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

Aluna: Maria Aparecida Denardi

UM ESTUDO COMPARATIVO DE MECANISMOS DE CONTROLE DE CONCORRÊNCIA PARA GERENCIAMENTO DE DADOS EM AMBIENTES DE COMPUTAÇÃO MÓVEL

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

Orientador: Murilo Silva de Camargo

Florianópolis, fevereiro de 2003.

UM ESTUDO COMPARATIVO DE MECANISMOS DE CONTROLE DE CONCORRÊNCIA PARA GERENCIAMENTO DE DADOS EM AMBIENTES DE COMPUTAÇÃO MÓVEL

Aluna: Maria Aparecida Denardi

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

_____

Prof. Dr. Fernando A. Ostuni Gauthier
(Coordenador do Curso)

Banca Examinadora

_____

Prof. Dr. Murilo Silva de Camargo
(Orientador e Presidente da Banca)

_____

Prof. Dr. Rosvelter J. Coelho da Costa

_____

Prof. Dr. Vitório Bruno Mazzola

“ Assim como pensas,
assim andarás e assim caminharas, e
assim como amas, assim atrairás.
Tu estás hoje onde teus pensamentos te trouxeram;
Tu estarás amanhã lá, onde os teus pensamentos te levarem.
Não podes escapar dos resultados dos teus pensamentos,
mas tudo podes suportar e aprender,
podes aceitar e podes te regozijar.
Realizarás os anseios do teu coração,
sejam inferiores ou sublimes ou uma mistura dos dois,
Pois tu sempre gravitarás na direção daquilo
que tu secretamente mais amas.
Nas tuas mãos será colocado o exato resultado dos teus pensamentos:
vais receber aquilo que mereces…
não mais não menos.
Seja qual for o teu ambiente atual,
cairás, permanecerás ou te erguerás com teus pensamentos,
Com tua visão, com teu ideal e te tornarás, então,
tão pequeno quanto o desejo que te controla ou
tão grande quanto a inspiração que te domina”.

Janes Allew

OFERECIMENTO

A Deus, pois com seu amor e sua luz
me concedeu força para vencer obstáculos.
A meus filhos Luís Felipe e Isadora
que ainda pequenos, foram compreensivos
com a minha ausência como mãe em vários momentos.

AGRADECIMENTOS

Ao meu Orientador Professor Murilo, que me impulsionou na evolução deste trabalho, através de incentivo e apoio. Aos meus Pais Francisco e Seliria Denardi, por tudo que fizeram por mim. A Claudete Gasparotto, pelo apoio sem igual em todas os momentos em que precisei.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

RESUMO

No ambiente de computação móvel, o gerenciamento de dados deve suportar as restrições impostas pelas características deste paradigma, como mobilidade, desconexão e baixa largura de banda. O gerenciamento de dados em computação móvel é alvo de muitas pesquisas, pois, tenta-se minimizar a comunicação sem fios, especialmente na direção da unidade móvel para a estação de apoio, onde os recursos são mais escassos, tanto do canal de rede como da própria unidade móvel.

Este trabalho apresenta um estudo dos principais modelos de transações móveis e seus métodos de controle de concorrência, e são demonstrados através de tabelas comparativas, quanto a arquitetura, método de controle de concorrência e tipos de transações usadas na unidade móvel e na unidade de apoio estacionária. São analisados os métodos de controle de concorrência, onde se leva em consideração o tráfego de mensagens na rede para prover o gerenciamento das transações conflitantes, o isolamento e a consistência dos dados. Através destas análises são classificados os mecanismos de controle de concorrência que poderão ser utilizados em banco de dados móveis.

PALAVRAS-CHAVE: Computação Móvel, Banco de Dados Móveis, Transações Móveis, Controle de Concorrência.

ABSTRACT

In the mobile computing environment, data management must adjust itself to the restrictions imposed by the characteristics of this paradigm, such as mobility, disconnection and low band width.. Data management in mobile computing is the target of quite a few research projects because one aims at minimizing wireless communication, especially in the direction from the mobile unit to the support station, where there are fewer resources, both in terms of the network channel and of the mobile unit itself.

This thesis presents a study on one of the major models of mobile transactions and its concurrence control methods, which are demonstrated by comparative tables in relation to architecture, concurrence control method and types of transactions used in the mobile unit and in the stationary support unit. Concurrence control methods are analyzed, taking into consideration the network message traffic in order to provide the management of conflicting transactions, the isolation and the consistency of the data. Based on such analysis concurrence control mechanisms that can be used in mobile database are classified.

KEY WORDS: *mobile computing, mobile database, mobile transactions, concurrence control.*

CAPÍTULO I

INTRODUÇÃO

A proliferação e desenvolvimento de sistemas celulares e a popularização de equipamentos capazes de receber informações por meio de ligações sem fio, nos últimos anos expuseram as capacidades e a efetividade de comunicações sem fios e, assim, pavimentou uma ampla área de aplicações em ambientes computacionais baseados em redes de comunicação desta natureza. Historicamente, comunicações de dados sem fios eram principalmente de domínio das grandes companhias com necessidades especializadas; por exemplo, organizações grandes que precisavam ficar em contato com a força de vendas móveis, ou serviços de entrega que precisavam manter contato com seus veículos. Porém, esta situação de comunicações de dados variáveis e sem fios está ficando tão comum quanto sua contraparte que são os dados sobre uma rede fixa.

A comunicação de dados sem fios surge parcialmente por causa da importância por uma computação móvel e, parcialmente, por causa da necessidade para aplicações especializadas como para vendedores comerciais, empresas imobiliárias, empresas aéreas, controle de informações de trânsito, principalmente voltadas a grandes cidades. Nos sistemas onde há mobilidade dos usuários e que envolva localização de objetos, é comum o interesse e a ocorrência de consultas ou atualização de dados. Para isso, utiliza-se banco de dados para produzir informações. Um SGBD (Sistema Gerenciador de Banco de Dados) tem o objetivo de propiciar um ambiente tanto conveniente como eficiente para recuperação e armazenamento das informações

As crescentes pesquisas focando assuntos de aplicação de sistemas móveis, especialmente para o propósito de acesso móvel à informação, em que a preocupação com a integridade, confiabilidade e disponibilidade da informação que transita em uma rede sem fio, têm sido constantes. Os sistemas móveis requisitam informações conectando-se aos repositórios de dados. Dessa forma, os aplicativos disponíveis para computação móvel têm-se apoiado cada vez mais em SGBD, pois necessitam do armazenamento persistente das informações e do controle de integridade relacional dos

dados. Por isso, este estudo é dirigido à verificação de mecanismos de controle de concorrência.

1.1 Objetivo Principal

O objetivo principal deste trabalho é realizar uma análise comparativa das soluções propostas para o controle de concorrência, visando o gerenciamento de dados em ambiente de computação móvel.

1.2 Objetivos Específicos

- Realizar um estudo comparativo quanto a arquitetura de banco de dados utilizada, métodos de controle de concorrência e tipos de transações usadas na unidade móvel e na unidade de apoio estacionária;
- Realizar um estudo comparativo quanto às características dos métodos de controle de concorrência utilizados em transações móveis, levando-se em consideração o tráfego de mensagens na rede para prover o gerenciamento das transações conflitantes, o isolamento e a consistência dos dados.

1.3 Metodologia

Para a realização deste trabalho, foi desenvolvido uma pesquisa teórica sobre os conceitos de computação móvel, banco de dado móveis e controle de concorrência para o gerenciamento de dados em ambientes móveis. Tendo como base diversas bibliografias, tais como: livros, dissertações, teses, artigos, relatórios técnicos, documentos oficiais de congressos, Workshops e sites da Internet. O material utilizado foi obtido principalmente por meio de pesquisa em bibliotecas digitais . Realizou-se uma pesquisa exaustiva, procurando obter o que já existe publicado na área deste trabalho.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte forma: O capítulo II apresenta uma revisão dos principais aspectos relacionados às arquiteturas de banco de dados.O capítulo III apresenta a conceituação de transações e as propriedades ACID. No capítulo IV são apresentadas as características de transações móveis e também alguns modelos de transações móveis. O capítulo V apresenta o conceito dos mecanismos de controle de concorrência e relaciona os mecanismos utilizados pelos modelos de transações móveis referidos no capítulo IV, bem como são definidas as arquiteturas de banco de dados de cada tipo de transação móvel. No capítulo VI são apresentados e analisados cinco mecanismos de controle de concorrência utilizados para transações móveis.

CAPÍTULO II

BANCO DE DADOS MÓVEIS

Os computadores portáteis têm se tornado, a cada dia, mais comuns, juntamente com as tecnologias de comunicação sem fio. À medida que esta tecnologia amadurece, milhões de pessoas se tornam usuários móveis, comunicando-se entre si e acessando vários recursos de informação. Em ambientes de negócios, a capacidade de acessar dados críticos, independente da sua localização, é ainda mais crucial já que dados corporativos precisam estar disponíveis para aplicações rodando em estações de trabalho móvel.

A computação móvel está sendo utilizada nas mais diversas áreas de negócios, nas quais incluem-se vendedores comerciais, empresas imobiliárias, empresas aéreas, controle de informações de trânsito, principalmente voltadas a grandes cidades e outras.

Nos sistemas em que há mobilidade dos usuários e que envolva localização de objetos, é comum o interesse e a ocorrência de consultas ou atualização de dados. Para isso, utiliza-se banco de dados para produzir informações. Um SGBD (Sistema Gerenciador de Banco de Dados) tem o objetivo de propiciar um ambiente tanto conveniente como eficiente para recuperação e armazenamento das informações (SILBERSCHATZ et al, 1999).

Por outro lado, as aplicações que rodam em clientes móveis, carregam informações dos repositórios de dados se conectando periodicamente. Estes clientes constituem uma nova classe, que se diferencia da tradicional, principalmente pelo modo diferente de carregar informações e exibem um novo padrão de acesso. Nesse caso, o banco de dados precisa ter habilidade para carregar informações de um repositório de dados, e operar com estas informações, mesmo quando estiver fora do alcance do repositório, ou seja, quando estiver em modo desconectado. As informações atualizadas no cliente móvel, quando em modo desconectado, serão propagadas para o sistema

assim que houver a conexão novamente com o repositório de dados (BADRINATH & PHATAK, 1995).

As transações de um banco de dados devem levar em consideração as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID). No entanto, com esta nova classe de clientes, no qual a desconexão é comum e não deve ser tratada como uma falha, algumas destas propriedades devem ser mais relaxadas e não apontadas como incorreções.

Neste capítulo, apresenta-se os principais aspectos relacionados a gerenciamento de dados em ambientes de computação móvel: conceitos básicos, tipos de arquiteturas, de banco de dados, replicação de dados, *caching* e *broadcasting*.

2.1 Arquiteturas de Banco de Dados para Ambientes Móveis

Existem duas principais classes de arquiteturas nas quais os sistemas de bancos de dados móveis se enquadram: bancos de dados cliente/servidor e bancos de dados distribuídos.

Na arquitetura de bancos de dados cliente/servidor, o gerenciador de banco de dados é uma unidade fixa e as unidades móveis consultam e mandam informações para esta unidade. Por outro lado, na arquitetura de bancos de dados distribuídos tem-se uma visão integrada dos dados, os quais são armazenados nas unidades fixas e móveis e que dispõem de uma certa autonomia local.

2.1.1 Arquitetura de Bancos de Dados Cliente/Servidor

Na arquitetura de banco de dados cliente/servidor, o servidor faz a maior parte do trabalho de gerenciamento de dados. Isso implica que todo o processamento de consulta e otimização, gerenciamento de transações e gerenciamento de armazenamento, é feito no servidor (ÖZSU & VALDURIEZ, 2001). O cliente, além da aplicação de interface do usuário, tem um módulo cliente de SGBD responsável pelo gerenciamento dos dados

que são colocados no *cache* do cliente. As solicitações são executadas pelos clientes e atendidas pelo servidor (Figura 2.1). Segundo (SILBERCHATZ et ali, 1999), as funcionalidades de um banco de dados podem ser divididas em duas categorias: *front-end* e *back-end*.

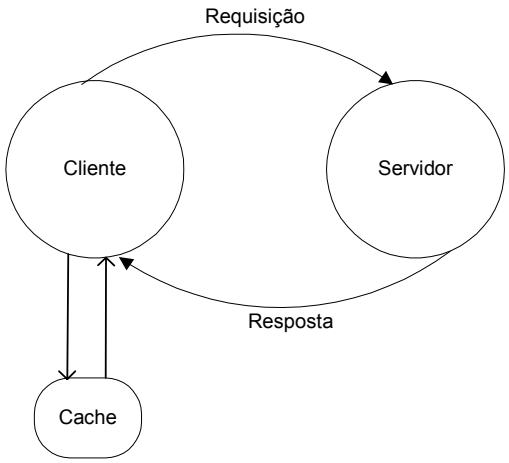

Figura 2.1 - Cliente Servidor Tradicional

A gerência de estruturas de acesso, desenvolvimento e otimização de consultas, controle de concorrência e recuperação de falhas são funcionalidades da categoria *back-end*. Por outro lado, as ferramentas como, gerador de relatório, formulários e recursos de interface, fazem parte da categoria *front-end*. A interface entre o *back-end* e o *front-end,* é feita por meio de um programa de aplicação,usando, por exemplo, SQL.

Porém, as estratégias de cliente/servidor tradicionais permitem a consistência de *cache* transacional que requer comunicações extensas entre um cliente e um servidor, o que não é apropriado em um ambiente de computação móvel, pois as desconexões são freqüentes e sem um tempo limite para ficar desconectado.

Para suportar as desconexões, as unidades móveis trabalham com o *hoarding*, que é um acúmulo de informações antes da desconexão. Neste contexto, os dados do banco de dados podem ficar armazenados no *cache* e serem usados durante a desconexão, simulando o acesso ao servidor. Podem ser efetuadas atualizações, deleções e consultas a estes dados do *cache*. Quando a unidade móvel é reintegrada à rede, conecta-se ao servidor e as atualizações são efetuadas(Figura 2.2).

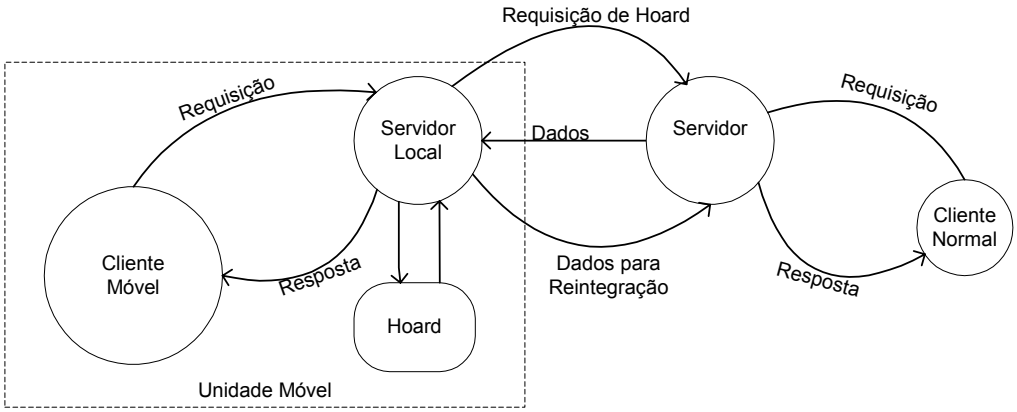

Figura 2.2 - Modelo *Hoard* e Reintegração

O modelo mostrado na Figura 2.2 considera o servidor tanto para unidades móveis quanto para fixas. O servidor, neste modelo, é fixo também.

Por se tratar de clientes móveis que acessam um banco de dados de um servidor, estes clientes precisam trabalhar com réplicas dos dados, pois a qualquer instante eles poderão se desconectar. Desse modo, usa-se o conceito de replicação e de *hoarding* de dados dentro da arquitetura cliente/servidor. Na seqüência, serão apresentados os conceitos de *hording* e de replicação, com o objetivo de verificar quais as vantagens e desvantagens destes, quanto aos dados carregados nas unidades móveis.

Quando um arquivo é submetido à reintegração no servidor, existe um *timestamp* (marca de tempo) deste acúmulo. Depois que o *timestamp* de modificação do arquivo é feito no servidor, o arquivo é reintegrado com sucesso no servidor. Nesse caso, a cópia do cliente, do arquivo, substitui a cópia de servidor (BADRINATH & PHATAK, 1998).

Existem algumas diferentes possibilidades de executar o *hoard* (acúmulo) e a reintegração dos dados tanto no cliente quanto no servidor.

Uma relação inteira pode ser mapeada sobre um único arquivo, ou um subconjuntos de tuplas de uma determinada relação pode ser mapeado sobre arquivos separados. No primeiro caso, cada cliente móvel levaria a relação inteira em seu disco local. Isto não é obviamente desejável desde que um cliente móvel típico não teria os recursos (espaço de disco) para acumular relações inteiras. Até mesmo se fizesse, a

maioria do acúmulo de dados poderiam ser colocados fora da localidade de acesso para o cliente móvel (BADRINATH & PHATAK, 1998).

