBDLs, não somente para garantir a serialização, mas também para permitir alto grau de concorrência global. Em geral, a abordagem pessimista é menos confiável para SBDMs do que basicamente para os SGBDLs, pois estes garantem somente *schedules* estritamente locais ou, no mínimo, *schedule* fortemente serializável.

O mecanismos *site graph* [Breitbart e Silberschatz. 1988] e o *locking* altruísta [Larson e Sheth. 1990] são dois protocolos que apenas requerem execução local fortemente serializável. A idéia básica dos dois é evitar execução concorrente de duas transações globais em mais de um nó.

O *site graph* é similar ao protocolo grafo serializável, mas com alto grau de granularidade (nós e transações). Isto detecta possíveis conflitos a nível global e os resolve pelo atraso de uma ou mais transações globais.

O *locking* altruísta melhora a concorrência pela prevenção, duplicando cada nó, envolvido na execução das transações. Se duas transações globais TG_1 e TG_2 acessarem ambas, os nós N_1 e N_2, a transação TG_2 pode iniciar a subtransação no N_2 antes de TG_1 finalizar em N_1. Mas, se TG_2 acompanha TG_1 no nó N_1, esta deve também acompanhar TG_1 em N_2.

O protocolo *Top-down* assume que todas as execuções locais são *sp-schedule*s [Elmagarmid et al. 1999], e o controle de concorrência é melhorado, pois duas transações globais conflitantes não devem executar seqüencialmente no nó local como no *site graph* e no protocolo *locking* altruísta. Transações globais podem duplicar tanto quanto os pontos de serialização são executados em uma ordem consistente de todos os nós locais.

Execuções locais fortemente recuperáveis são apenas caso especial do *sp-schedule*, nas quais os pontos de serialização são operações simples de comprometimento (*commitment*). Por essa razão, todos os protocolos que trabalham com *sp-schedule*, como os *top-down*, também trabalham com execuções locais fortemente recuperáveis, isto é, com pontos de serialização fixos. Nota-se que isto não melhorará a concorrência, pois as operações de *commitment* são as últimas operações realizadas em cada transação.

O controle de concorrência global pode ter uma grande melhora, se todos os *schedules* locais forem rigorosos. Foi apresentado por Yuri Breitbart [Breitbart et al. 1991] um protocolo que assegura serialização global pelo controle do *commitment* das transações globais. Mais especificamente, o GTG não escalona o *commitment* de uma transação global até que todas as operações previamente tenham completado sua execução. A concorrência melhora, pois se regulam as operações que foram concorrentemente escalonadas.

6.2.3 ABORDAGEM OTIMISTA

A abordagem otimista tem como objetivo a detecção exata dos indesejáveis conflitos entre transações globais. A hipótese é que duas transações globais não conflitem, ainda que possam se igualar de forma geral. Por essa razão, os SGBDLs não utilizam a abordagem pessimista. Como conseqüência , a abordagem otimista é mais aceita para os casos gerais, onde poucas suposições podem ser feitas sobre as execuções locais.

Calton Pu, em 1988, apresentou um modelo otimista para super bancos de dados, cuja a idéia principal era: "*detectar inconsistências de schedules globais, por comparação da ordem de execução de pontos de serialização, para cada transação global*". Já o protocolo otimista *Ticket*, proposto por D. Georgakopoulos [Georgakopoulos et al. 1994], assegura o desenvolvimento de uma técnica que, efetivamente, obtém a ordem da serialização das transações globais em tempo de execução. A idéia principal é criar um item de dado especial (chamado de *ticket*) para cada nó local, em todas as transações globais. Primeiro é requerida a leitura do valor do *ticket* ao qual é incrementado antes de cada uso. A execução global é inconsistente se duas transações globais quaisquer lerem valores inconsistentes do ticket em mais de um nó.

O problema principal com o protocolo otimista *Ticket* é a introdução de conflitos diretos entre todos os pares de transações globais que acessam um nó comum, resultando em muitas transações abortadas. Introduzir conflitos é aparentemente necessário, pois o GTG pode não ser capaz de resolver inconsistências iguais, se detectadas. O exemplo abaixo apresenta de forma mais clara o problema:

Supondo-se que TG_1 e TG_2 são duas transações globais.

Supondo-se que ambas lêem os itens A e B do nó N_1.

Considerando-se que tg_1 e tg_2 são subtransações de TG_1 e TG_2 no nó N_1, respectivamente.

$TG_{1,0}$: $R_1(A)$, onde R é uma leitura

$TG_{2,0}$: $R_2(B)$

Se L é uma transação local concorrentemente executada com $TG_{1,0}$ e $TG_{2,0}$ no nó N_1.

L : $W_1(A)$ W1(B)

Considerando E_1 uma execução local de $TG_{1,0}$, $TG_{2,0}$ e L. Três ordens de serialização de $TG_{1,0}$ e $TG_{2,0}$ são possíveis:

Se E_1: $W_1(A)W_1(B)TG_{1,0}(A)$, $TG_{2,0}(B)$;

$TG_{1,0} \rightarrow TG_{2,0}$, se E_1: $R_1(A)W_1(A)W_1(B)R_2(B)$; ou

$TG_{2,0} \rightarrow TG_{1,0}$, se E_1: $W_1(A)R_1(A)R_2(B)W_1(B)$

TG_2 conflita com TG_1 antes da execução do $W_1(B)$. Se no conflito, ambos TG_2 e TG_1, foram comprometidos antes de $W_1(B)$, então há somente uma maneira para resolvê-lo: realizar o abortando da transação local L. Isto é, violará a propriedade de autonomia de execução.