No segundo caso, leva-se em consideração a propriedade de atomicidade da transação, pois as unidades de reintegração são tuplas e não atualizações de transação. Conseqüentemente, um mecanismo a mais é exigido para controlar ``ligeiramente'' os dados incompatíveis, enquanto há uma negociação eficaz para o acúmulo de dados, que será em uma ordem de magnitude menor que o tamanho da relação.

O servidor é ponto de sincronismo nessa arquitetura. Todas as atualizações devem ser propagadas ao servidor. O servidor também controla privilégios de acesso dos vários clientes. São armazenadas no servidor relações inteiras (como oposto, o armazenamento nos clientes é apenas parte da relação, considerando o *hoard*). Quanto a organização física da relação no servidor, ela pode estar em fragmentos físicos para otimizar a integração e o *hoarding.* Ainda com relação à organização física dos dados no servidor, dependerá muito do projetista do banco de dados, para que as relações fiquem inteiras ou em fragmentos.

2.1.2 Arquitetura de Bancos de Dados Distribuídos

Consideramos um sistema de banco de dados distribuído, quando existe uma coleção de múltiplos bancos de dados logicamente interrelacionados, distribuídos sobre uma rede de computadores convencional ou sem fios (ÖZSU & VALDURIEZ, 2001).

Dentro da arquitetura móvel, temos estações base que controlam a comunicação das unidades móveis entre si, dentro de uma mesma célula ou a comunicação com outra unidade móvel em outra célula, ou ainda a comunicação com unidades fixas.

Neste ambiente, os dados podem estar dispostos nas unidades móveis ou na rede fixa (Figura 2.3).

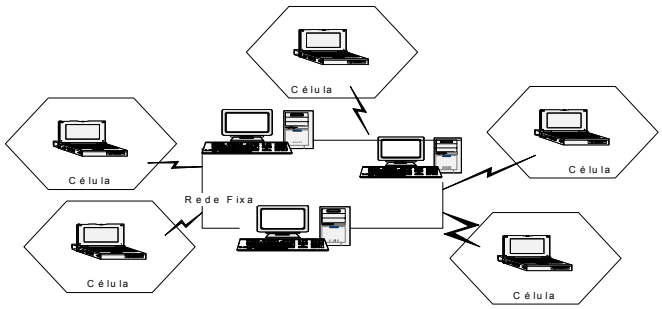

Figura 2.3 – Disposição das Unidades Móveis e Fixas

Uma abordagem mais comumente usada para a transferência de dados nesta estrutura é que as unidades móveis comunicam-se com unidades fixas que armazenam os dados e baixam os dados destas à medida que precisarem deles. Com isso, o problema de gerenciamento de dados distribuídos não é muito afetado pela mobilidade, pois os dados estão armazenados principalmente nos computadores da rede fixa.

Outra abordagem é, que as unidades móveis armazenam dados nativos que podem ser compartilhados e que em um certo momento serão acessados por outras unidades móveis. Porém, nesta arquitetura, onde unidades móveis podem se mover entre várias células, a mobilidade cria questões como migração de endereços, manutenção de diretórios e dificuldades para localizar estações. Isto exige uma funcionalidade adicional do sistema para rastrear as unidades móveis.

Nos tópicos seguintes serão abordadas algumas funções dos bancos de dados em ambientes móveis, que são atingidos pela mobilidade das unidades.

2.1.2.1 Gerenciamento de Diretório

Normalmente, em banco de dados distribuídos tenta-se colocar os dados distribuídos de forma que fiquem perto de onde são mais acessados. No entanto, este conceito muda quando projetamos banco de dados distribuídos sobre ambiente móvel, pois as unidades que contêm os dados residentes mudam de local. Por isso, a questão fundamental é localizar a unidade móvel que contém os dados requisitados.

Duas abordagens foram propostas por IMIELINSKI & BADRINATH (1994). Na primeira abordagem, cada unidade móvel tem uma estação base inicial, que mantém o controle de sua localização, recebendo informação de sua movimentação. Esse método pode gerar latência de acesso bastante elevado, caso a unidade móvel esteja distante de sua base inicial. Isto acontece porque, para localizar a unidade móvel de uma certa base inicial, terá que entrar em contato com outra base inicial para ter a localização atual.

Na segunda abordagem, a base é a difusão restrita dentro da área da estação móvel que deseja acessar esta estação. Como exemplo, pode-se visualizar uma estação A que deseja estabelecer acesso. Primeiro é difundida uma mensagem de investigação às estações circunvizinhas para verificar se B está nessa área. Caso não ocorra resultado, a estação A entra em contato com a estação inicial de B. Existem "n" abordagens sobre a localização de unidades móveis.

Outra questão de gerenciamento de diretórios é encontrar a localização ótima dos dados, quando a unidade em que estão os dados a serem acessados está em movimento. Devemos lembrar que o gerenciamento de diretórios é muito importante, pois qualquer transação só pode ser efetuada com a localização dos dados (ÖZSU & VALDURIEZ, 2001).

2.1.2.2 *Cache*

Vários sistemas aplicativos utilizam a opção de colocar dados no cache da unidade móvel por um período de tempo. Como estamos considerando unidades móveis, a fonte de alimentação é limitada e restringe a quantidade de processamento executada nestas, fazendo com que a computação seja efetuada nas unidades fixas e enviando para as unidades móveis apenas os resultados.

Porém, existem unidades móveis com um poder de processamento e armazenamento maior. Estas unidades suportam a participação no transporte de dados, no qual um sistema gerenciador de banco de dados pode residir por inteiro. Com isso, surgem as questões relacionadas ao gerenciamento dos caches nas estações móveis.

2.1.2.3 Difusão de Dados

Devido à mobilidade das unidades, qualquer dado a ser difundido deve ser dinâmico e adaptável. À medida que a unidade se desloca de uma célula para outra, a necessidade de dados do cliente se move conjuntamente. Portanto, a difusão de dados terá que se adaptar a esse movimento. Quando a unidade se desloca de uma célula para outra, os dados que esta unidade necessitava devem ser retirados do conteúdo da difusão de dados da célula antiga.

2.1.2.4 Otimização e Processamento de Consultas

Uma das funções mais atingidas pela mobilidade do ambiente é o processamento de consultas. Percebe-se o efeito da mobilidade nas consultas propostas, assim como nas técnicas de otimização que podem ser utilizadas.

Há a necessidade de admitir consultas dependentes da localização. Essas consultas podem ser processadas, aumentando a cada consulta com a informação de localização. Independente da abordagem utilizada, quando uma consulta é solicitada, ela deve estar vinculada a uma localização e a um conjunto de valores de dados. Pois, com a mobilidade da unidade móvel, a localização quando a consulta é requisitada pode ser diferente quando houver o término desta consulta.

Quanto à otimização de consultas, como as unidades de acesso se movem, há uma dificuldade para determinar os custos de comunicação entre a unidade de acesso e a unidade onde os dados residem. Considera-se ainda que, como os dados estão distribuídos entre as unidades, então ambos podem estar em movimento. Como maior fator complicador, está a baixa largura de banda e a variação de largura de banda entre a rede fixa e a rede móvel. Logo, neste tipo de ambiente os planos e custos associados mudam com base no movimento. Neste caso, são necessárias estratégias dinâmicas que se adaptem ao ambiente móvel.

2.2 Caching

Caching é um mecanismo caracterizado por sua granularidade, estratégia coerente, e política de substituição de *cache*. Este mecanismo é utilizado para otimizar o tempo de resposta a consultas na computação móvel, pois a baixa largura de banda aumenta muito o custo da comunicação sem fio. Esta técnica também é útil para dar suporte quando a unidade móvel estiver em estado desconectado.

Uma questão crítica nas técnicas de *caching* é como antecipar os dados que serão necessários futuramente na unidade móvel. Uma possível alternativa é utilizar as informações a respeito do histórico de uso dos dados de forma a prever necessidades de acesso futuro. Quando há a reconexão da unidade móvel à rede, o dispositivo reintegra as atualizações efetuadas quando da desconexão. O software na unidade móvel armazena em *log* as operações de atualização realizadas que serão enviadas ao servidor, quando houver a conexão.

Porém, para que o mecanismo de caching dê um bom resultado, é preciso definir as estruturas dos dados e as estratégias de gerenciamento do *cache*. Para esta definição, é necessário considerar as características de cada aplicação e o padrão de acesso aos dados (FERREIRA & FINGER, 2000).

O funcionamento do *cache* é baseado nas possíveis operações sobre os dados, na natureza de seu conteúdo e na unidade básica de seus dados.

As unidades mais utilizadas em *cache* são: arquivos, tabelas, objetos, páginas, tupla e consultas. Cada aplicação possui sua própria relação custo/benefício entre o "*overhead"* de informações de controle e a flexibilização para reutilização dos dados.

As operações sobre os dados podem ser divididas em 3 categorias, como segue:

- Dados somente para consulta – O *cache* não permite atualizações locais sobre os dados. Com isso, os dados não precisam ser reintegrados ao banco de dados e a estrutura do *cache* poderá ser mais simples.
- Dados não concorrentes – Atualizações locais são permitidas, mas os dados replicados ficam bloqueados no servidor, evitando atualizações conflitantes.

- Dados concorrentes – Ocorre quando os dados replicados podem ser alterados tanto no servidor quanto no *cache*. Para prevenir possíveis conflitos, o *cache* deve armazenar informações sobre as alterações ocorridas.

Quanto a natureza do *cache*, podemos definir como estática ou dinâmica. Considera-se natureza estática quando o conteúdo *cache* é definido com antecedência, podendo ser solicitado pela aplicação no início da execução.

Considera-se natureza dinâmica quando seu conteúdo é definido de forma dinâmica em tempo de execução.

Existem estratégias de gerenciamento do *cache*, que são utilizadas para melhorar o desempenho do *cache*. Essas estratégias são definidas levando em consideração a reutilização dos dados. Quanto mais reutilizados forem os dados, à sua classificação será atribuído um valor maior. A classificação define quais dados não são do interesse da aplicação e devem ser substituídos.

2.3 Replicação

Replicação é uma cópia idêntica de uma relação que o sistema mantém. As réplicas são armazenadas nos diversos nós (unidades) da rede. A réplica pode ser da relação inteira ou de fragmentos da relação.

Consideremos as unidades móveis que acessam dados, também acessados por outras unidades ou por elas mesmas em diferentes locais e equipamentos, respectivamente. Devido a conexões raras, esporádicas e fracas, uma cópia local precisa ser mantida no equipamento móvel para evitar o estabelecimento de uma conexão a cada consulta/gravação, facilitando o trabalho da unidade móvel em modo desconectado (LUBINSKI & HEUER, 2000). Tal cópia incrementa a disponibilidade, diminui o tempo de resposta e também melhora a tolerância a falhas. O dado replicado tem que ser mantido também no servidor fixo. Entretanto, esta consistência tem que ser certificada, sem conexões freqüentes ao servidor – por razões de custo, entre outros.

A replicação de dados é um dos problemas-chaves de sistemas de informação móveis. Um tratamento aplicável de replicação neste novo paradigma da computação precisa de soluções novas que os protocolos de replicação existentes não oferecem (LUBINSKI & HEUER, 2000). Bancos de dados comerciais e sistemas de informação atualmente em uso em ambientes móveis sobrecarregam o usuário com a manutenção e consistência dos dados replicados. Para iniciar um serviço móvel, o DBA (Administrador de Banco de Dados) reúne os dados a serem usados no equipamento móvel e o transfere do servidor(*hoarding*). A seguir, o DBA ajusta os parâmetros para especificar quais ações são possíveis sobre o dado em particular, tanto no cliente móvel quanto no servidor. O usuário móvel tem que contar com a consistência dos dados enquanto trabalha num ambiente desconectado. Se novos ou mais dados são requisitados, o usuário móvel precisa conectar-se ao servidor para transferir os dados. Quando consulta o sistema de informação móvel em um ambiente desconectado, o usuário tem que avaliar a atualização e a consistência do dado sem ajuda do sistema. Checagens de consistência são possíveis quando os dados são transferidos de volta ao servidor depois de terminar o serviço móvel. Qualquer conflito subsequente na consistência precisa ser manualmente resolvido pelo DBA.

Requisitos que as técnicas de replicação em banco de dados móveis precisam atingir (LUBINSKI & HEUER, 2000):

- Alta disponibilidade de dados no equipamento móvel, a despeito de baixos custos e uma consistência de dados aceitável;
- Uso de informação de contexto semântico sobre o usuário, atributos da aplicação e do dado, a fim de executar checagem de consistência significativa a baixos custos;

Uso de informação de contexto semântico para realizar replicação dinâmica, que é carga transparente de informação a ser atualizada do servidor, a pedido (quando consulta o sistema móvel, por exemplo).

2.4 *Broadcast* (Difusão de Dados)

A difusão de dados, segundo BARBARÁ (1999), é a entrega de informações de um conjunto de provedores para um grande número de clientes. Este processo de entrega de informações é caracterizado pela assimetria nas comunicações. A largura da banda da rede é maior no sentido servidor–cliente, que no sentido cliente–servidor chamado de *up-link*. Quando um servidor recebe um pedido de dados, localiza-o e retorna para o cliente os dados, o modelo é chamado de *pull-based* (baseada em puxar). Outro modelo é *push-based* (baseado em empurrar), no qual a entrega de dados é feita repetidamente pelo servidor a um conjunto de clientes. Os clientes monitoram a difusão dos dados e aceitam somente aqueles que lhes são necessários para sua computação. Neste modelo, um dos grandes problemas a ser solucionado é a decisão sobre quais os ítens de dados que devem ser disseminados. Uma possível solução para este problema pode ser, prover o cliente com a possibilidade de construir *profile*s *(*perfis) abrangendo apenas os dados que lhe são de interesse.

Os clientes (unidades móveis) estão interessados em acessar dados específicos dos dados difundidos. Segundo IMIELINSKI et al. (1997), o período de *access time* (tempo de acesso) é o tempo comum decorrente do momento que um cliente expressa seu interesse em um dado, submetendo um pedido e o recebimento do dado no canal de difusão. O *tuning time* (tempo de afinação) é a quantia de tempo gasto escutando o canal de difusão. Escutar o canal de difusão exige que o cliente esteja no modo ativo. Isto aumenta o consumo do cliente. Os dados difundidos deveriam ser organizados de forma que o *access time* e o *tuning time* possam ser minimizados.

Em ACHARYA et al (1995), foi proposto *broadcast disks* (disco de difusão), no qual o acesso aos dados não é uniforme. Esta abordagem define discos de difusão que possuem diferentes velocidades e tamanhos. Dados que são transmitidos com a mesma freqüência são considerados pertencentes a um mesmo disco. O algoritmo proposto tem como entrada as probabilidades de acesso às informações (dados) pelos clientes e os parâmetros dos discos, que são o número de discos, que determina a quantidade de freqüências diferentes a serem usadas para difundir os dados,e ,para cada disco, o número de ítens e suas freqüências relativas à difusão. O resultado é a geração de uma

lista de alocação de dados e a ordem de transmissão dos dados de tal forma que o tempo médio entre chegadas de um mesmo dado aproxime-se das expectativas do cliente. Na Figura 2.5, são apresentados três tipos diferentes de organização de difusão de ítens de igual tamanho. A sequência (a) da Figura 2.5 somente considera os dados a serem transmitidos, sem se preocupar com a probabilidade de acesso às informações. Esta forma de transmissão é conhecida como transmissão plana de dados. Nas letras (b) e (c) o item de dados A tem o dobro da freqüência de transmissão dos ítens B e C. Na letra (b), as transmissões de A estão agrupadas, porém de forma aleatória, enquanto que na letra (c) já existe um padrão regular no intervalo de transmissão de qualquer item de dados. A seqüência em (c) sugere uma difusão de multi-discos, sendo que A está armazenado em um disco com velocidade duas vezes maior que as dos discos onde se encontram os ítens B e C.

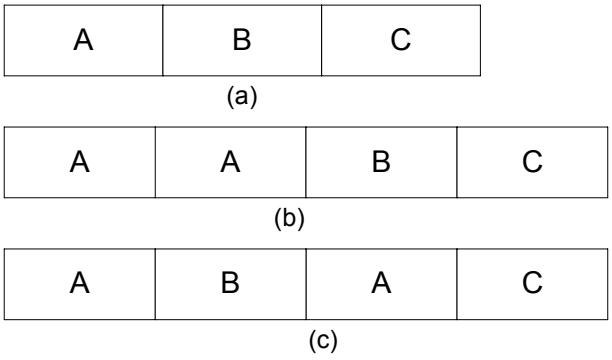

Figura 2.5 - Discos de Difusão
Fonte: PITOURA & SAMARAS (1998)

Os clientes têm necessidades que são dinâmicas. Portanto, é necessário que se estabeleça algum mecanismo para a compensação dos desvios na informação disseminada. Em ACHARYA et al. (1997) encontra-se a integração das arquiteturas *pull-based* e *push-base*d. Em uma operação do tipo *pull-based* os clientes podem requisitar os ítens de dados por meio do envio de mensagens para o servidor, que então retorna a informação para os clientes. A proposta trabalha com a possibilidade de utilização de dois canais de comunicações independentes: *frontchannel* e *backchanne*l. O canal *frontchannel* é utilizado para as operações do tipo *push-base*d, enquanto que o canal *back-channel* é utilizado somente para as operações tipo *pull-base*d. A

disponibilidade de largura de banda da rede é compartilhada entre os dois canais, conforme a necessidade de comunicação

Conforme IMIELINSKI et al. (1994), o cliente pode estar interessado por alguns ítens de dados identificados por alguma chave. Uma forma de índice será transmitida juntamente com os ítens de dados. Este índice conterá a informação de quando o item será transmitindo. Dessa forma, o cliente necessita sintonizar apenas o canal que irá conter a informação necessária. Na maior parte do tempo, o cliente poderá permanecer no modo adormecido *(doz*e), proporcionando economia de energia da estação móvel.

O grande problema é a definição da forma de agrupar o índice e os dados no canal de difusão com o objetivo de minimizar, para o cliente, os tempos de acesso e sintonia aos ítens transmitidos. A proposta é a política "indexação (1, m)" na qual todo o índice é transmitido a cada fração 1/*m* de dados transmitidos. Juntamente com cada dado, é feita a transmissão da quantidade de ítens que ainda irão aparecer no canal de transmissão antes de uma nova transmissão de índice. Para acessar um determinado registro, o cliente deverá sintonizar o canal de difusão e verificar o número de ítens a serem transmitidos até que aconteça a próxima transmissão de índice. Após este procedimento, poderá entrar em modo adormecido *(doz*e), aguardando que a transmissão dos dados aconteça no período informado.

2.5 Considerações

Nesse capítulo foram apresentadas as arquiteturas de bancos de dados que podem ser utilizadas para o ambiente móvel, bem como replicação de dados, *caching* e difusão de dados. Evidencia-se, nesse capítulo, que para se trabalhar com banco de dados em ambientes móveis, existe a necessidade de saber como os dados chegarão até a unidade móvel, como será feito o pedido dos dados, como serão armazenados temporariamente e como serão replicados. Só com essas informações bem definidas será possível ter um bom gerenciamento de dados. Esse gerenciamento de dados se dá pelo bom gerenciamento das transações. No próximo capítulo serão conceituadas transações e as propriedades ACID.

CAPÍTULO III

TRANSAÇÕES

Transação é uma coleção de operações que desempenha uma função lógica dentro de uma aplicação do sistema de banco de dados. Cada transação é uma unidade de atomicidade e consistência. As transações não devem violar nenhuma regra de consistência do banco de dados. Portanto, o banco de dados que estava consistente antes do início da transação deve permanecer consistente após o término da transação. No entanto, durante a execução de uma transação, há a necessidade de aceitar inconsistências temporárias. Essas inconsistências, embora necessárias, podem gerar problemas caso ocorra uma falha (SILBERSCHATZ et ali, 1999) .

Neste capítulo, será abordado o conceito das propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID).

3.1 Gerenciamento de Transações

Quando se trata de ambientes móveis, as desconexões podem ocorrer por tempo possivelmente longos e com as limitações de largura de banda, o que impõe a reavaliação quanto ao modelo de transações e técnicas de processamento das transações, pois devido a mobilidade às transações podem ser executadas de diversas localizações. O controle de concorrência, que é o responsável por administrar a execução concorrente de várias transações, tem como base técnicas típicas baseadas em bloqueios. Porém, como estamos nos referindo a um ambiente móvel, propenso a desconexões e falhas, o bloqueio pode não ser uma boa solução. Caso haja um bloqueio sobre alguns dados em uma unidade danificada, poderiam ser mantidos bloqueados por tempo indefinido, impedindo, dessa forma, o término de uma transação.

Existem algumas abordagens que sugerem modelos de transação mais longas, combinados com esquemas de replicação, além de sub-transações que têm alguma flexibilidade de consistência e processamento de consolidação, que serão relatadas no decorrer do próximo capítulo.

Verifica-se a necessidade de um relaxamento das propriedades ACID, devido a alta taxa de desconexão e a variabilidade de localização.

> **Atomicidade** – Uma transação é unidade atômica de processamento que deve ser executada totalmente, ou totalmente desfeita.
>
> **Consistência** – A execução de uma transação deve levar o banco de dados de um estado consistente a outro estado consistente ao término de sua execução.
>
> **Isolamento** – A execução de uma transação não pode ser afetada por outra transação que esteja sendo executada concorrentemente.
>
> **Durabilidade** – Os efeitos de uma transação confirmada não podem ser desfeitos, e as alterações não podem ser perdidas nem mesmo no caso de uma falha eventual (FERREIRA & FINGER, 2000).

O gerenciamento dessas transações pode ser estático na unidade móvel ou no servidor de banco de dados, ou pode migrar de estação base para estação base, conforme a unidade móvel se desloca (ÖZSU & VALDURIEZ, 2001).

3.2 Propriedades de Transações

Para garantir que uma transação seja confiável e consistente e assegurar a integridade dos dados, o sistema de banco de dados deve manter as propriedades ACID, que serão apresentadas a seguir.

3.2.1 Atomicidade

A transação deve ser atômica, ou seja, ela deve ser executada por completo, ou será totalmente desfeita. No caso de ocorrer qualquer interrupção na execução da transação, é de responsabilidade do sistema gerenciador de banco de dados determinar o que deve ser feito com a mesma, ao se recuperar a falha. O sistema de banco de dados mantém registrado os valores dos dados antes do início da transação e só apaga esta cópia quando a transação for considerada confirmada. Caso contrário, estes valores poderão ser restabelecidos.

Comumente, há dois tipos de falhas. Primeiro, quando a falha da transação ocorre por erros nos dados de entrada, impasses ou alguma razão que impediria de concluir sua tarefa com sucesso. No caso desta falha, o aborto da transação pode ser executado pela própria transação ou ainda pelo SGBD. Esta falha caracteriza a recuperação de transações

O segundo tipo está ligado a quedas no sistema, falhas de processadores, interrupções nas ligações de comunicação e assim por diante. Este tipo de falha é involuntário à transação. Esta falha caracteriza a recuperação de falhas. A principal diferença entre esta falha e a anterior, é que esta pode estar com os dados ainda em memória volátil, o que provocaria a perda dos dados.

3.2.2 Consistência

A consistência de uma transação é o banco de dados estar em um estado consistente antes do início da transação, a transação ser executada e ao término desta o banco de dados estar em estado consistente novamente.

Segundo ÖZSU E VALDURIEZ (2001), há uma classificação quanto aos dados sujos (valores de dados atualizados por uma transação mas ainda não confirmados), em quatro graus de consistência:

- Uma transação de consistência de grau 3:

- 1ª consideração - A transação não subrescreve dados sujos de outras transações;
- 2ª consideração - A transação não consolida nenhuma gravação até concluir todas as gravações, ou seja, até o final da transação completa.
- 3ª consideração - A Transação não lê dados sujos de outras transações.
- 4ª consideração - Outras transações não sujam quaisquer dados lidos por esta transação até que esta seja executada por completo.

- Uma transação tem grau 2 de consistência se, verificar as três primeiras considerações acima citadas no grau 3.
- Uma transação tem grau 1 de consistência se, verificar as duas primeiras considerações citadas no grau 3.
- Uma transação tem grau 0 de consistência se, verificar a primeira consideração citada no grau 3.

Com estes graus de consistência, é possível projetar aplicativos mais flexíveis e que possam utilizar transações que operem em níveis diferentes.

3.2.3 Isolamento

O isolamento de uma transação garante que nenhuma outra transação visualize seus resultados antes desta estar completa. O isolamento é um dos fatores mais importantes para a manutenção da consistência entre as transações.

Considerando os graus de consistência visto anteriormente, verifica-se que quanto maior o grau de consistência maior também é seu grau de isolamento. O isolamento das transações permite que haja execução concorrente, ou seja, mais que uma transação sendo executada simultaneamente.

O isolamento de transações está intrinsecamente ligado à consistência do banco de dados e esta relacionado diretamente ao controle de concorrência.

Os níveis de isolamento são definidos segundo seu fenômeno (ÖSZU & VALDURIEZ, 2001):

Dados sujos: Dados com valores já modificados por uma transação, mas não efetivados ainda.

Leitura não repetível ou leitura vaga: Uma transação lê o valor de um item de dados. Logo após outra transação elimina este item de dado e efetiva. Quando a primeira transação volta a ler este mesmo item, não o encontra, o que gera dois valores de leitura dentro de uma mesma transação.

Fantasma: Uma transação pesquisa em certos ítens de dados, seguindo um critério para esta pesquisa, mas outra transação insere novas tuplas após esta pesquisa. No momento em que a primeira transação pesquisa novamente, haverá mais dados que serão diferentes dos recuperados.

3.2.4 Durabilidade

Esta propriedade da transação garante que, uma vez executada e consolidada a transação, seus resultados se tornem permanentes no banco de dados. Com isso, está assegurado também a permanência destes dados no banco, mesmo se ocorrer alguma falha do sistema. Portanto, com os dados gravados em disco, se caso ocorrer alguma falha, o banco de dados terá facilidade em reconstruir as atualizações realizadas pela transação.

3.3 Controle de Concorrência

Quando várias transações são executadas concorrentemente em um banco de dados, a consistência do banco de dados pode ser violada, mesmo que individualmente as transações estejam corretas. O controle de concorrência tem a responsabilidade de

administrar a interação entre as transações concorrentes para garantir a consistência do banco de dados (SILBERSCHATZ et al, 1999).

Geralmente uma transação móvel é uma transação distribuída, na qual algumas partes da computação são executadas em unidades móveis e algumas partes em unidades fixas.

Entretanto, para ambientes móveis, considerando que o poder de processamento das unidades móveis é menor, implica que o tempo de execução de uma transação será mais longo, além de utilizar por um tempo bastante longo dos recursos do sistema da unidade (PHILIP et al, 1993)(FERREIRA & FINGER, 2000).

Além de o processamento ser menor, se torna comum no ambiente móvel, alguns nós ficarem temporariamente indisponíveis, ou seja, em estado desconectado. Neste caso, a transação não pode ser considerada como falha e ser abortada .

Características das transações móveis:

- Transações móveis envolvem rede sem fio e podem ser executadas tanto em unidades fixas quanto em unidades móveis.
- Quando é usada uma conexão sem fio com a largura de banda média, as transações tendem a ser:
- monetariamente caras;
- ter uma longa vida, devido a demoras de transmissão da própria rede;
- propensão a erros, devido às freqüentes desconexões e também porque as unidades móveis são mais vulneráveis a acidentes;
- são baseadas em sessão. Pelo alto custo, por exemplo, de conexão celular, os dados são enviados em sessões.

A mobilidade resulta em transações que:

- acessam sistemas heterogêneos;
- acessam dados de locais (imprecisos);
- podem envolver dados que são relocados dinamicamente.

Para modelar estas transações móveis que são de longa duração, propensas a erros e heterogêneas, o uso das propriedades ACID restringem muito o controle de uma estrutura complexa. Além disso, as transações ACID não permitem a confirmação(*commit*) parcial ou o aborto (*abort*) parcial ou ainda a recuperação(*recovery*) parcial (PITOURA & BHARGAVA, 1994).

4.4 Considerações

No decorrer desse capítulo, foram estudadas as propriedades ACID. Isso se fez necessário para se ter uma base melhor quanto ao gerenciamento de transações, pois por meio do cumprimento destas propriedades é possível garantir que um banco de dados passe de um estado consistente para outro estado consistente. Porém, como as transações terão que suportar um ambiente móvel, existe a possibilidade do não cumprimento de todas estas propriedades e, mesmo assim, dar garantia ao banco de dados sobre os dados acessados. No próximo capítulo serão abordados os modelo de transações móveis..

CAPÍTULO IV

TRANSAÇÕES MÓVEIS

O ambiente móvel abordado neste trabalho é constituído de uma rede com unidades móveis e unidades fixas, em que as unidades móveis se locomovem e suas conexões de rede também, enquanto ainda estão executando processamentos. As unidades móveis quando em movimento, retêm suas conexões de rede pelo suporte de estações base, ou chamadas também de *Moblity Support Station* (suporte móvel estacionário -MSS). Onde cada estação base é responsável pelo gerenciamento de localização de uma unidade móvel bem como pelo gerenciamento do fluxo de dados enviados.

Considerando que as unidades móveis podem se desconectar acidentalmente quando entram em uma região fora do alcance de qualquer estação base, assim como intencionalmente, quando a largura de banda estiver muito baixa, ou se a bateria estiver quase sem carga e a unidade móvel entra em estado de conservação de energia, se torna caro, em termos de comunicação, o envio e recebimento de dados. Por esta razão, a radiodifusão dos dados das estações base se torna aplicável, visto que a comunicação vertical das unidades móveis se torna bastante onerosa.

No decorrer deste capítulo, serão descritos os modos de operação de uma unidade móvel, considerando que seu modo de operação será determinante para o tipo de transação executado no sistema. Também serão abordadas características das transações móveis que diferem um pouco das transações tradicionais, principalmente no que diz respeito a isolamento e consistência. Após estas características, serão descritos alguns modelos de transações móveis.

4.1 Modos de Operação de uma Unidade Móvel

Para conter as restrições impostas pelo ambiente móvel, é necessário que se verifique os vários modos de operação de uma unidade móvel (Figura 4.1).

Completamente Conectado: Neste estado a unidade móvel está completamente (com largura de banda alta) conectada à rede fixa.

Em Parte Conectado: Este modo é utilizado quando a largura de banda esta escassa. Quando está neste modo, a unidade móvel pode receber conexão da estação base (estação base para unidade móvel), mas não pode estabelecer uma conexão unidade móvel para estação base.

Modo cochilo: a velocidade do *clock* do processador está reduzida e virtualmente nenhuma computação é terminada.

Completamente Desconectado: Quando uma unidade móvel estiver fora de qualquer estação base.

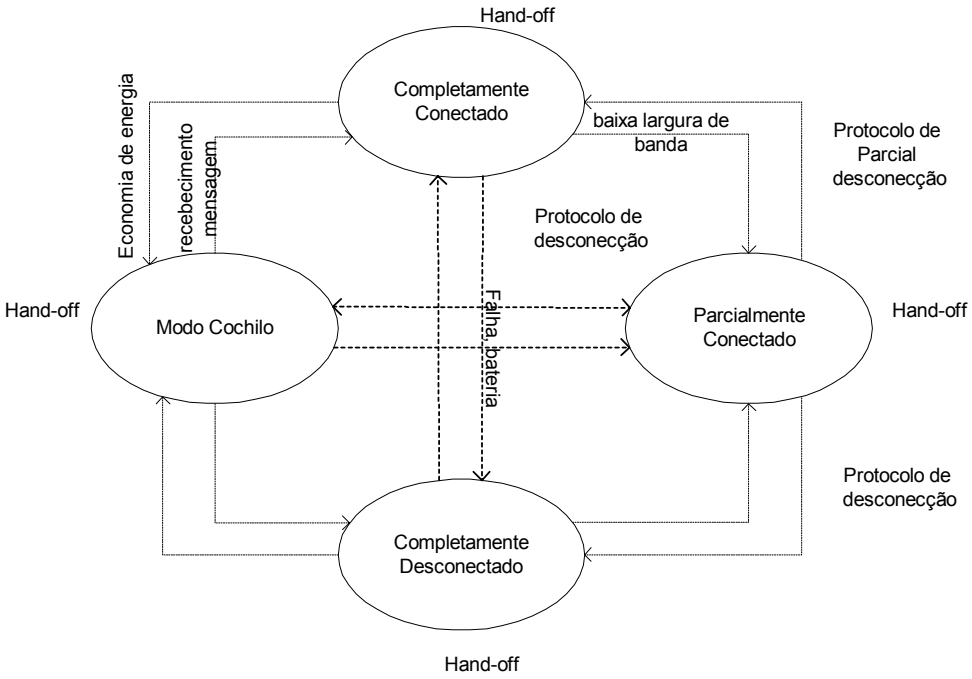

Figura 4.1 – Modos de Operação da Unidade Móvel

Mediante as restrições impostas pela mobilidade, se fazem necessários modelos apropriados de transações em que se acomodem as limitações da computação móvel, levando em consideração o compartilhamento de dados e a garantia de justeza da execução das transações nas unidades móveis como nas estações bases.

4.2 Características das Transações Móveis

Segundo MADRIA (1998), as transações móveis podem ter que dividir as suas computações em conjuntos de operações, alguns dos quais poderiam ser executados em unidades móveis enquanto outros em estações base. Uma transação móvel compartilha os seus estados e resultados parciais com outras transações, devido a desconexão e mobilidade.