6.3 Controle de Concorrência em Bancos de dados Federados

O problema do controle de concorrência em bancos de dados federados é especialmente dificultado devido à própria autonomia e heterogeneidade dos bancos de dados locais participantes.

Como solução para o problema de Controle de concorrência para BDFs, foi proposto por San-Yih Hwang [Hwang et al. 1993] um protocolo conhecido como DASGO (*Dynamic Adjustment of Global Serialization Order*). Este protocolo utiliza três mecanismos de controle de concorrência: ordem global pré-definida, controle de concorrência otimista e ajuste da ordem de serialização global.

Tal protocolo está sendo apresentado neste trabalho porque oferece alto grau de concorrência, grande capacidade para reduzir os desperdícios de recursos, além de possuir a capacidade de detectar precocemente abortos de eventuais transações

serializáveis não-globais. O DASGO basea-se em uma arquitetura de referência BDFs que será exibida na próxima seção. A apresentação desta arquitetura se faz necessária para que se esclareça como ocorre a execução das transações globais e locais e como o controle de concorrência, proposto pelo referido protocolo, atua.

6.3.1 ARQUITETURA DE REFERÊNCIA DE BDFS

Um sistema de bancos de dados federados é composto de um gerenciador de transações globais (*transaction global manager*)-GTM e um número de Agentes de Transações Globais (Global Transaction Agents)-GTA, um para cada sistema gerenciador de bancos de dados local (SGBDL). A figura 6.3 exibe a referida arquitetura.

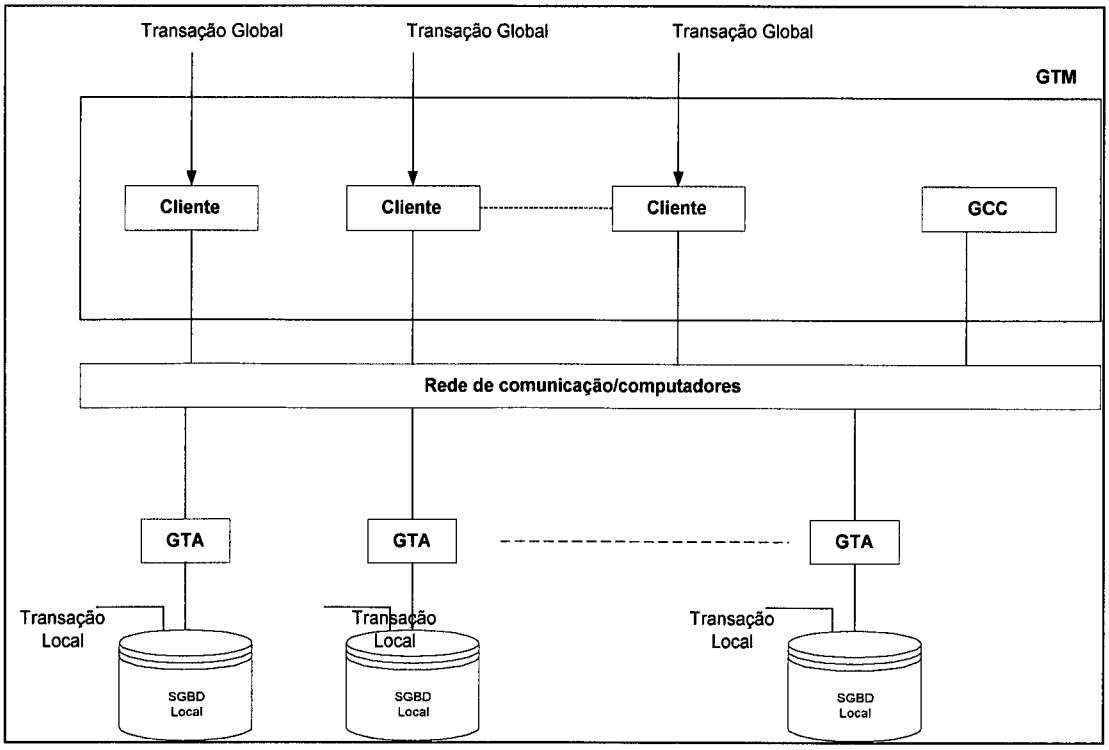

Figura 6.3. Arquitetura de referência do SGBDF

O GTM é uma unidade lógica que consiste do Controlador de Concorrência Global (global concurrency controller)-GCC e um número de clientes, um para cada nó da rede. Cada cliente é associado ao sistema do usuário final, enquanto que cada GTA é associado com um SGBD Local.

Devido à autonomia local, um GTA é tratado como um processo de usuário pelo SGBD local. O GCC é responsável pelo controle de concorrência de transações globais e manutenção da informação da execução da transação global. Os clientes e os GTAs consultam com o GCC para determinar a execução de uma transação global. Isto é, o GTM e todos os GTAs trabalham cooperativamente para garantir a execução serializável global.

Uma transação global, acessando um bancos de dados local, é modelada como uma seqüência de operações de leituras/escritas. Uma transação global chega ao estado de comprometimento (*commit*) se todas as suas subtransações são executadas com sucesso, nos nós locais.