- O processo da transação pode ser migrado a um computador não-móvel se nenhuma interação do usuário for preciso.
- As transações móveis requerem suporte das estações base para computações e para comunicações.
- Quando o usuário móvel se mover durante a execução de uma transação, esta execução deve continuar na célula nova. A transação parcialmente executada pode ter sua continuidade na estação base, de acordo com a instrução dada pelo usuário móvel. Para continuar esta transação no seu novo destino, são requeridos mecanismos diferentes.
- Como as unidades móveis se movem de uma célula para outra, os estados de transação, estados dos dados de dados acessados e as informações de local também se movem.
- As transações móveis são transações duradouras devido à mobilidade tanto dos dados como das unidades e devido às desconexões freqüentes.
- As transações móveis suportam e tratam a concorrência, recuperação, desconexão e consistência mútua dos objetos de dados replicados.

No caso de sistemas distribuídos, é executada uma transação distribuída simultaneamente em processadores múltiplos e conjuntos de dados. A execução das transações distribuídas é coordenada completamente pelo sistema inclusive aspectos como controle de concorrência, replicação e *commit* atômico. A transação móvel por outro lado, é executado consecutivamente por múltimplas estações de bases, e possivelmente em múltiplos conjuntos de dados, dependendo do movimento da unidade móvel. A execução da transação móvel não é assim completamente coordenada pelo sistema. O movimento da unidade móvel controla a execução.

Para suportar transações móveis, o modelo de processamento das transações deve acomodar as limitações da computação móvel, como incerteza da comunicação, vida de bateria limitada, baixa largura de comunicação e reduzida capacidade de memória. Computações móveis devem minimizar os abortos devido à desconexão. Operações em dados compartilhados têm que assegurar justeza de transações sendo executados nas unidades móveis e estações base. O bloqueio na execução de transações em unidades móveis ou na MSS deve ser minimizado para reduzir o valor da comunicação e aumentar concorrência. A unidade móvel deve ter autonomia local para dar suporte a transações móveis e permitir o processamento e o *commit* na própria unidade, apesar da desconexão temporária.

4.3 Modelos de Transações Móveis

Considerando as características das transações móveis que são impostas pelos modos de operação de uma unidade móvel, foram propostos vários modelos de transações móveis. Este modelos apresentam modelos de processamento das transações que acomodam as limitações da computação móvel.

4.3.1 Modelo de Transação baseado em Semântica – segundo WALBORN & CHRISANTHIS (1995)

O modelo de transação baseado em Semântica assume um ambiente cliente/servidor que está disposto nas MSS. Para assegurar a consistência dos dados compartilhados na presença de concorrência e falhas, são utilizadas transações de atualização e recuperação de dados entre unidades móveis e unidades estacionárias.

Uma grande parte de sistemas de processamento de transações pregam um pouco de conhecimento semântico para prover maior disponibilidade de dados e alcançar um alto grau de concorrência , assim como simplificar a recuperação na presença de falhas.

A semântica dos objetos depende das seguintes características:

- **Semântica de operações** – está relacionada aos efeitos de uma operação no estado de um objeto;
- **Valores de operações de entrada/saída** – se refere a ambas as direções do fluxo de informações de e para um objeto e a interpretação dos valores de entrada e saída;
- **Organização do objeto** – a organização abstrata de um objeto;
- **Uso do objeto** – como o objeto é usado e o que é feito com a informação extraída dele.

As três primeiras características do objeto são usadas para definir várias formas de comutatividade, que determinam a semântica se duas operações puderem ser executadas concorrentemente sem compromisso de serializabilidade. A última característica , o uso do objeto, define critérios de justeza quanto à aplicação específica que transcendem a comutatividade e a serializabilidade para permitir mais operações até mesmo para executar simultaneamente e assincronamente.

A propriedade semântica comumente mais utilizada é a comutação. Se duas operações comutam, seus valores de retorno são os mesmos, independente da ordem de suas execuções. Operações que comutam podem ser organizadas arbitrariamente em uma execução concorrente para efetuar a serializabilidade.

Operações *push* com valores idênticos de entrada, podem comutar, enquanto não há comutação, quando os valores forem diferentes.

Valores de saída também podem ser utilizados para determinar se duas operações estão em estado de comutação.

Podemos utilizar comutatividade para aumentar o acesso concorrente e simples recuperação de dados compartilhados na unidade móvel. Para objetos armazenados no servidor de banco de dados, se todas as operações de um objeto comutam com cada outra em todos os estados, então este objeto pode ser colocado no *cache* e pode ser manipulado na unidade móvel assincronamente, sem qualquer interferência do servidor de banco de dados. É exigido da unidade móvel, apenas a propagação periódica para o servidor das atualizações da transações móveis localmente consolidadas (operação *commit*)

Este modelo estendido à computação móvel aumenta a concorrência, explorando operações de comutatividade. Esta técnica requer para o *caching* uma porção grande do banco de dados ou mantém cópias múltiplas de muitos ítens de dados. Também podem ser utilizados fragmentos de objetos de dados para facilitar a transação baseada em semântica que são processadas em bancos de dados móveis. Cada fragmento de objeto de dados tem que ser colocado no *cache* independentemente e manipulados assincronamente.

Segundo ADIBA et al (2001), transações móveis são invocadas pela unidade móvel e, do ponto de vista do servidor de banco de dados, elas são duradouras por causa de demoras de comunicação. Unidades móveis pedem fragmentos incluindo dois parâmetros: critérios de seleção e condições de consistência. Os critérios de seleção indicam os dados para serem colocados no *cache* da unidade móvel e o tamanho de fragmento exigido. As condições de consistência especificam as restrições para preservar consistência nos dados inteiros. Fragmentação de dados executada no servidor permite uma boa granularidade para o controle de concorrência. A cópia-mestre exclusiva fornece cópias de fragmentos para as unidade móveis e as transações podem ser completamente executadas nestes. Um processo de reconciliação é executado pelo servidor quando a re-conexão acontecer. Este modelo pode ser usado com tipos diferentes de transação.

Neste modelo, o foco é o uso de informação de semântica do objeto para melhorar a autonomia da unidade móvel em modo desconectado. Esta abordagem usa a organização de objetos e semântica de aplicação para dividir dados grandes e complexos em fragmentos manejáveis menores do mesmo tipo.

4.3.2 Modelo de Transação baseado em *Clustering* – segundo PITOURA & BHARGAVA (1995)

O modelo de transação baseado em *Clustering* (agrupamento), assume um sistema completamente distribuído e é projetado para manter consistência de banco de dados. O banco de dados é dividido dinamicamente em agrupamentos, nos quais cada um se agrupa semanticamente com dados relacionados ou dados situados próximos. Um agrupamento pode ser distribuído em várias unidades fortemente conectadas. Quando uma unidade móvel está desconectada, ela se torna um agrupamento por si só. Para todo objeto são mantidas duas cópias, uma delas (versão rígida) deve ser globalmente consistente, e a outra (versão fraca) pode tolerar algum grau de inconsistência mas deve ser localmente consistente. A transação móvel é rígida ou fraca. Transações fracas acessam só versões fracas, considerando que rígido acessam versões rígidas (ADIBA et al - 2000).

A execução de transações rígidas acontecem quando as unidades móveis estiverem fortemente conectadas e transações fracas quando as unidades móveis estiverem desconectadas. São introduzidos dois tipos de operações nas unidades móveis quando desconectadas, leitura fraca e escrita fraca. Transações rígidas contêm padrões de leitura e escrita rígidas (operações rígidas). Quando a re-conexão é possível (ou quando a consistência de aplicação requer isto) um processo de sincronização executado no servidor de banco de dados permite ao banco de dados ser globalmente consistente.

Segundo SEYDIM (1999), um modelo aninhado-aberto é proposto para este modelo. O modelo é baseado em agrupar segundo a semântica ou proximidade dos dados situados, para junto formar um agrupamento. Os dados são armazenados ou colocados no *cache* da unidade móvel para apoiar suas operações autônomas durante as desconexões. É assumido um ambiente completamente distribuído, onde os usuários submetem transações de terminais móveis e fixos. Transações podem envolver dados remotos e dados armazenados localmente no dispositivo do usuário

Segundo PITOURA & BHARGAVA (1995), os ítens do banco de dados são divididos em grupos e estes são unidades de consistência, em que é exigido que todos os ítens de dados, dentro deste agrupamento, sejam completamente consistentes, enquanto que ítens que residem em agrupamentos diferentes podem exibir saltos de

inconsistências. Os agrupamentos podem ser construídos dependentes da localização física dos dados. Usando esta definição de localidade, os dados localizados podem ser considerados residentes em unidades fortemente conectadas, como pertencentes a um mesmo agrupamento, enquanto os dados que residem em unidades móveis desconectados ou remotos podem ser considerados como agrupamentos separados. Deste modo, pode ser criada uma configuração de um *cluster* dinâmico.

A natureza de desconexão voluntária também pode ser usada para definir o agrupamento. Então, podem ser criados *clusters* (agrupamentos) de dados explicitamente ou podem ser fundidos por uma provável desconexão ou conexão associada à unidade móvel. O movimento da unidade móvel também causará o lugar do agrupamento, pois quando entrar em uma célula nova, poderá mudar seu agrupamento também.

Por outro lado, agrupamentos de dados podem ser definidos usando a semântica de dados como os dados de local ou definindo um perfil do usuário. Dados de local que representam o endereço de uma unidade móvel, são dados que rapidamente variam e são replicados para muitos locais. Estes dados são freqüentemente imprecisos e quando atualizam suas cópias criam *overhead.* Isto pode não ter necessidade para prover consistência para este tipo de dados. Por outro lado, definindo perfis do usuário para a criação de agrupamento, pode ser possível diferenciar os usuários, baseado nas exigências de seus dados e aplicações. Por exemplo, pode ser considerado que dados que são acessados freqüentemente por algum usuário ou dados que são um pouco privado a um usuário, pertencem ao mesmo agrupamento independente do seu local ou semântica.

PITOURA & BHARGAVA (1994) definem a consistência completa a ser requerida para todos os dados dentro de um agrupamento e graus de consistência para dados replicados de diferentes agrupamentos. O grau de consistência pode variar dependendo da disponibilidade de largura da banda de rede entre agrupamentos, permitindo pequena divergência em disponibilidade. Isto proporcionará para aplicações a capacidade de adaptar. O banco de dados móvel é visto como um conjunto de ítens de dados que são divididos a um conjunto de agrupamentos.

Ítens de dados estão relacionados por várias restrições chamadas restrições de integridade, que expressam relações de ítens de dados que um estado de banco de dados tem que satisfazer. As restrições de integridade entre ítens de dados dentro do mesmo

agrupamento são chamadas de restrições intra-agrupamento e restrições entre ítens de dados de agrupamentos diferentes são chamadas restrições de inter-agrupamento. Durante a desconexão ou quando a conexão for fraca ou cara, os únicos dados que o usuário pode acessar, podem não satisfazer restrições de inter-agrupamento estritamente. A largura da banda disponível pode proporcionar para o usuário dados de variável nível de detalhe ou qualidade. Para maximizar o processo local e reduzir acesso à rede, é permitido ao usuário interagir localmente em um agrupamento de dados disponível de m-grau consistente, usando operações de leitura-fraca e escrita-fraca. Estas operações permitem ao usuário operar com a falta de consistência rígida que pode ser tolerada pela semântica das suas aplicações. Por outro lado, o padrão de leitura e escrita são chamadas de operações de leitura rígida e escrita rígida, para diferenciar de operações fracas.

Baseados nestas idéias, dois tipos básicos de transação estão definidos em PITOURA & BHARGAVA, (1995): transações rígidas e fracas. Como insinuam os nomes, transações fracas consistem só de operações de leitura fraca e escrita fraca e elas só acessam cópias de dados que pertencem ao mesmo agrupamento, podendo ser considerado local àquele agrupamento. Uma operação de escrita fraca em um item de dados lê um cópia localmente disponível que é o valor escrito pela última operação fraca ou rígida de escrita àquele agrupamento. Uma operação de escrita fraca escreve uma cópia local e não é permanente, a menos que esteja consolidada a fusão na rede. Igualmente, transações rígidas consistem só em operações de leitura e escrita rígida. Uma operação de escrita rígida esta definida como o que lê o valor do item de dados que é escrito pela última operação de escrita rígida, em que uma operação de escrita rígida escreve uma ou mais cópias do item de dados.

Operações fracas para a interface de banco de dados provêm aos usuários o acesso local a um agrupamento, emitindo transações fracas de dados consistentes e dados globalmente consistente emitidos por transações rígidas. Operações fracas suportam desconexão desde que um dispositivo móvel possa operar desconectado e contanto que as aplicações estejam satisfeitas com cópias locais. Usuários podem usar transações fracas para atualizar dados principalmente privados e transações rígidas para atualizar dados comuns altamente usados. Além disso, permitindo aplicações para especificar as suas exigências de consistência, pode ser alcançada com a utilização melhor da largura da banda.

4.3.3 Modelo de Transação *Reporting*

Segundo CHRYSANTHIS P. K. (1993), este modelo analisa transações aninhadas e transações aberto-aninhadas, mostrando as limitações de seus ambientes móveis, considerando um ambiente de banco de dados móvel como um sistema de multibancos de dados especial, com exigências específicas de transações na unidade móvel, que são consideradas como um conjunto de sub-transações. Neste modelo, é proposta a transação aberto-aninhada que suporta atômicidade, transações não-compensáveis e dois tipos adicionais,: *reporting* e co-transações .

Transações aninhadas fechadas consolidam de baixo para cima até a raiz. Conseqüentemente, uma sub-transação aninhada começa depois de seu pai e termina antes. E a consolidação da sub-transação é condicionada à consolidação de seu pai. O aninhamento aberto relaxa a restrição de atomicidade do nível superior da transação aninhada fechada, em que os resultados só podem ser vistos após a consolidação da transação pai. A transação aninhada aberta permite que seus resultados parciais sejam observados fora da transação (ÖSZU & VALDURIEZ, 2001). Enquanto em execução, as transações podem compartilhar os resultados parciais e parcialmente podem manter o estado de uma sub-transação móvel (executada na unidade móvel) em MSS.

Transação Aninhada: neste modelo se estende o conceito de atomicidade, permitindo o aborto de uma sub-transação sem que toda a transação seja abortada (FERREIRA & FINGER, 2000).

No sistema de transações aninhadas, as transações possuem uma estrutura interna. Esta estrutura pode ser descrita como hierarquia de sub-transações dispostas em formato de árvore. Todas as transações, menos uma, que é a primordial, são sub-transações de uma outra. A estrutura hierárquica determina o tipo de relacionamento entre transações: uma sub-transação que se encontra imediatamente abaixo de outra é chamada de filha desta; duas sub-transações filhas de uma mesma transação mãe são chamadas descendentes na hierarquia que liga a primeira à segunda. Na Figura 4.2 esta ilustrada uma transação aninhada.

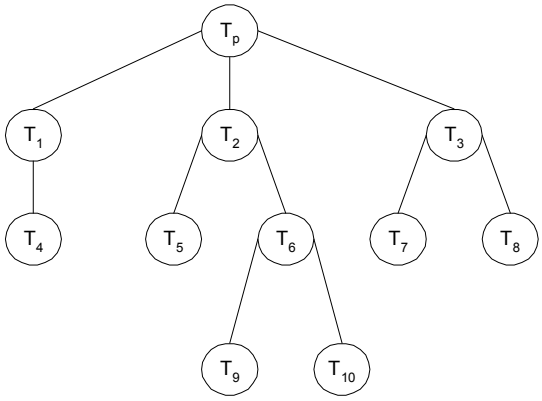

Figura 4.2 – Estrutura interna de uma transação aninhada

Uma transação móvel é estruturada como um conjunto de transações, algumas das quais são executadas na unidade móvel. São considerados que para limitações na unidade móvel se faz necessário o uso de uma unidade estacionária (MSS), por exemplo, para armazenar parte do estado da computação ou executar parte da computação. Transações aberto-aninhadas com sub-transações têm propostos quatro tipos:

- **Transações atômicas** estão associadas com os eventos {*Begin*, *Commit*, *Abort*}, tendo como propriedades padrão *abort* e *commit*.
- Transações **não-compensáveis** são componentes de transações que não são associados com uma transação de compensação (transação de compensação atua de modo inverso em relação à transação a qual ela está associada). Transações não-compensáveis podem ser consolidadas a qualquer hora, desde que elas não possam ser compensadas, pois não lhes são permitidos consolidar os efeitos em objetos quando eles consolidam. Transações não-compensáveis são estruturadas como sub-transações (como em transações aninhadas). Para consolidação é delegado tempo em seu pai para todas as operações que eles invocaram.
- **Transações *reporting* -** informam a outra transação alguns dos seus resultados a qualquer ponto durante a execução. Um relatório pode ser considerado como uma delegação de estado entre transações. Assim, as transações *reporting* são associadas com a primitiva da transação de Relatório, além das operações primitivas *Begin, Commit* e *Abort*.

- **Co-transações** se comportam como co-rotinas nas quais o controle é passado de uma transação para outra na hora de compartilhar os resultados parciais. Co-transações estão suspensas na hora de delegação e eles retomam a sua execução quando eles recebem um relatório. Assim, transações não-compensáveis são opostas às co-transações, pois estas não podem ser executadas concorrentemente.

5.3.4 Modelo de Transação *Pro-motion*

O modelo da transação *Pro-motion* propõe um sistema de processamento de transação móvel que suporta o modo desconectado. São introduzidos compactos para permitir execuções locais em unidades móveis. A informação necessária para gerenciar o compacto é encapsulada nele próprio. Para melhorar a autonomia e aumentar a concorrência, a semântica de objeto é usada na construção de compactos sempre que possível. Compactos são unidades básicas de *caching* e controle.

O sistema *Pro-motion* de processamento de transação móvel foi desenvolvido por WALBORN & CHRYSANTHIS e tem como alvo aplicações que migram no banco de dados existente e suportar o desenvolvimento de aplicações de banco de dados que envolvem acesso a dados móveis e sem fio. *Pro-motion* é um sistema que utiliza a arquitetura cliente/servidor.

Segundo WALBORN & CHRYSANTHIS (1997), as transações subjacentes processadas no modelo *Pro-motion* têm o conceito de transações *nested-split* (partidas-aninhadas). Transações partidas aninhadas é um exemplo de aninhamento aberto que relaxa a restrição de atomicidade do nível do topo das transações aninhadas fechadas, em que, uma transação aninhada aberta permite que seus resultados parciais sejam observados fora da transação.

Uma das questões principais quanto a transação processada localmente na unidade móvel, é a visibilidade e a permissão para novas transações verem uma transação não consolidada (dados sujos), que pode resultar dependências indesejáveis e o aborto em cascata. Porém, nenhuma atualização na unidade móvel desconectada pode ser incorporada no servidor de banco de dados, subseqüentemente transações usando os mesmos ítens de dados não podem proceder até que uma conexão aconteça e transação

móvel consolide. *Pro-motion* considera o sub-sistema móvel inteiro como uma transação extremamente grande, duradoura que é executada no servidor com uma sub-transação que é executada em cada unidade móvel. Cada uma destas sub-transações da unidade móvel, por sua vez, é a raiz de outra sub-transação partida-aninhada.

A infra-estrutura de *Pro-motion* foi construída sob a arquitetura de cliente-servidor de *multi-tier* com um agente móvel chamado de agente compacto, um servidor estacionário *front-end* chamado de gerente de compacto, e uma ordem de intermediário de gerentes de mobilidade (MSS) para ajudar administrar o fluxo de atualizações de dados entre os outros componentes do sistema. A infraestrutura de promotion é mostrada na Figura 4.3.

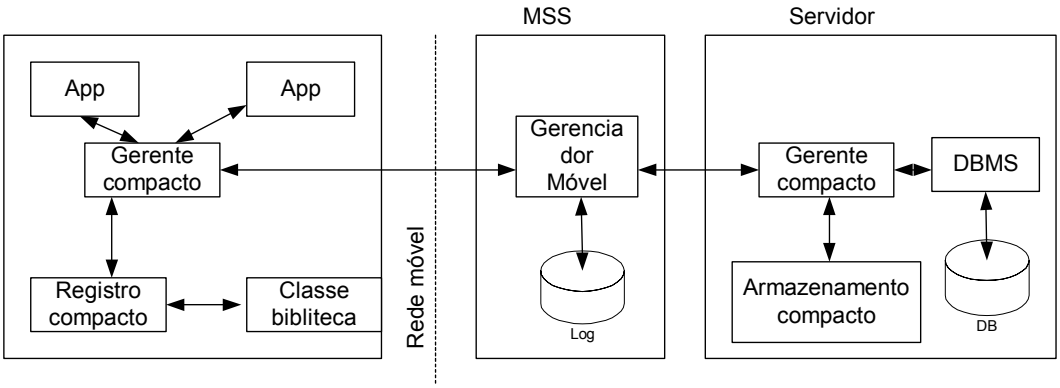

Figura 4.3 - Arquitetura do Sistema Promotion
Fonte: WALBORN & CHRYSANTHIS (1999)

Seu bloco fundamental é o compacto, o qual funciona como a unidade básica de replicação para *caching*, *prefetching* (pré-carregamento), e *hoarding* (acumulando).

Um compacto está definido como um pedido satisfeito para dados de *cache*, com suas obrigações, restrições e informação de estado. Representa um acordo entre o servidor de banco de dados e a unidade móvel, onde o servidor de banco de dados delega controle de alguns dados à unidade móvel para ser usado no processo de transação local. O servidor de banco de dados não tem a necessidade de estar atento as operações executadas por transações individuais na unidade móvel, bastando, ter atualizações periódicas de um compacto para cada um dos ítens de dados manipulados pelas transações móveis.

Segundo WALBORN (1997), no modelo os compactos são representados por objetos que encapsulam:

- o *cache* de dados;
- métodos para acesso dos dados no *cache*;
- informação sobre o estado atual do compacto;
- regras de consistência, que devem ser seguidas para garantir a consistência global dos ítens de dados;
- obrigações, com um *deadline*, que criam um salto de tempo para o qual os direitos dos recursos são segurados pela unidade móvel;
- métodos que provêem uma interface com que a unidade móvel possa gerenciar o compacto.

A estrutura principal do compacto é mostrada na Figura 4.4

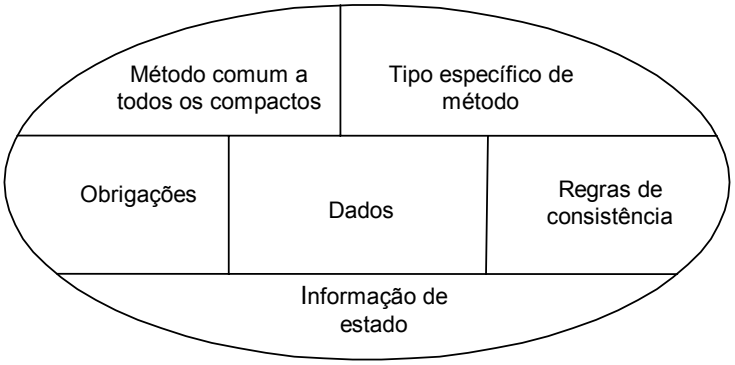

Figura 4.4 - Compacto
Fonte: WALBORN (1997)

O gerenciamento de compacto é um esforço cooperativo entre servidor e unidade móvel. Os compactos são obtidos do servidor de banco de dados via pedido da unidade móvel quando uma demanda real ou antecipada é criada. Os compactos são periodicamente atualizados com os resultados das transações processadas na unidade móvel. Quando há necessidade pela unidade móvel ou pelo servidor de banco de dados de mudanças, os compactos podem ser renegociados para redistribuir recursos e, quando a unidade móvel não precisa mais dos recursos, são devolvidos para o servidor de banco de dados e são apagados do armazenamento local.

Sempre que uma unidade móvel precisar de dados, esta envia um pedido para os dados no servidor. Se os dados estiverem disponíveis para satisfazer o pedido, o servidor cria um compacto que é transmitido a uma unidade móvel para prover os dados

e métodos que satisfazem a necessidade da execução da transação na unidade móvel. Caso os compactos já estiverem disponíveis na unidade móvel, evita-se esta comunicação que pode se tornar cara. Uma vez que a unidade móvel recebe o compacto, é armazenado em um registro de compacto usado pelo agente de compacto para localizar todos os compactos abertos e seus estados.

Cada compacto tem uma interface comum que é usada pelo agente de compacto para gerenciar os compactos listados no registro do compacto. O conjunto básico de métodos para gerenciar os compactos inclui:

Inquire – inquire, de forma que o compacto possa obter o estado atual do compacto.

Notify – notifique, de forma que o compacto possa receber notificação quando o estado da unidade móvel muda.

Dispatch – despache, usado para processar operações no compacto emitido pela transação que é executada na unidade móvel

Commit – consolide, faz com que as operações de uma transação específica se tornem permanentes no banco de dados.

Abort - aborto, abandona as mudanças feitas por um dado compacto para uma determinada transação.

A flexibilidade oferecida pelos compactos permite ao Promotion suportar vários esquemas de replicação. O gerente de mobilidade toma conta de transmissões entre agentes. São executadas transações de unidade móvel até mesmo em modo conectado. Um processo de sincronização é executado pelo agente compacto e o gerente compacto na re-conexão. Este processo confere o modo compacto através de transações localmente consolidadas. Se os compactos preservam consistência global, então uma consolidação global é executada.

4.3.5 Modelo de Transação com Replicação em *two-tier* (duas-fileiras)

O modelo de replicação *two-tier* assume dois tipos de nós (unidades): nós móveis que estão desconectados parte do tempo e nós base, que estão sempre conectados. Os nós móveis armazenam uma réplica do banco de dados e podem originar transações tentativas; um nó móvel pode ser o mestre de alguns ítens de dados.

Nós base armazenam uma réplica do banco de dados. A maioria dos ítens de dados são mestres de nós base.

Segundo GRAY et al (1996), os nós móveis têm duas versões de ítens de dados: versão mestre e versão tentativa. Versão mestre é o valor mais recente recebido do objeto mestre. A versão do objeto mestre é a versão de mestre. Porém, quando os nós estão desconectados ou com réplicas relaxadas podem ter versões mais velhas.

Versão tentativa: é criada com o valor mais recente devido a atualizações locais, esta é mantida como um valor tentativo. O objeto local pode ser atualizado por meio de transações tentativas. Este modelo suporta dois tipos de transações: básicas e tentativas. As transações básicas são executadas acessando versões mestre do dado (esquema de replicação mestre-*lazy*), e transações tentativas são executadas acessando versões tentativas (cópias locais). Transações tentativas podem executar atualizações na unidade móvel no modo desconectado.

Transação tentativa: trabalham em modo desconectado, lêem dados mestres e produzem dados tentativos locais. A transações tentativas poderão ser convertidas em transações básicas quando houver a conexão com os nós base. Transações tentativas têm que seguir uma regra de escopo: eles podem envolver objetos mestres em nós base e objeto mestre em nós móvel e podem originar uma transação (transações escopo). A idéia é que o nó móvel e todos os nós base entrarão em contato quando a transação tentativa é processada como uma transação básica realmente. Assim uma transação real poderá ler a cópia de mestre de cada item de dado no escopo.

A transação básica gerada por uma transação tentativa pode falhar ou pode produzir resultados diferentes. A transação básica tem um critério de aceitação: um teste resultante das *output* têm que passar por uma ligeira diferença do resultado da transação básica para ser aceitável.

Se uma transação tentativa falhar, o nó que originou e a pessoa que gerou a transação são informados da falha e por que falhou. Aceitação de falha é equivalente ao mecanismo de reconciliação dos esquemas de replicação de *lazy-group* (grupo relaxado). As diferenças são: (1) o banco de dados do mestre sempre é convergido—não há nenhuma ilusão de sistema; e (2) o nó que originou somente deve contactar um nó base para descobrir se uma transação tentativa é aceitável.

Conforme GRAY et al (1996), considerando o caso desconectado, se um nó móvel desconectou um dia atrás, ele terá uma cópia base dos dados a partir de ontem. Gerou

transações tentativas naqueles dados base e nos dados locais da cópia mestre utilizada pelo nó móvel. Estas transações geraram versões de dados tentativas no nó móvel. Essas atualizações tentativas são todas visíveis ao nó móvel.

Quando um nó móvel conecta a um nó base, o nó móvel:

1. descarta suas versões de objeto tentativas desde que eles sejam *refreshed* dos mestres;
2. envia atualizações de réplica para qualquer objeto mestre, tanto para o nó móvel quanto para o nó base;
3. envia todas suas transações tentativas (e todos seus parâmetros de input) para o nó base para ser executado na ordem na qual eles consolidaram no nó móvel;
4. aceita atualizações de réplica do nó base (este é o padrão de replicação do mestre relaxado), e
5. aceita notificação do sucesso ou fracasso de cada transação tentativa.

O nó base é a outra classe das duas classes. Quando contatado por uma nota móvel, o nó base:

1. envia transações de atualização de réplica atrasadas ao nó móvel;
2. concorda com atualizações atrasadas de transações para objetos móvel-mestre do nó móvel;
3. aceita a lista de transações tentativas, as suas mensagens de entrada e seus critérios de aceitação. Reprisa cada transação tentativa na ordem em que consolidou no nó móvel. Durante este re-processamento, a transação básica lê e escreve cópias de mestre do objeto que usa um modelo de execução de mestre-relaxado. A regra do escopo assegura que a transação básica só acessa dados mestres originado do nó móvel e do nó base. Assim, cópias de mestre de todos os dados na transação escopo estão disponíveis à transação básica. Se fracassar os critérios de aceitação da transação básica, esta é abortada e uma mensagem diagnóstico é devolvida ao nó móvel. Se os critérios de aceitação requerem que a transação básica e a tentativa tenham saídas idênticas, então transações subseqüentes que lêem resultados tentativos escritos por T também falharão. Por outro lado, ps critérios de aceitação da transação mais fraca é possível;

4. depois que o nó base consolida uma transação básica, ele propaga a réplica desta transação atualizada relaxada e envia a todos os outros nós à réplica. Este é o padrão mestre-ralaxado;
5. Quando todas as transações tentativas forem re-processadas como transações básicas, o estado do nó móvel é convergido com o estado básico. Figura 4.5

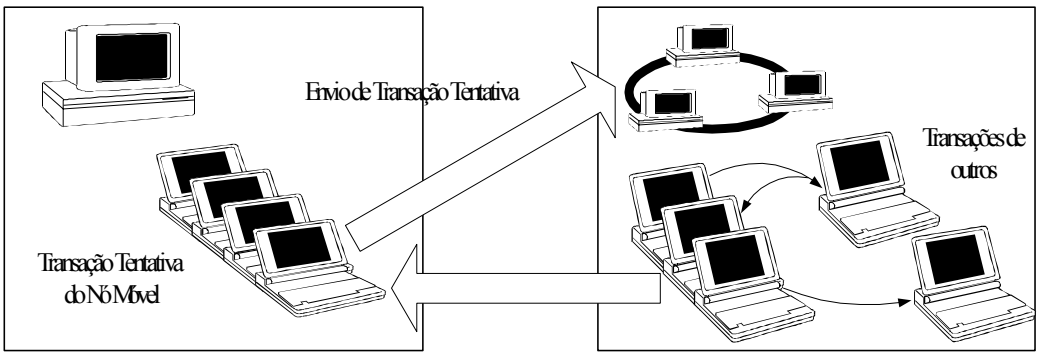

Figura 5.5- Execução de Transações Tentativas e Base em Replicação *Two-tier*
Fonte: GRAY et al (1996)

As propriedades fundamentais do esquema de replicação *two-tier* são:

1. nós móveis podem fazer atualizações de banco de dados tentativas;
2. transações básicas executam através de única-cópia a serializabilidade, assim o estado do sistema base mestre é o resultado de uma execução de serializabilidade;
3. uma transação fica durável quando a transação básica completar;
4. réplicas de todos os nós conectados convergem com o estado de sistema básico;
5. se todas as transações comutam, não há nenhuma reconciliação.

A taxa de reconciliação para transações básicas será zero se toda as transações comutarem. A taxa de reconciliação é dirigida pela taxa à qual as transações básicas falharam quanto aos critérios de aceitação. O processamento da transação básica pode produzir resultados diferentes dos resultados tentativos. Isto é aceitável para algumas aplicações.

Ambos, *clustering* e replicação de *two-tier* requerem um gerente de transação na unidade móvel para prover a execução da transação local, controle de concorrência, gerenciamento de *log* e recuperação.

Se a transação tentativa completa prosperamente e passa no teste de aceitação, então o sistema de replicação assume que tudo está bem e propagam as réplicas atualizadas. Os usuários estão cientes que todas as atualizações são tentativas até que a transação se torna uma transação básica. Se a transação básica falhar, o usuário pode ter que revisar e re-submeter uma transação. O programador deve projetar as transações para ser comutativas e ter critérios de aceitação para descobrir se a transação tentativa concorda com os efeitos de transação básicos.

4.3.6 Modelo de Transações *Prewrite*

Segundo MADRIA (1998), no modelo *Prewrite*, a idéia principal é dividir a execução de transação entre a unidade móvel e o servidor de banco de dados Com isso tenta aumentar disponibilidade de dados em unidade móvel, introduzindo uma operação *prewrite* (pre-escrita) além da escrita padrão. Um *prewrite* faz o valor de dados visível a pré-*commit* antes da consolidação da transação móvel. Atualizações permanentes no banco de dados são executadas depois pelas operações *write* e *commit*. São mantidas duas variantes de dados: *prewrite* e write. Variante de *Prewrite* trabalha com declaração de dados futuros, mas pode ser estruturada ligeiramente, diferente do valor correspondente *write*, por exemplo, em um objeto de documento de tipo o *prewrite* é um abstrato e o *write* é o documento completo.