Para a execução do algoritmo proposto pelo protocolo, é necessário fazer algumas considerações no ambiente de trabalho, como a seguir:

✓ Cada SGBD local deve garantir a serialização

✓ Um SGBD não pode distinguir entre transações globais e locais;

✓ Cada SGBD local permite um estado visível preparado para o comprometimento (prepared-to-commit) para as transações.

✓ A comunicação entre os nós do da rede é confiável.

6.3.2 Mecanismo do Controle de Concorrência Proposto - DASGO

A estratégia usada por este protocolo é, combinar os aspectos favoráveis de abordagens existentes, incorporando novos mecanismos para superar os problemas. A seguir serão discutidos os mecanismos utilizados:

1. **Mecanismo de ajuste dinâmico na ordem de serialização entre transações concorrentes:** melhora a performance do sistema reduzindo abortos desnecessários de transações, deste modo aumentando a utilização de todos os recursos. Neste mecanismo, a ordem global de serialização da subtransação no SGBD local é comparada com a ordem global das transações. A condição para o sucesso da validação da transação é que a

ordem da serialização local siga a ordem da global. Pelo tradicional mecanismo de marcador de tempo por ordem (*timestamp ordering*), ou marcador de tempo global, uma transação global é abortada se nenhuma das ordens de serialização local não fizeram comparação à ordem de serialização global. Contudo algumas transações não precisam ser abortadas, desde que a condição de validação seja suficiente, mas não necessária

2. **Ordem global pré-definida**: quando uma transação global inicia, é designada uma ordem global única, antes de ser submetida ao bancos de dados local. Com o mecanismo de ordem global pré-definido, a tarefa do controle de concorrência global é monitorar a ordem da serialização local na execução das subtransações globais, de cada SGBD local .

3. **Controle de Concorrência Otimista**: para aumentar o grau de concorrência, o DASGO emprega a abordagem otimista proposta por H. T. Kung e J. T. Robison [Kung e Robison.1981]. Uma transação global inicia a obtenção de sua ordem global e cada uma das suas subtransações é submetida para um SGBD local sem suspensões.

A execução da transação, neste protocolo, é descrita no diagrama de estados da figura 6.4.

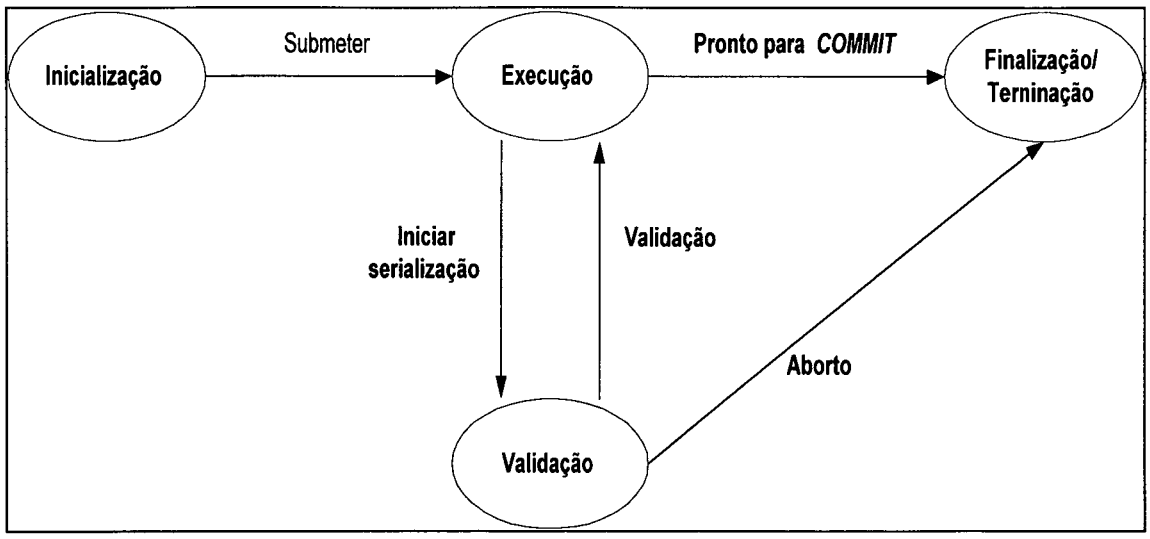

Figura 6.4. Diagrama de estados da execução de uma transação global – DASGO

A transação global pode estar em quatro estados:

1. **Inicialização**: Uma nova transação global obtém a ordem global. É então decomposta em subtransações, para cada uma das quais é designado um marcador de tempo (*timestamp*) para execução nos nós locais.

2. **Execução**: As subtrações globais são executadas nos SGBDs locais.

3. **Validação**: Uma transação global é validada pela comparação da ordem da serialização global contra a ordem de serialização local. Se dois tipos de ordens não conflitarem, então continua a execução; caso contrário é abortada ou a ordem da serialização da transação global é ajustada para igualar as ordens de serialização local.

4. **Final/Terminal**: Uma transação global termina apenas quando todas as subtransações são finalizadas ou se nenhuma das subtransações foi abortada.

6.3.3 Considerações sobre o Protocolo DASGO

O protocolo DASGO, emprega os três mecanismos básicos de controle de concorrência, isto é, Ordem Global Pré-Definida, Controle de Concorrência Otimista e Ajuste Dinâmico da Ordem das serializações Globais e utilizar a validação precoce. Este protocolo reduz o desperdício dos recursos do sistema, como também conclui com êxito a serialização global livre de *deadlocks*.