A transação móvel é executada na unidade móvel, mas são feitas atualizações permanentes ao servidor de banco de dados por um gerente de dados (GD). Prewrite assegura que, delegando a responsabilidade de escrita ao banco de dados, o processo de transação será reduzido na unidade móvel. Três operações *(prereads, prewrites e precommit)* serão executadas pela unidade móvel. *Writes* e *reads* ordinárias permanentes são feitas pelo gerente de dados. A MSS tem anotado as capacidades e mantém relação íntima com o gerente de dados. Na Figura 4.6 é demonstrada a arquitetura móvel utilizada pelo *prewrite*.

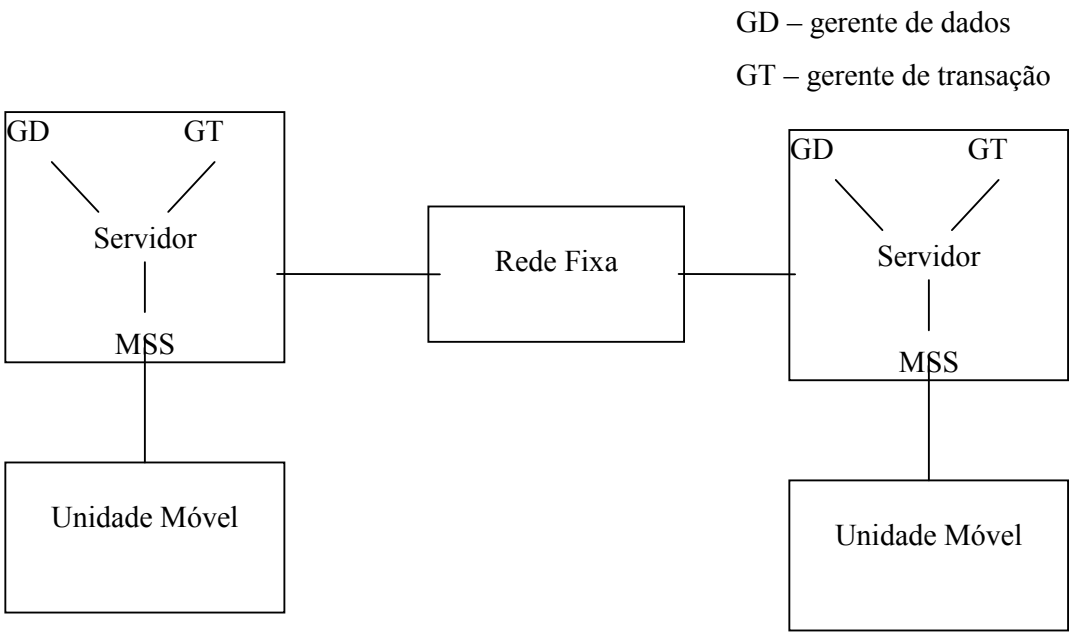

Figura 4.6 – Arquitetura Móvel *Prewrite*
Fonte: MADRIA (1998)

Cada operação de *prewrite* em uma unidade móvel deixa visível o valor da transação que eventualmente será escrito no banco de dados da MSS. Uma vez que os objetos de dados requeridos são *reads* ou pre-*reads* e são computados *prewrite* na transação, um *pre-committ* é executado na unidade móvel.

Uma transação requer um *read* e pre-*read* de todos os objetos de dados pedidos antes do *pre-committ* porque depois que uma transação libera um bloqueio para uma operação de *prewrite*, não pode adquirir um bloqueio para uma operação *de read* devido à condição de *two phase locking* (bloqueio em duas fases). Depois do *pre-commit*, as operações de *prewrites* são visíveis a outros processamentos de transações tanto na unidade móvel quanto na MSS. Os *prewrites* são controlados a nível de gerente de transação (GT), considerando que a escrita física é controlada à nível de gerente de dados.

Neste modelo de transação móvel, uma transação começa sua execução na unidade móvel. Quando uma transação chegar à unidade móvel, os pedidos de *read* da transação são processados na unidade móvel no caso de ter objetos de dados consistentes no *cache*. Caso contrário, a unidade móvel envia pedido de algumas *reads* (para qual a unidade móvel não tem nenhum objeto de dados consistente no *cache*) para a MSS. Quando uma transação de *read* chega a MSS, é analisada a transação e retorna

os valores de *prewrite* em resposta. Se um prewrite não está disponível, os valores *write* são devolvidos. A MSS identifica que o valor de retorno é um valor de *prewrite.* No caso da transação precisar do final da *write*, tem que iniciar uma *read* novamente. Uma vez que todos valores pedidos são devolvidos da MSS para a unidade móvel, a transação é *pre-committed* na unidade móvel depois de declarar todos os valores de *prewrite* dos objetos de dados. A execução de uma transação *pre-committed* é então trocada da unidade móvel para a MSS completar a sua execução permanente que envolve a atualização do banco de dados. Este procedimento move a aparte cara da execução da transação para a rede estática. Na MSS, a transação na verdade atualiza todos os objetos de dados para qual os valores de *prewrite* foram declarados mais cedo e foram consolidados depois disso (ver Figura 4.7). Assim, o valor de *prewrite* do objeto de dados é visível para a MSS e para outras unidades móveis naquela célula, antes do objeto de dados ser atualizado na MSS (MADRIA & BARGHAVA, 1997).

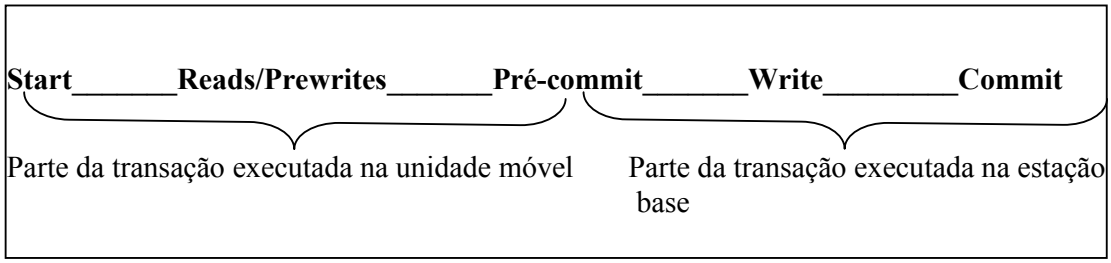

Figura 4.7 – Transação *Prewrite*
Fonte: MADRIA & BARGHAVA, 1997

São armazenados *Prewrite*s no *workspace* privado da transação na unidade móvel e uma vez que a transação é *pre-committed*, estes valores são movidos para a MSS

4.3.7 KT (Transações de Canguru)

De acordo com SEYDIN (1999), este modelo de transação móvel foi definido direcionando-se ao comportamento de movimento da transação. Transações móveis são nomeadas como Transações Canguru pelo fato de pular de uma estação base à outra, com o movimento das unidades móveis. O modelo captura o comportamento de

movimento e o comportamento de dados que reflete no acesso a dados localizados no banco de dados ao longo da rede estática.

Este modelo assume que há um *Data Access Agent* (Agente de Acesso a Dados), que serve de interface entre uma transação móvel e o sistema básico. Cada estação base abriga necessariamente um AAD. O agente de acesso a dados não mantém informação sobre a localização da unidade móvel, embora precise de tal informação. Quando recebe um pedido de transação de um usuário móvel, o AAD redireciona o pedido para a estação base que contém o dado ou sistema básico requisitado. Quando uma unidade móvel passa para outra estação base, o AAD na nova estação recebe as informações sobre as transações daquela unidade móvel em andamento na velha estação. Desta forma, o AAD age como um Gerente de Transação Móvel, o que inclui manter informações sobre as transações ativas que passaram pelo AAD e manter um *log* da operações requisitadas a partir deste AAD.

Quando uma transação móvel se mudar para uma célula nova, o controle da transação pode mover-se ou pode ser retida no local original. Se permanecer no local original, mensagens terão que ser enviadas a qualquer hora do local original à estação base atual onde a unidade móvel pede informação. Se a função de gerente de transação move com a unidade móvel, um *overhead* destas mensagens pode ser evitado.

O modelo está baseado no conceito de transação tradicional que é uma sucessão de operações incluindo, **leia, escreva, comece transação, fim da transação, consolide e aborte** a transação. A estrutura básica é de uma transação local (TL) para um DBMS (Sistema de Multibanco de Dados).

O modelo de transação canguru focaliza o movimento da unidade móvel durante a execução de transações. As transações são geradas na unidade móvel e são completamente executadas em um Sistema de Multibanco de Dados (MDBS) na rede. Por outro lado, Transações Globais (TG) podem consistir em qualquer sub-transações vistas como transações locais de algum DBMS (Sub-transação Global - GST) ou sub-transações viram um seqüência de operações que podem ser globais (GTs). Uma propriedade de salto é somada para modelar a mobilidade das transações (Figura 4.8).

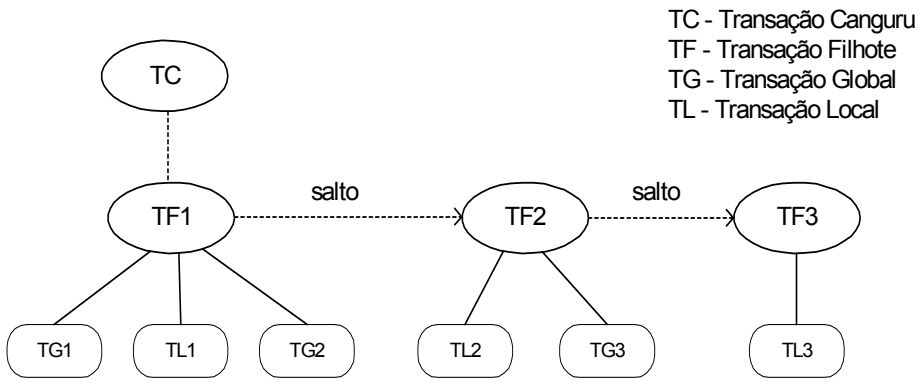

Figura 4.8 - Estrutura básica da Transação Canguru
Fonte: SEYDIN (1999)

Segundo DUNHAM et al (1997), uma transação global e uma transação filhote são diferentes uma da outra. Só a transação filhote é uma parte da transação canguru e deve ser coordenada por um AAD em algum local da estação base. Uma transação canguru tem um único número de identificação que consiste no número de estação base e o número de seqüência único dentro da estação base. Quando a unidade móvel se mover de uma célula a outra, o controle da transação canguru muda para um AAD novo em outra estação base. O AAD no local da nova estação base cria uma transação filhote nova, como o resultado do processo de *handoff*. Transações filhotes também têm números de identificações em seqüência onde o ID de uma transação filhote tem o ID da transação canguru e a seqüência do número.

A mobilidade do modelo de transação é capturada pelo uso de transações partidas. A transação filhote velha está assim consolidada independentemente da transação filhote nova. Se acontecer uma falha de qualquer transação filhote, a transação canguru inteira poderá ser desfeita, compensando qualquer transação previamente completa.

Existem dois modos de processamento para uma transação canguru: o **modo compensatório** e o **modo partido**. Quando uma transação canguru é executada no modo compensatório, o aborto de uma transação filhote fará com que seja desfeita esta transação filhote e de todas as outras transações filhotes na transação canguru. Transações filhotes já consolidadas terão que ser compensadas.

No modo partido, o padrão é, se uma transação filhote falhar, outras transações filhotes já consolidadas não serão compensadas. Transações filhotes ativas podem terminar ou serem desfeitas, e a decisão é deixada a cargo de cada transação filhote.

O Gerente de Transação Móvel (GTM) mantém uma tabela de Estado de Transação na estação base e o DAA mantém o estado dessas transações. Também é mantido um *log* local no qual o GTM escreve os registros que são precisos para propósitos de recuperação, mas o *log* não contém qualquer registro relacionado à recuperação das operações de banco de dados. A maioria dos registros no *log* são relacionados ao estado da transação canguru e algumas informações de compensação.

4.3.8 MDSTPM – *Multidatabase Transaction Processing Manager Arquiteture* (Arquitetura de Gerencia de processo de Transação de Multibanco de Dados)

Segundo YEO & ZASLAVSKY (1994), o modelo MDSTPM propõe um *framework* para suportar submissões de transação de unidades móveis em um ambiente de multibanco de dados. Um sistema de multibanco de dados (MDS) está definido como um sistema de banco de dados distribuído e integrado, consistindo em um sistema de gerenciamento de vários componentes autônomos de banco de dados. Cada um dos sistemas de banco de dados de componente subjacentes é localmente responsável pela administração de transações. Para facilitar a execução de transações globais, uma camada adicional de software precisa ser implementada por meio da qual permissões de escalonamento e coordenação de transações sejam coordenadas pelo sistema de gerenciamento de banco de dados heterogêneo.

A contribuição relativa a desconexões da unidade móvel é a implementação da *Message and Queuing Facility* (Mensagem e Fazendo fila com Facilidade (MFF)) que administra o intercâmbio de mensagem entre a unidade móvel e o sistema de multibanco de dados em rede normal. Uma MDSTPM é assumida em cada unidade, tanto móvel como base, sobre um sistema gerenciador de banco de dados(SMBD) local existente. O processamento local é responsabilidade do SMBD local. O MDSTPM coordena a execução de transações globais, gera programação e coordena compromissos (ADIBA et al, 2001).

De acordo com YEO & ZASLAVSKY (1994), o MDSTPM consiste nos componentes seguintes:

- O Gerente de Comunicação Global (GCG) é responsável pela geração e;
- Administração de filas de mensagem dentro do seu próprio local. Adicionalmente, também comunica, entrega e troca estas mensagens com seus locais semelhantes e unidades móveis na rede;
- O Gerente de Transação Global (GTG) coordena a submissão de sub-transações global para seus locais pertinentes. O Coordenador de Gerente de Transação Global;
- (CGTG) é o local onde a transação global é iniciada. Todos os GTGs que participam;
- para aquela transação global é conhecido como GTMPs. O GTG pode ser um Subgerente Global *Scheduling* (SGS) ou um Subgerente Global de Concorrência (SGC). O SGS é responsável pela *scheduling* de transações globais e sub-transações. O SGC é responsável por aquisição de exigências necessária para o controle de concorrência para a execução próspera de transações globais e sub-transações. O GTG é responsável pela programação e compromisso de transações globais ;
- Gerente de Transação Local (GTL) é responsável pela execução e recuperação de transações executadas localmente;
- O Gerente de Recuperação Global (GRG) coordena a consolidação e a recuperação de transações globais e sub-transações depois de uma falha. Assegura que os efeitos da consolidação das sub-transações globais são escritas no banco de dados local subjacente ou nenhum dos efeitos de sub-transações globais abortadas é escrito. Também usa-se o protocolo *write-ahead*, de forma que o efeito para o banco de dados é escrito imediatamente sem ter que esperar pela sub-transação global completar ou consolidar;
- Gerente de Interface Global (GIG) coordena a submissão de pedido/resposta entre o MDSTPM e o gerente de banco de dados local nos quais podem estar sendo executado um sistema de banco de dados relacional ou um sistema de banco de dados orientado a objeto. Este componente provê função de

extensibilidade inclusive a tradução de um pedido de SQL para um pedido de consulta em linguagem orientada a objeto.

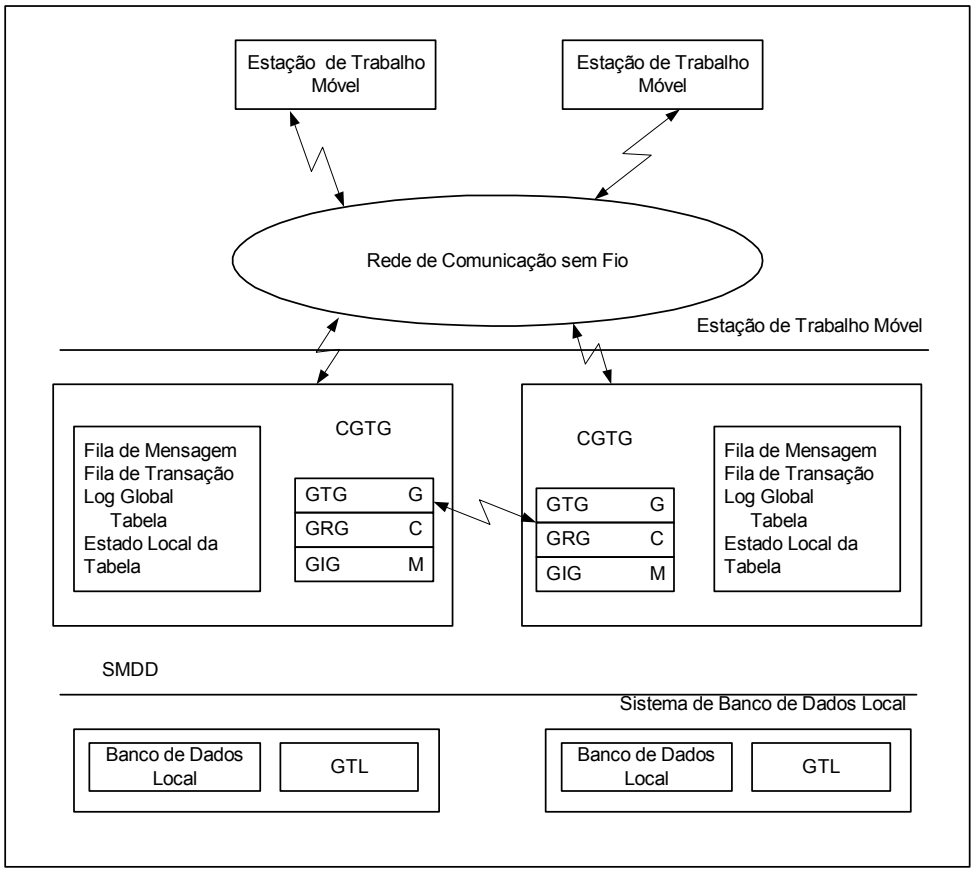

Figura 4.9 – Modelo MDSTPM

É assumido que cada cliente móvel submete uma transação a um agente coordenador (Figura 4.9). Uma vez que a transação foi submetida, o agente coordenador programa e coordena sua execução em nome do cliente móvel. Unidades móveis podem desconectar voluntariamente da rede antes de ter qualquer transação associada completa. A arquitetura deve satisfazer o seguinte:

- prover o gerenciamento de *framework* de transação de forma que os usuários e programas aplicativos poderão acessar dados por transparência de múltiplo locais;
- aumentar a concorrência de banco de dados e disponibilidade de dados pela adoção de um controle de concorrência distribuído e mecanismo de recuperação que preserve a autonomia local;

- implementar o conceito de extensibilidade para suportar vários sistemas de banco de dados dentro do *framework* de forma que os componentes possam cooperar com banco de dados orientados a objeto ou relacional;
- prover um ambiente onde a transação proposta processe o componente operando independentemente e transparente ao DBMS local;
- que incorpora o conceito de computação móvel pelo uso de estações móveis de trabalho.

A abordagem utiliza o gerenciamento de unidades móveis e transações globais submetidas a estas unidades móveis para fazer parte do MDS durante suas conexões com um respectivo coordenador de seu nodo. Uma vez que uma transação global foi submetida, o coordenador local pode *schedule* e coordenar a execução da transação global em nome da unidade móvel. Deste modo, a unidade móvel pode desconectar da rede sem esperar uma transação global para completar.

Segundo ADIBA et al (2001), em MDSTPM como em KT, a maneira que são obrigadas as propriedades ACID depende de cada DBMS em cada local. Cada MDSTPM é responsável para coordenar suas transações globais. Para unidade móvel, por causa de desconexões, é designado um coordenador com antecedência. Então, uma vez que uma estação base submeter uma transação global, pode desconectar e executar algumas outras tarefas sem ter que esperar pela transação móvel para consolidar. O coordenador da unidade gerencia a transação móvel em nome da unidade móvel.

4.3.9 Transações Compensatórias

Segundo SILBERSCHATZ et al (1999), o trabalho de uma transação de compensação é reverter o efeito de uma transação consolidada de uma forma que não dependa do que aconteceu ao banco de dados entre o momento em que a ação foi executada e o momento em que a transação de compensação é executada.

Transações compensatórias são utilizadas para desfazer transações confirmadas ou transações ativas que afetam outras transações, sem ser preciso utilizar abortos em cascata. Estas transações violam dois princípios dos métodos tradicionais:

- Durabilidade – por meio da qual uma transação confirmada seja tratada como permanente e irreversível,e
- Isolamento – em que uma transação só deve ler dados de transações confirmadas.

Transações compensatórias são utilizadas por aplicações em que há transações muito longas e que ficariam impraticáveis, caso os princípios violados fossem seguidos rigorosamente. O mecanismo das transações compensatórias é aplicado no caso em que existe a necessidade de se permitir que transações leiam dados não-confirmados de outras transações.

4.4 - Considerações

Através deste capítulo, podemos verificar as arquiteturas de banco de dados utilizada pelas transaçoes móveis, e a arquitetura adicional que cada modelo requer a mais no específico banco de dados. Na tabela 1, pode-se observar o resumo dos modelos propostos, arquitetura de banco de dados utilizada e arquitetura adicional.

Tabela 1 – Arquiteturas Utilizadas pelos modelos de transações móveis.

Modelo	**Arquitetura do Banco de Dados**	**Arquitetura Adicional**
Pro-motion	Cliente/Servidor	Agente de Compacto
*Cluster*ing	Totalmente Distribuído	Gerente de Transação
Reporting	Multibanco de Dados	
Two-Tier	Cliente/Servidor	Gerente de Transação
Prewrite	Cliente/Servidor	Gerente de Transação/Gerente de Dados
Canguru	Multibanco de Dados	Agente de Acesso a Dados
MDSTPM	Multibanco de Dados Heterogêneo	Camada de Gereciamento de Transações
Semantics Based	Cliente/Servidor	

Também neste capítulo foram apresentados modelos de transações móveis que suportam a desconexão. Percebe-se no decorrer deste estudo que uma grande parte dos

modelos de transações móveis utilizam dois tipos de transações, as transações fracas e as fortes (tabela 2). Estas transações fracas são processadas em modo desconectado na unidade móvel e as transações estritas são processadas em modo conectado. As transações fracas podem se tornar estritas no momento da reconexão da unidade móvel a rede, reintegrando os dados e propagando as atualizações. Existem também os modelos que utilizam transações aninhadas que delegam a execução de operações a sub-transações, estas estariam em unidades móveis. Uma transação aninhada só terá êxito se todas as sub-transações forem executadas com sucesso

Na tabela 2 resume-se também em que local as transações são executadas.

Tabela 2 - Tipos de transações e modo de execução

Modelo	**Tipo de Transação**	**Execução na Unidade Móvel**	**Execução na Rede Fixa**
Pro-motion	Transação móvel	Transação móvel e *commit* local	Gerente de Compacto atualiza o banco e faz permanente o *commit* local
*Cluster*ing	Transações *weak e strict*	Transações *weak* e *commit* local – modo desconectado	Transações *stricts* e *commit* das transações fracas
Reporting	Sub-transações	Sub-transações	Transações Globais e sub-transações
Two-Tier	Transações tentativas e básicas	Transações Tentativas – modo desconectado	Transações básicas e consolidação de tentativas
Prewrite	Transações pré-*write* e pré-*read*, *read e write*	Transações pré-*write* e pré-*read* – modo desconectado	Transações *write* e *read* e consolidação de prés
Canguru	Sub-transações	Sub-transações	Coordena e executa a transação inteira

Modelo	**Tipo de Transação**	**Execução na Unidade Móvel**	**Execução na Rede Fixa**
MDSTPM	Transações Locais	Transações Locais	Coordena e executa multitransações
Semantics Based	Transação Móvel e *commit* local	Transação Móvel e *commit* local	Reintegra e atualiza os fragmentos

CAPÍTULO V

CONTROLE DE CONCORRÊNCIA

A implementação do controle de concorrência trata das propriedades de isolamento e consistência das transações. Quando o banco de dados suporta transações concorrentes o mecanismo de concorrência tenta encontrar um equilíbrio entre a concorrência das transações e a consistência do banco de dados.

Segundo CASANOVA & MOURA (1985), no controle de concorrência é atribuído a gerência de transações para escalonar as ações de uma transação de tal forma que seja processada corretamente. Por outro lado, cabe ao mecanismo de controle de concorrência arbitrar a intercalação das ações de transações diferentes de tal forma que todas as transações sejam processadas sem que uma interfira na outra.

Nas seções que seguem, serão conceituados serialização, métodos de controle de concorrência e os metodos utilizados nas transações móveis vistos no capítulo IV.

5.1 A Propriedade de Serialização de Transações

A execução de transações concorrentes deve ser realizada em modo isolado, não permitindo que, enquanto qualquer delas não termine sua execução, os dados atualizados por ela estejam disponíveis a outras transações. Considerar-se que a execução concorrente de uma transação deve deixar o banco de dados em um estado que possa ser alcançado por outra execução seqüencial em alguma ordem, problemas como atualizações perdidas serão sanados.

Uma escala de execução é definida sobre um conjunto de transações, aqui denominado por T.$\{T_1, T_2 \ldots, T_n\}$, e sobre este conjunto será especificado uma ordem de intercalação de execução dessas transações.

São definidos dois tipos de transação. Transação que lêem seus ítens de leitura constituem um conjunto de leitura (CL) e transações que gravam, seus ítens de dados gravados constituem seu conjunto de gravação (CG).

São definidos como operações conflitantes de transação se pelo menos uma entre duas ou várias tem a operação de gravar. Transações de leitura não causam conflitos. Pode-se salientar dois tipos de conflitos, quanto à concorrência da execução. Conflito **leitura-gravação** e conflito **gravação-gravação**. Caso as operações pertençam a duas transações distintas, então serão chamadas de transações **conflitantes**.

Segundo ÖSZU & VALDURIEZ (2001), se em um escalonamento as operações de várias transações não estão intercaladas, dizemos que seu escalonamento é serial. Dizemos que um escalonamento é serializável se, e somente se ele é equivalente de conflitos a um escalonamento serial.

5.2 Problemas quanto ao Controle de Concorrência em Banco de Dados Móveis

Com a mobilidade dos clientes móveis, que ora podem estar conectados e ora desconectados, bem como pela largura média da banda para transmissão de dados e ainda pelo poder de processamento menor do que de unidades fixas, as transações destes e entre estes bancos de dados devem ser tratadas de forma que a longa duração de uma transação não cause o aborto desta, bem como uma desconexão não possa ser considerado como uma falha.

O controle de concorrência em banco de dados móveis deve monitorar e controlar as execuções concorrentes, garantindo consistência do banco de dados, pois as aplicações em um ambiente de difusão de dados deve acessar dados consistente, ou seja, com restrições de integridade satisfeitas, além de acessarem sempre dados atualizados.

Pela mobilidade e transmissão em difusão (*broadcasting*), existem alguns fatores que dificultam a realização do controle de concorrência. Uma delas é a largura de banda estreita para cliente-servidor, que limita a capacidade de clientes enviarem solicitações ao servidor. Outros são os períodos de desconexão de clientes móveis, por limitação da

bateria, por deslocamento para uma área não coberta e que ocasionam transações de clientes móveis com longa duração.

5.2.1 *Deadlock*

Segundo DATE C.J.(2000), d*eadlock* é uma situação na qual duas ou mais transações estão em estado de espera simultânea, cada uma esperando que uma das outras libere um bloqueio antes de poder prosseguir. Se ocorrer um *deadlock*, é desejável que o sistema o detecte e o interrompa.

No contexto de banco de dados móveis, uma transação sendo executada por uma unidade móvel, pode ter um bloqueio sobre um item de dado x no servidor ou em uma outra unidade móvel, e outra transação de outra unidade precisa acessar este mesmo item de dado para poder prosseguir com a computação. Porém, a primeira transação que detém o bloqueio se desconecta, mas mantém o bloqueio. A segunda transação poderá esperar por um tempo indeterminado para tentar acessar o dado x.

Este tipo de *deadlock* pode ocorrer tanto na arquitetura cliente/servidor, banco de dados distribuído, multibanco de dados heterogêneo. Independente da arquitetura, este problema deve ser tratado.

5.2.2 Atualizações perdidas

Na pré-ordenação um valor é substituído por um valor mais novo. Se o valor atual do objeto já tem uma marca de tempo maior que a marca de tempo da atualização, a atualização está descartada. Porém, o esquema de marcas de tempo pode perder os efeitos de algumas transações porque aplica as atualizações mais recentes. Esquemas de marca de tempo são vulneráveis a atualizações perdidas.

Uma unidade móvel que trabalha com dados replicados e usa marcas de tempo para controlar a concorrência está propensa a este tipo de problema. Considere duas unidades móveis com dados replicados de uma cópia mestre que está no servidor. A unidade móvel 1 enquanto desconectado executou algumas transações no tempo 1, e a

unidade móvel 2 também executou transações no tempo n+1. Quando se reconectaram, o fizeram quase no mesmo instante. Quando foram reintegrar os dados na cópia-mestre, apenas a atualização mais nova (n+1) foi considerada.

5.3 Mecanismo de Controle de Concorrência

.

À medida que as transações solicitam leitura e gravações de ítens de dados do banco de dados, essas solicitações são repassadas a um escalonador. Na maioria das situações, o escalonador executará as leituras e gravações diretamente, primeiro chamando o gerenciador de *buffers*, se o item de dados desejado não estiver em um *buffer*. A Figura 6.1 demonstra o papel do escalonador.

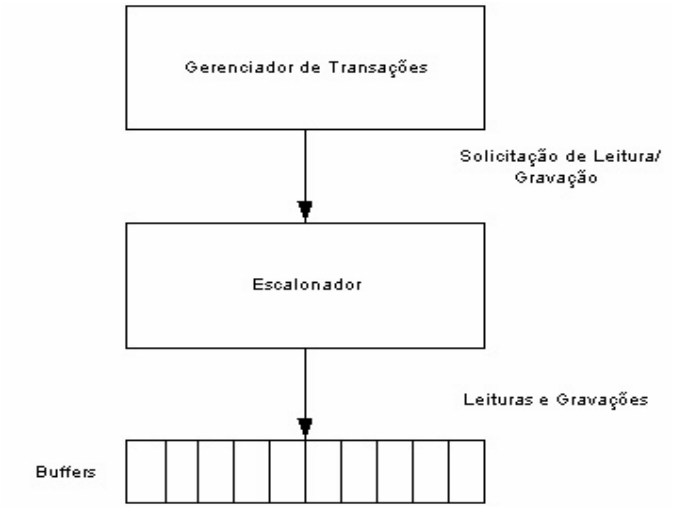

Figura 5.1 - O Papel do Escalonador.

Um escalonamento é uma seqüência ordenada no tempo das ações importantes executadas por uma ou mais transações. No controle de concorrência, as ações importantes de leitura e gravação têm lugar nos *buffers* da memória principal e não no disco (MOLINA et al , 2001).

O escalonamento deve examinar as condições sob as quais uma coleção de transações executadas de forma concorrente preservará a consistência do banco de dados.

Considerando a arquitetura mais comum para um escalonador, a que utiliza bloqueios sobre ítens de dados a fim de evitar um comportamento não serializável. O

controle de concorrência baseado em bloqueio, visa a sincronização das transações, assegurando que os dados compartilhados por operações conflitantes sejam acessados por uma operação de cada vez. Este acesso é assegurado pela associação de um "bloqueio" a cada unidade de bloqueio. Esse bloqueio é definido por uma transação antes de ser acessado e é redefinido no final e seu uso.

6.3.1 Controle de Concorrência Baseado em Bloqueio

Um escalonador que utiliza algoritmos de bloqueio, utiliza uma tabela de bloqueio para ajudá-lo a executar seu trabalho. Esta tabela de bloqueio informa, para todo o banco de dados, a transação que contém atualmente um bloqueio sobre esse item.

Existem dois modos de bloqueio: modo compartilhado e modo exclusivo. No modo exclusivo, uma operação obtém o bloqueio de um item de dado que está associado a uma unidade de bloqueio. Este bloqueio permite acesso ao item de dados apenas pela transação que o bloqueou. Caso uma outra transação requisite o bloqueio desta unidade de bloqueio, terá que entrar na fila de espera e só obterá tal bloqueio no momento em que a primeira transação liberar a unidade de bloqueio. Normalmente o bloqueio exclusivo é utilizado por operações de gravação.

O modo compartilhado é utilizado para a operação de leitura, pois não afetará o estado do item de dado. No modo compartilhado, duas ou mais operações de leitura podem adquir o bloqueio sobre uma mesma unidade de bloqueio.

No caso de existir uma operação de leitura e uma de gravação, o bloqueio não poderá ser compartilhado.

A matriz de compatibilidade demonstra quais as operações que podem ser compatíveis quanto ao bloqueio de um mesmo item de dados, num mesmo instante de tempo (Tabela 3).

Tabela 3 - Matriz de Compatibilidade de bloqueios

Operações	**Bloqueio de Leitura**	**Bloqueio de Gravação**
Bloqueio de Leitura	Há compatibilidade	Não há compatibilidade
Bloqueio de Gravação	Não há compatibilidade	Não há compatibilidade

5.3.1.1 Bloqueio em duas fases

Os algoritmos de controle de concorrência baseados no bloqueio em duas fases, executam transações em duas fases: fase de crescimento, em que a transação obtém o bloqueio e acessa o item de dados, e a fase de contração, na qual o bloqueio é liberado.