6.4 Método Híbrido de Controle de Concorrência

Este método combina o controle de concorrência otimista (OCC) com o *loking* Two-Phase, em sistema de bancos de dados distribuídos particionados. Os *locks* são apenas requisitados no momento da validação e processamento do *commit*, garantindo, dessa forma, a serialização global.

O protocolo otimista de controle de concorrência (OCC) permite alto grau de concorrência de transações e mostra-se superior ao 2PL, em sistemas com "infinitos" ou

pelo menos "adequados" recursos de hardware. Mais especificamente, a maior parte da grande vazão de transações, deve-se ao uso de OCC versus 2PL, às custas da capacidade do processamento adicional requerido para reinícios de transações. Outra vantagem do método OCC comparado com o 2PL, é que o primeiro é livre de *deadlock*, desde que o esquema de detecção do *deadlock* para SBDD tende a ser complexo e freqüentemente tem se mostrado incorreto. Entretanto, os *time-outs* podem ser utilizados para detecção de *deadlocks*. A sua implementação é complicada, devido a requerer a determinação de um intervalo de *time-out* apropriado. Esquemas de bloqueios livres de *deadlock*, tais como *wound-wait* e *wait-die* [Thomas. 1979] são alternativas. Esses métodos são superados quando o nível de contenção de dados e recursos de hardware adequados estão disponíveis. Outra abordagem para prevenir *deadlock* é limitar o tempo de espera de transações bloqueadas a 1. O método de controle de concorrência de espera limitada (*wait depth limit* – WDL) alcança esses objetivos, entretanto tenta minimizar o desperdício de processamento, devido aos reinícios de transações. O trabalho de P. Franaszek [Franaszek.1993] mostra que esse método também supera o 2PL em ambientes de construção de auto bloqueios.

Uma questão importante no CCO é a eficiência dos métodos de avaliação. Pode ser um método simples, que foi primeiramente proposto por H. T. Kung e J. T. Robison [Kung e Robison.1981] para sistemas centralizados. Causa um número desnecessariamente alto de reinicalizações, que pode ser prevenido, utilizando *timestamp* para detecção de conflitos. Há várias extensões do método original de avaliação para ambientes distribuídos. No caso de transações mais longas, esses métodos causam um número intoleravelmente alto de reinício e são suscetíveis a "inanição". Transações nunca podem tem sucesso por causa de repetidos reinícios.

Para evitar os problemas relatados anteriormente, alguns autores como G. Lausen. [Lausen.1982] e G. Le Lann [Le Lann. 1980] propuseram a combinação do método de bloqueio e do CCO, onde as transações podem ser sincronizadas tanto otimisticamente como pessimisticamente. Apesar de que este seja um passo na direção correta, o método resultante já não é livre de *deadlock.*

O modelo de controle de concorrência otimista proposto oferece benefícios substanciais sobre os métodos de controle de concorrência padrão e podem ser usados para aumentar a performance do processamento de transações em sistemas de dados

particionados ou não compartilhados. O protocolo pode ser descrito utilizando as seguintes características.

1. As transações são executadas de forma otimista, ignorando bloqueios mantidos por outras transações no objeto. Neste caso, foi investigado um método alternativo, onde o acesso a um objeto é adiado se há atualizações pendentes.

2. Antes que a validação global seja realizada, as transações a serem validadas solicitam ou requerem os bloqueios apropriados, para todos os itens acessados. Os bloqueios são somente mantidos durante a fase de *commit* (se a validação é bem sucedida), de forma que os conflitos de bloqueio são menos prováveis que no bloqueio padrão.

3. Se a validação falhar, todos o bloqueios adquiridos são retidos pela transação, enquanto é executada novamente. Isto garante que a execução da segunda fase tenha sucesso, se forem referenciados novos objetos. Desta forma, reinícios freqüentemente, assim como "inanição", podem ser previstos. Investigou-se também um método alternativo em que a requisição de bloqueios pudesse ser adiadas até um ponto, onde a execução de uma transação é considerada bem sucedida.

4. *Deadlocks* em solicitações de bloqueio, são prevenidos pelo uso de um paradigma de bloqueio estático, prevendo-se bloqueios no objeto acessado por uma transação na sua 1ª fase e processando essas solicitações na mesma ordem em todos os nós.

5. Solicitação de bloqueio não causam mensagens adicionais.

6. O protocolo é complemente distribuído.

O conceito principal utilizado no método híbrido de controle de concorrência otimista, é o controle dependente de fase [Franaszek et al. 1993], na qual uma transação

tem permissão de ter múltiplas fases de execução com diferentes métodos de controle de concorrência em diferentes fases.

6.5 CONSIDERAÇÕES FINAIS

Os métodos de controle de concorrência para bancos de dados distribuídos heterogêneos devem assegurar as propriedades (ACID) das transações, garantindo principalmente autonomia aos bancos de dados envolvidos.

Todos os métodos apresentados, asseguram, de uma forma completa ou parcial, as propriedades e as características dos SBDH, fazendo com que, ao serem combinados, os métodos resultantes possam diminuir ou até mesmo evitar desperdício dos recursos do sistema, como também, atingir a serialização global e local das transações e subtransações com sucesso, diminuir ao máximo a existência de *deadlocks* , visto que o ideal é ser livre de *deadlock*s.

Se for feita uma comparação para se escolher o melhor mecanismo, deverá se levar em consideração alguns parâmetros como: número de bancos de dados locais por nó; tamanho do bancos de dados, possibilidade de existir *deadlock*s, capacidade de poder dividir em subtransações as transações globais, entre outros.

CAPÍTULO 7

CONCLUSÕES E RECOMENDAÇÕES PARA TRABALHOS FUTUROS

7.1 CONCLUSÕES

A principal razão das pesquisas na área de controle de concorrência em bancos de dados distribuídos heterogêneos é a garantia das operações de atualizações. O controle de concorrência é responsável pelo sucesso da execução destas operações e consequentemente da garantia da consistência dos dados. A a autonomia e a heterogeneidade dos bancos de dados, influenciam diretamente na gerência de controle de concorrência. Neste trabalho foram apresentado os tipos de autonomia e heterogeneidade dos bancos de dados distribuídos heterogêneos e como estas características influenciam no gerenciamento do controle de concorrência.

As formas mais comuns de heterogeneidade dos bancos de dados distribuídos são: heterogeneidade de modelos de dados, de linguagens para desenvolvimento das aplicações, heterogeneidade semântica, etc.

A forma como os dados estão distribuídos no ambiente heterogêneo é outro fator importante; a distribuição de dados mais comuns são: a replicação, que pode ser total ou parcial; a fragmentação que pode ser horizontal, vertical e mista; e a transparência, que pode ser de replicação e fragmentação; de acesso e localização.

A autonomia dos bancos de dados distribuídos está relacionada com a distribuição do controle do sistema, isto é, quanto mais autonomia os módulos do SGBDDs possuirem, menos homogeneidade terão. A autonomia garante liberdade para usar quaisquer modelos de dados, linguagens de consultas, técnicas de gerenciamento de transações. Quanto mais autônomo o sistema de bancos de dados, maior será a capacidade de tomada de decisões, executar as transações que lhe são submetidas, da forma que convém ao SGBDs.

Os bancos de dados heterogêneos podem ter vários tipos de autonomia: de projeto, de comunicação, de execução, de associação. Cada tipo de autonomia, de forma isolada ou em conjunto, possibilita maior facilidade ou dificuldade no projeto dos mecanismos de controle de concorrência.

A arquitetura dos bancos de dados distribuídos heterogêneos é derivada de fatores como a autonomia e a heterogeneidade. Neste trabalho foi apresentada as duas arquiteturas mais frequentemente usadas: os bancos de dados federados e os bancos de dados múltiplos.

Os sistemas de bancos de dados federados possuem dois tipos:

Fracamente acoplados: são bancos de dados altamente autômos, que somente realizam operações de leitura;

Fortemente acoplados: são sistemas que realizam qualquer tipo de transação com qualquer tipo de operação (atualização).

Os BDs Múltiplos são um tipo especial de bancos de dados distribuído, pois cada bancos de dados participante é um sistema, por si só, gerenciador de bancos de dados múltiplo (SGBDM) autônomo e heterogêneo. A tabela 7.1 resume a arquiteura de BDDHs em função das caracterísitcas dos mecanismos utilizados para resolver o problema do controle de concorrência.

Arquiteura / **Características Dos Mecanismos**	**Bancos De Dados Múltiplos**	**Bancos De Dados Federados**
Independente da distribuição	Sim	Não
Usa Locking de duas fases	Sim	Sim
Abordagem otimista	Sim	Sim
Abordagem Pessimista	Sim	Não
Deadlocks	Não	Sim

Tabela 7.1. Resumo da arquiteura dos BDDHs em função dos mecanismos de CC

Considerando-se as caracterísiticas de autonomia e heterogeneidade em BDDHs, o mecanismo mais utilizado comercialmente de controle de concorrência é o *Two-Phase Locking* (2LP), para sincronizar o acesso aos bancos de dados. Porém, para se projetar um mecanismos de controle de concorrência, leva-se em consideração a arquitetura do sistema para o qual está sendo desenvolvido. Logo, a dinâmica desses mecanismos podem variar em função dos tipos de bancos de dados.

Foram apresentados, no capítulo 6, os mecanismos de CC mais utilizados em bancos de dados heterogêneos. Cada mecanismo apresentado faz parte de uma abordagem otimista, pessimista ou de ambas. Cada abordagem possui uma dinâmica específica para gerenciar os controles de concorrência.

A abordagem pessimista sincroniza a execução das transações no início do ciclo de vida da mesma (*Timestamp* e *Wait*). Na abordagem otimista, a validação vai para o fim do ciclo de vida (*Rollback*), o que acarretará menores quantidades de transações conflitantes. Logo, quando o sistema em uso necessita de validações no início do ciclo de vida das transações, a melhor opção é trabalhar com o mecanismo de *timestamp* e *wait*, pois evita-se introduzir prováveis inconsistências nos bancos de dados; nestes mecanismos as transações são bloqueadas no início da execução, e após sua execução são desbloqueadas. Nesta abordagem poderão ocorrer problemas como o *livelock* (ocorre falha no momento de realizar os bloqueios necessários) e o *deadlock* (paralisação do sistema, pois não foram bloqueados os recursos necessários para uma ou mais transações executarem). A tabela 7.2 resume as caracterisiticas dos algoritmos de controle de concorrência em função das abordagens.

Abordagem / **Características dos Algoritmos**	**Otimista**	**Pessimista**
Concorrência	Alta	Baixa
Detecção de abortos	Precocemente	Não detecta
Bloqueios dinâmicos	Realiza	Não realiza
Desperdícios de recursos	Pouco	Muito
Deadlocks globais	Livre	Ocorre
Sincronização da execução das transações	No final do ciclo	No início do ciclo

Tabela 7.2. Resumo dos Algoritmos em função das Abordagens

Sistemas que necessitam de alto grau de confiabilidade optam por usar esse tipo de mecanismos, pois tem a garantia de que a execução de todas as transações

necessárias somente irão ser finalizadas, se os recursos forem alocados a *priori*. A desvantagem da utilização deste mecanismos é o desperdícios de recursos e a grande quantidade de prováveis *deadlocks*.

A abordagem otimista possui um mecanismo inverso da pessimista, pois as validações das transações ocorrem no fim do ciclo de vida da transação. Tudo o que for necessário para a execução das transações é alocado dinamicamente. Qualquer problema que vier a ocorrer com essas alocações, apenas a transação ou as transações que estiverem envolvidas com o problema terão que reiniciar. Isso faz com que o sistema tenha um melhor desempenho, diminui o desperdício de recursos além de evitar *deadlocks* globais. Os recursos são alocados no momento em que são necessários e somente as transações que não conseguirem alocar seus recursos é que precisam ser reinicializadas. Os *dealocks* não paralisaram o sistema de forma global e o máximo que poderá acontecer é a paralisação das unidades de trabalho que estiverem envolvidas na alocação.

O mecanismo de controle de concorrência distribuído híbrido foi projetado especificamente para bancos de dados particionados, enquanto que o Protocolo DASGO foi projetado para bancos de dados fortemente acoplados (DASGO).

7.2 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Como recomendação, no momento da aplicação/utilização dos mecanismos de controle de concorrência, algumas considerações devem ser feitas:

1. Qual o ambiente do sistema, isto é, o tipo de aplicação que este sistema controlará;
2. Possibilidades de falhas nos nós da rede envolvido;
3. Recursos disponíveis;
4. Necessidade de maior ou menor grau de autonomia nos bancos de dados.

Após analisar o ambiente do sistema e considerar as recomendações anteriores, deve-se levantar quais os mecanismos que melhor se aplicam ao tipo de sistema, ou mesmo se há necessidade de desenvolvimento de mecanismo de controle de concorrência, pois, como foi discutido no capítulo 6, os SBDFs fracamente acoplados,

não realizam operações de atualizações de dados, logo o controle de concorrência não é fator crítico no sistema.

As recomendações que podem ser feitas nos mecanismos propriamente ditos são principalmente no mecanismo 2LP, que é um mecanismo de bloqueio em duas fases. Na 1ª fase, bloqueam-se todos os recursos necessários e na 2ª fase ocorre o desbloqueio dos recuros. Uma forma de melhorar o desempenho deste mecanismo é que as duas fases sejam realizadas de forma dinâmica, isto é, cada recurso deve ser bloqueado quando for ser utilizado e desbloqueado após seu uso. Porém, deve ser inserida a condição de se realizar novos bloqueios durante a execução da transação. No 2PL padrão, não ocorrem novos bloqueios enquanto o a transação não for finalizada.

Em mecanismos que utilizam 2PL pode ser introduzida a alteração de realizar os bloqueios e desbloqueios dinamicamente, além de inserir uma ordem de prioridade dos bloqueios para evitar *deadlock*s

Outra alteração é introduzir a alteração anterior do 2PL no protocolo de controle de concorrência híbrido, o que aumentaria o desempenho deste protocolo, mas poderia causas *deadlocks*, pois todos os bloqueis e os desbloqueios ocorrem dinamicamente e algumas transações poderiam ficar esperando por recursos que também estão sendo esperados por outra(s) transação(ões). Neste caso, para evitar-se *deadloks,* deve-se introduzir uma ordem de prioridades para bloqueios dos recurso das transações. Tal ordem, deveria obedecer à critérios (parâmetros) introduzidos pelo DBA do sistema ou levando-se em consideração o nível de prioridade da transação, que pode ser definida pelo sistema (aplicativo) ou mesmo pelo SGBDDHs.

Nos mecanismos que utilizam o algoritmo *timesatamp* Ordering pode-se introduzir o ajuste da ordem de serialização global dinamicamente; isso tem como conseqüência a diminuição dos *deadlock*s.

Com relação à sincronização da execução das transações, estas deveriam ser sincronizadas no fim do ciclo de vida, pois dimuiria o desperdício de recursos.

Há algumas soluções existentes no mercado, como o *DB2* ou o *Oracle*, que permitem interconexão com produtos como *Interbase*, *MySQL* e outros. Interligam as bases de dados através de interfaces para atualizaçõe, sendo que realizam o controle de concorrência utilizando o mecanismo 2PL, pois há necessidade de bloqueios para garantir a consistência. Nestes casos, para melhor desempenho e garantia da

consistência, recomenda-se que os modelos de dados, em cada nó do sistema, sejam os mesmos.

Em ambientes totalmente heterogêneos, onde os modelos de dados, a linguagem de desenvolvimento dos programas da aplicação e etc., são todos diferentes, cada nó participante do sistema poderá utilizar um mecanismo diferente de controle de concorrência. Mas, para que a sincronização na atualização ocorra, é muito difícil evitar o uso de mecanismos baseados no *locking* ou no *rollback*, pois ambos garantem a consistência dos bancos de dados.

CAPÍTULO 8

REFERÊNCIAS BIBLIOGRÁFICAS

ATZENI, Paolo; Ceri, Stefano; Paraboschi, Stefano et al. **Database Systems**: concepts, languages & architectures. McGraw Hill Publishing Company. 2000. p. 349-390.

BATINI, C; Lenzerini, M.; Navathe, S. B.. **A comparative analysis of methodologies for database schema integration**. ACM Computing Surveys. p. 323-364. December 1986.

BERNSTEIN, P. A.; Goodman, Nathan. **An algorithm for concurrency control and recovery in replicated distributed databases**. ACM Trans. Database Syst. p. 596 – 615. December 1984.

BERNSTEIN, P. A.; Hadzilacos, Vassos; Goodman, Nathan. **Concurrency Control and Recovery in Database Systems**. Addison-Wesley Publishing Company. Reading, Massachusetts. 1987. p. 143- 160

BHARGAVA, Bharat. **Concurrency Control and Reliability in Distributed Systems**. Van Nostrand Reinhold Company Inc. New York. 1987. p. 318-333.

BHARGAVA, Bharat. **Concurrency Control in Database Systems**. IEEE Transaction on Knowledge And Data Enginnering, Vol. 11, No. 1. p. 3-16. January/February 1999.

BHASKER, Bharat; Egyhazy, Csaba; Tritanis, J.; kostantinos, P.. **The architecture of a heterougeneous distributed database management system:** The access view integrated database (DAVID). ACM Computing Surveys. ACM Annual Conference on Comunications. p. 173-180. septembre 1992.

BOAVENTURA NETTO, Paulo Osvaldo. **Grafos:** Teoria, Modelos, Algoritmos. Editora Edgard Blücher LTDA. São Paulo. 1996. p. 9-41.

BREITBART, Yuri; Silberschatz, Avi. **Mutidatabse Update Issues**. Proceedings of the second international conference on Information and Knowledge Management. p. 135-142. June. 1988.

BREITBART, Yuri; Georgakopoulos, D.; Rusinkiewicz, M.; Silbercschatz, Avi. **On rigorous transaction scheduling**. IEEE Transactions on Software Engineering. p. 100-115. September. 1991.

BURGER, Albert; Kumar, Vijay. **"PRABHA":** A Distributed Concurency Control Mechanism. Jounal of the Association for Computing Machanery. Vol. 2. No. 4. 1990. p. 392-397.

DATE, C. J.. **Introdução a sistemas de bancos de dados**. Editora Campus LTDA. Rio de Janeiro (Tradução da 4ª edição original). 1991. p. 431-446.

EGYHAZY, Csaba J.. **A software architecture for heterogeneous distributed database systems**. Proceedings of the 19th anual conference on computer science. San Antonio, TX USA. 1991. p. 656.

ELMAGARMID, Ahmed; Rusinkiewicz, Marek; Sheth, Amit. **Management of Heteregeneous and Autonomous Database Systems**. Morgan Kaufmann Publishers, Inc. San francisco, California. 1999. p. 1-50.

ELMASRI, Rammez; Navathe, Shamkant. **Fundamentals of Database Systems**. Second edition. Addison-Wesley. Publishing Company. 1998. p. 527-575.

FAKETE, Alan; Lynch, Nancy; Wehil, William E.. **A serialization graph constructions for nested transactions**. Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. 1990. p. 203-290.

FINGER, Marcelo. **Controle de concorrência e distribuição dos dados**. http://www.whiteorb.org/~jef.html. 04/09/200.

FRANASZEK, P. A.; Haritsa, J. R.; Robinson, J. T. et al. **Distributed Concurrency Control based on Limited Wait Depth**. IEEE transcation Parallet na distributed Systems. Vol. 4, no. 6. November. 1993. p. 151-200.

GARDARIN, G.; Valduriez, Patrick. **Ralational databases and knowledge bases**. Addison-Wesley Publishing Company. Massachusetts. 1989. p. 215-250.

GEORGAKOPOULOS, D. **Transaction Management in Multidatabase Systems**. PhD Thesis, Departament of Computer Science, University of Houston, 1990.

GLIGOR ,V.; Popescu-Zeltin, R.. **Transaction management in distributed heterogeneous database management system**. Information Systems. 1986. p. 287-297.

HADZILACOS, Thanasis. **Serialization Graph Algorithms for Multimersion Concurrency Control**. Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART of principles of database systens. 1988. p. 135-141.

HWANG, SAN-YIH; Huang, Jiandong; Srivastava, Jaideep. **Concurrency Control in Federerated databases: a dynamic approach**. Proceedings of the second international conference on Information and Knowledge Management. Novembre. 1993. p. 694-703.

KORTH, Henry F.; Silberschatz, Abraham. **Databse Systems Concepts**. New York. McGraw-Hill, 1998. p. 379-424.

KUNG, H. T.; Robison, J. T.. **On optimistic:** Methods for concurrency control. ACM Transaction Database Systems. Vol. 6. No.2. p. 213-226. june 1981.

LAMPORT, L. **Time clocks and ordering of events in a distributed system**. Comm. ACM. vol. 21. Tema7. p. 558-565. 1978.

LARSON, James A.. **A flexible reference architecture for distributed database management**. Proceedings of the 1985 ACM thirteenth annual conference on Computer Science. 1985. p. 58 – 69.

LARSON, James A.; Sheth, Amit P.. **Federated Database Systems for managing distributed, heterogeneous, and autonomous databases**. ACM Computing Surveys, vol.22, nº 3. p. 183-236. septembre 1990.

LAUSEN, G.. **Concurrency Control in Database System:** A step towards the integration of optimistic methods and locking. Proc. ACM Ann. Conf. 1982. p. 241 – 251.

LE Lann, G. **Concistency, syncronization and concurrency control. In:** Distributed databases. Draffan, I.W. and F. Poole. Cambridge University Press, Cambridge. Massachusetts. 1980. p. 195-222.

LUTTI, Mário V.. **Topologia de banco de dados**. www.vipsite.com.br. 20/02/00

MAKKI, Kia; Pissinou, Niki. **Detection and resolution of deadlocks in distributed database systems.** Proceedings of the 1995 conference on International conference on information and knowledge management, 1995. p. 411 – 416.

ÖZSU, M. Tamer; Valdurez, Patrick. **Principles of distributed database systems.** 2nd ed. Prentice Hall. New Jersey. 1999. p. 75-155.

PAPADIMITRIOU, Christos H.. **The serializability of concurrent database updates**. Journal of the ACM. Vol. 26, no. 4, 1979. p. 631-653.

PAPADIMITRIOU, Christos H. **A Theorem in Database Concurrency Control**. Jounal of the Association for Computing Machanery. Vol. 29. No. 4. p. 998-1006. 1982.

PAPADIMITRIOU, Christos H.; Kanellakis, Paris C.. **On Concurrency Control by**

Multiple Versions. ACM Transactions on Database Systems. Vol. 3. No. 1. p. 89-99. March 1984.

PERRIZO, William; Rajkumar, Joseph; Ram, Prabhu. HYDRO: **A heterogeneous distributed database system**. Proceedings of the 1991 ACM SIGMOD. International conference on management of data. Denver, CO USA. 1991. p. 32-39.

PU, Calton. **Superdatabase for composition of heterogeneous databases**. In International IEEE Conference Mangement of Data. 1988. p. 167 – 176.

ROTHNIE, J.B.; Bernstein, P. A.; Fox, S.; Goodman, N. et al. **Introduction to a system for distributed databases (SDD-1)**. ACM Trans. Database Syst. Vol 5. No. 1. p. 1-17. Mar. 1980.

ROUSSOUPOULOS, N.; Litwin, Witold; Mark, L.. **Interoperability of multiplus autonomous database**. ACM Computing Surveys. 1990. p. 267-291

SHETH, Amit P.. **When wil we have thue heterogeneous database systems**? Position Statement. IEEE. 1987. p. 747-748.

SCHULDT, Heiko; Alonso, Gustavo; Schek, Hans-Jörg. **Concurrency control and recovery in transactional process management**. Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. May 1999. p. 316 – 326.

SILBERSCHATZ, Abraham; Galvin, Peter. **Operation Systems Concepts**. Massachusetts. Addison-Wesley Publishing Company. 1998. p. 207-235.

STONEBRAKER, Michael; Brown, Paul. **Object-relacional DBMSs. Tracking the next great wave**. Morgan Kaufman Publishers, Inc. San Francisco, California. 1999. p. 27-53.

STONEBRAKER, Michael; Hellerstein, Joseph M.. **Readings in... database systems.** Morgan Kaufmann Publishers, INC. 1998. p. 169-381.

STONEBRAKER, Michael; Woodfill, J.; Ranstrom, J. et al. **Performance analysis of distributed data base systems**. http://db.cs.berkeley.edu/papers/#csd. 20/06/2000.

STONEBRAKER, Michael; DuBourdieux, D.; Edwards, W.. **Problems in supporting database transactions in an operating system transaction manager.** http://db.cs.berkeley.edu/papers/#other. 20/03/200.

TANENBAUM, Andrew S.. **Sistemas Operacionais Modernos**. Editora Prentice Hall do Brasil LTDA. 1995. p. 316-344

THOMAS, R. H.. **A majority consensus approach to concurrency control for multiple copy sistems**. Transaction Database System. ACM. Vol. 4. No. 2. p. 125-198. 1979.

THOMASIN, A.. **Anlysis of Some Optimistic Concurrency Control Schemes Based on Certication**. Jounal of the Association for Computing Machanery, Vol. 29. No. 4. p. 192-422. 1985.

THOMASIN, A.. **Concurrency control:** methods, performance, and analysis. ACM Computing Surveys. Vol 30, No. 1, 1996. p. 70-119.

THOMASIN, A.. **Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing**. IEEE Transaction on Knowledge And Data Enginnering, Vol. 10. No. 1. p. 173-188. January/February 1998.

TRAIGER, Irving L.; Gray, Jim; Galtieri, Cesare A.; Lindsy, Bruce G.. **Transactions and consistency in distributed database systems**. ACM Transaction on Database Systems. Vol.7. No.3. p. 323-342. Septembre 1982.