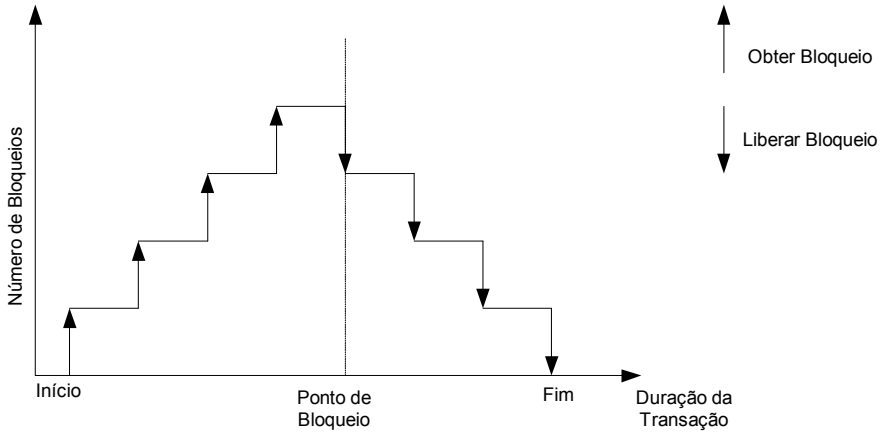

Figura 5.2 - Gráfico de Bloqueio 2PL
Fonte: ÖSZU & VALDURIEZ (2001)

O gerenciador de bloqueio libera o bloqueio logo que o acesso ao item de dados é concluído. Com esta liberação, outras transações que estão na fila de espera podem obter o acesso a este item de dado. Isso faz com que aumente o grau de concorrência.

5.3.2 Controle de Concorrência baseado em Pré-ordenação

Algoritmos baseados em timbre de hora selecionam a *priori* uma ordem de serialização e executam as transações de acordo com ela. Com isso, os algoritmos baseados em marca de tempo não tentam manter a serialização por exclusão mútua.

Segundo MOLINA et al (2001), para atribuir uma "marca de tempo" a cada transação, o gerenciador de transações atribui a cada transação T_i uma única marca de tempo $ts(T_i)$ no momento em que uma transação notifica ao escalonador que está se iniciando. Duas abordagens para gerar marcas de tempo podem ser seguidas:

- um modo de criar marca de tempo é usar o relógio do sistema, desde que o escalonador não opere tão rápido que possa atribuir marcas de tempo a duas transações em um único pulso do relógio;

- outro modo é o escalonador manter um contador. Toda vez que uma transação se inicia, o contador é incrementado por 1, e o novo valor se torna a marca de tempo da transação.

Independente do método usado para geração de marcas de tempo, o escalonador deve manter uma tabela de transações atualmente ativas e suas marcas de tempo.

Com este método, é preciso associar a cada elemento do banco de dados, duas marcas de tempo e um bit adicional:

a) RT (x), a hora de leitura de x, que é a marca de tempo mais alto de uma transação que leu x.

b) WT (x), a hora de gravação de x, que é a marca de tempo mais alto de uma transação que gravou x.

c) C (x), o bit de consolidação para x, que é verdadeiro se e somente se a transação mais recente a gravar x já foi consolidada.

Onde existe um escalonamento de 3 transações, T_1, T_2 e T_3 que acessam a três elementos do banco de dados A, B e C, é indicado as marcas de tempo das transações e a hora das leituras (R) e gravações (W) dos elementos. É suposto que, no início, cada um dos elementos do banco de dados tenha uma marca de leitura igual e de gravação igual a 0. As marcas de tempo são atribuídas quando as transações entram em contato com o escalonador e notificam que estão começando. No entanto, mesmo T_1 sendo o primeiro acesso não tem a marca de tempo menor. Possivelmente, T_2 foi a primeira a notificar o escalonador de seu início, e T_3 fez o mesmo logo em seguida, com T_1 notificando por último (MOLINA et al ,2001). Um exemplo de escalonamento por marcas de tempo é mostrado na Figura 5.3

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0 WT = 0	RT = 0 WT = 0	RT = 0 WT = 0
R_1(B)				RT = 200	
	R_1(A)		RT = 150		
		r_3(C)			RT = 175
w_1 (B)				WT = 200	
w_1 (A)			WT = 200		
	W_2(C) Abortar;				
		W_3(A)			

Figura 5.3 – Exemplo de Marcas de Tempo
Fonte: MOLINA et al ,2001

T_1 lê B. Considerando que a hora de gravação de B é menor que a marca de tempo de T_1, essa leitura é realizável e tem permissão para acontecer. A hora real de B é definida como 200, a marca de tempo. As outras duas ações de leitura também são válidas e resultam na definição da hora de cada elemento do banco de dados como a marca de tempo da transação que fez a leitura.

Depois, T_1 grava B. Como a hora de leitura de B é menor que a marca de tempo de T_1, a gravação é realizável. Quando é realizado, a hora de gravação de B é aumentada para 200, a marca de tempo da transação T_1 que fez gravação. Logo, T_2 tenta gravar C. Entretanto, C já foi lido pela T_3 que, teoricamente, foi executada na hora 175, enquanto T_2 teria gravado seu valor na hora 150. Portanto, T_2 resultaria num comportamento fisicamente irrealizável, e T_2 deve ser revertida.

Como última etapa há a gravação de A por T_3. Como a hora de leitura de A, 150, é menor que o timbre de hora de T_3, 175, a gravação é válida. Porém, já existe um valor posterior de A armazenado nesse elemento do banco de dados. Desse modo, T_3 não é revertida, mas também não grava seu valor.

5.4 Métodos de Controles Concorrência

Segundo ÖSZU & VALDURIEZ (2001), nos métodos tradicionais de controle de concorrência existe a classificação de controle de concorrência otimistas e pessimistas, em que os pessimistas têm a visão de que muitas transações entrarão em conflito entre si e os otimistas, de que não haverá um número grande de transações que entrarão em conflito entre si.

Na Figura 5.4 está demonstrada a seqüência de fases de um método de controle de concorrência pessimista sobre uma transação.

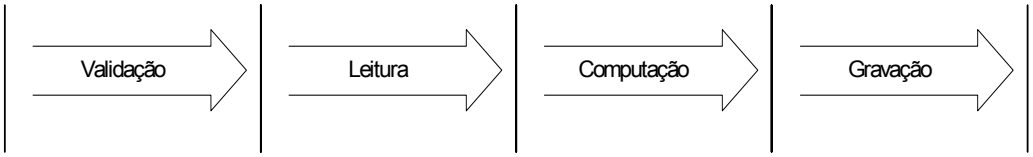

Figura 5.4 – Fases de Execução de Transação Pessimista

Na Figura 5.5 está demonstrada a seqüência de fases de um método de controle de concorrência otimista sobre uma transação

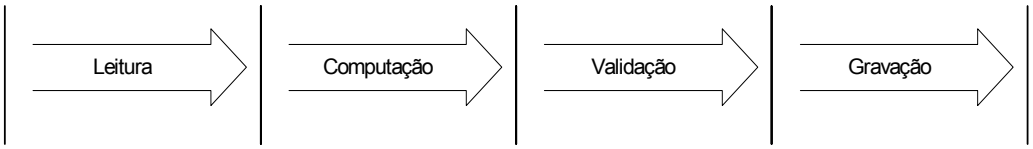

Figura 5.5 – Fases de Execução de Transação Otimista

5.5 Métodos de Controle de Concorrência Utilizados em Transações Móveis

Os métodos de controle de concorrência otimistas são os mais utilizados, pois são mais relaxados por considerarem a quase inexistência de conflitos. Estes utilizam os métodos de bloqueio de timbre de hora, mas com relaxamento quanto a consistência dos dados e o isolamento.

Como as unidades móveis em certos momentos podem estar desconectadas, e as transações móveis têm a necessidade de continuar a computação, os métodos

pessimistas não são utilizados com rigidez, pois podem levar a *deadloks,* com freqüência

5.5.1 *Clustering*

De acordo com PITOURA & BHARAT (1995), *clustering* usa dois tipos básicos de unidades de transações relativos a consistência:

a) transações que consistem somente em operações de leitura fraca e escrita fraca, que são chamadas de transações fracas;

b) transações que consistem somente em operações de leitura rígida e escrita rígida, que são chamadas de transações rígidas.

Para gerenciar o isolamento, *Clustering* propõe nova tabela para soluções de conflitos. *Clustering* utiliza o bloqueio em duas fases e propõe quatro tipos de bloqueio que correspondem às operações fracas e estritas (WR, WW, SR, SW). São propostas quatro tabelas de conflitos para bloqueio compatíveis. O projeto da função h utiliza as tabelas de conflito para refletir operações estritas em versões fracas, dependendo da necessidade da consistência da aplicação. Exemplo: requer consistência estrita traduzindo uma escrita estrita em um objeto estrito de escrita para todas as suas cópias. Conseqüentemente, um bloqueio SW não é compatível com qualquer outro bloqueio. Transações fracas liberam seus bloqueios para *commit* local e transações estritas para *commit* (PITOURA & BHARAT 1995).

A transação (T) é uma ordem parcial (OP, <), onde OP é um conjunto de leituras fracas ou estritas, escritas fracas ou estritas, aborto e *commit* local são executados pela transação e representam sua ordem de execução.

A ordem parcial tem que especificar a ordem de operações de dados conflitantes e têm que conter exatamente uma operação de aborto ou *commi*t, que será o último na ordem.

Duas operações fracas (rígidas) de dados conflitam se elas acessarem a mesma cópia de um item de dados e pelo menos uma delas, fraca ou rígida, é uma operação de escrita. A Tabela 4 mostra a matriz de compatibilidade das transações fracas e estritas.

Tabela 4 – Matriz de Transações Fracas e Estritas

	Leitura-fraca	Leitura-rígida	Escrita-fraca	Escrita-rígida
Leitura-fraca			Não compatível	Não compatível
Leitura-rígida				Não compatível
Escrita-fraca			Não compatível	Não compatível
Escrita-rígida	Não compatível	Não compatível	Não compatível	Não compatível

Uma operação de leitura fraca no item de dado x (W_*Read*[x]) lê uma cópia disponível local de x. Este valor é gravado pela última operação de gravação fraca ou estrita no *cluster*. Uma operação de gravação (W_*Write*[x]) grava em uma cópia local e não é permanente a menos que *committed* na fusão da rede. Um operação de leitura estrita (S_*Read*[x]) lê um valor de x gravado pela última operação que gravou estritamente. Uma operação de gravação estrita (S_*Write*[x]) grava uma ou mais cópias de x.

Transações fracas têm dois pontos para consolidar, um *commit* local no agrupamento associado e um *commit* implícito global depois da fusão de agrupamento. Atualizações feitas no *commit* local de transações fracas só são visíveis a outras transações fracas no mesmo agrupamento, mas não são visíveis a transações rígidas antes da união ou transações locais são consolidadas globalmente.

5.5.1.1 Consistência na Configuração do *Cluster*

Segundo PITOURA & BHARAT (1994), em um determinado *cluster* que tem execuções concorrentes de operações fracas e estritas, o critério de correção é baseado em gráfico de caracterização. Um escalonador intra-*cluster*, é uma observação da execução entrelaçada de transações que incluem *commit* local de transações fracas e *committ* global de transações estritas.

Primeira condição: estado em que os gerentes de transação traduzem cada operação submetida por uma transação em operações apropriadas nas cópias de dados;

Segunda condição: estado em que o escalonador intra-*cluster* preserva a ordenação estipulada por cada transação;

Terceira condição: estado em que o escalonador registra a ordem de execução de operações conflitantes;

Quarta condição: estado em que uma transação não pode ler uma cópia a menos que seja previamente inicializado;

Quinta condição: estado em que se uma transação escreve um item de dados x antes de ler x, então tem que escrever a mesma cópia de x que subseqüentemente lê;

Sexta condição: indica que para transação estrita, se uma escrita é traduzido para um escrita em uma cópia de dados num *cluster* Cl_i, todas as outras escritas desta transação também têm que escrever as cópias correspondentes no *cluster* Cl_i. Esta condição é necessária para assegurar que transações fracas não vejam resultados parciais de uma transação estrita.

Dado um escalonamento S, a projeção de S em transações estritas é o escalonamento obtido de S, apagando todas as operações fracas, e a projeção de S em um *cluster* Cl_k é o escalonamento obtido de S, apagando toda a operação de S que não acessa Cl_k.

Uma correta execução de transações fracas e estritas deve manter m-consistência entre *clusters* e consistência rígida dentro de cada *cluster*.

5.5.2 *Prewrite*

De acordo com MADRIA & BHAGAVA (1997), *prewrite* utiliza três tipos de operações que são executadas na unidade móvel (*pré-read, pré-write, pré-commit*) que equivalem a transações fracas. Já as operações fortes como *write*, *read* e *commit* são executadas pelo MSS.

Uma operação de *prewrite* também pode ser executada concorrentemente com outra transação *write*, desde que *prewrites* sejam gerenciados a nível de gerente de transação e as *writes* a nível de gerenciador de dados. Na primeira fase do controle de concorrência que controla as operações de *prewrite* e *pre-read*, é executada pelo gerente de transações na MSS. Na segunda fase, o controle de concorrência que controla as operações de *write* e *read* é executado pelo gerente de dados. O gerente de dados só é acessado quando houver execução de atualizações no banco de dados.

Desde que os valores de *prewrite* são visíveis depois do pre-*commit*, o tipo de bloqueio segurado para o *prewrite* é convertido em bloqueio para a operação de escrita. Depois do *pre-commit*, outra transação pode pegar o bloqueio. O bloqueio adquirido para uma operação não é liberado depois do *pre-commit* porque o bloqueio é em duas fases.

Neste contexto de *prewrite*, são usados bloqueios para todos os tipos de operações. Para a operação *read* é usado *read-lock*, para *pre-read* é usado *pre-read-lock*, *prewrite-lock* é usado para *prewrite* e *write-lock* para *write*. Um *prewrite-lock* conflita com *pre-read* e *prewrite-locks*, porém, não conflita com *read-locks* e *write-locks*. Um *pré-write-lock* não pode ser liberado depois do *pré-commit*, pois a transação ainda precisa adquirir um bloqueio de *write* para as atualizações .

Um bloqueio de *pre-write* adquirido para uma transação *pre-committed* é convertido para um bloqueio de *write*, contanto que nenhuma outra transação segure bloqueios conflitantes. Uma vez que *prewrite-lock* é atualizado para *write-lock*, a mesma transação não pode adquirir qualquer outro bloqueio. Porém, bloqueios *pre-re*ad podem ser adquiridos por outros para acessar valores *pre-writ*e.

Segue um protocolo de Bloqueio onde:

Pre-read-lock (x) concede o pedido *pre-read-loc*k para a transação T em X se nenhuma outra transação segurar um *prewrite-lo*ck em X.;

Read-lock (x) concede o pedido *read-lock* para a transação T em X se nenhuma outra transação segurar um write-lock em X.;

Prewrite-lock (x) concede o pedido *prewrite-lock* para a transação T em X se nenhuma outra transação segurar um *prewrite-lock* ou *pre-read-l*ock em X.;

*Write-l*ock (x) atualiza uma *prewrite-lock* em X segurado por uma transação T para *write-lock* se

Begin
Se a fila de *write-lock* em X estiver vazia então
Begin
Se a transação T é *pré-commited* e não há outra transação que segure um *read-lock* ou *write-lock* em X então
Converta *prewrite-lock* para *write-lock*;
End;
senão

```
    Begin
        ponha a transação T na fila de espera write-lock para X;
    End;
End.
```

Segundo MADRIA & BHARGAVA (1997), as transações consolidam em ordem serializável de transações. Na história da execução é decidida a ordem de ação de pre-*com*mit. A uma transação não é permitido abortar depois *pre-commit*.

Assim, os *prewrites* provêem execução não-rígida sem abortar em cascata. No caso da transação *pre-commit* ser forçada a abortar devido a razões dependentes do sistema, como queda de sistema, a transação pode reiniciar quando o sistema retornar.

Para reiniciar uma transação que falhou na *pre-commit* do último estado consistente, são usados valores dos *logs* de *prewrite* e *write* que podem ser salvos em armazenamento estável. Nesse modelo, é relaxada a propriedade de isolamento entre propriedades ACID, como o *prewrite* é visível a outros depois *pre-commit,* mas antes da transação estar finalmente consolidada na MSS. A durabilidade do *prewrite* é garantido quando é executado o *pre-commit* (MADRIA & BHARGAVA, 1997).

5.5.2.1 Escalonamento Serializável

Uma operação executada por uma transação T_i em um objeto de dados x é denotado por $r_i(x)$, *write* como w_i (x), *pré-read* como $pr_i(x)$ e *prewrite* como pw_i (x). Pré-*commi*t é denotado como pc_i , *commit* como c_i e aborto como a_i. Uma história serial é consecutiva onde, para toda transação T_i e T_j , qualquer ou todas as operações executadas por T_i precede todas as operações executadas por T_j ou vice-versa. Duas histórias H e H′são equivalentes se são definidas sobre o mesmo conjunto de transações e se p e q conflitam e $p <_H q$ e $p <_{H'} q$. Existe uma história serializável de conflitos se ela for equivalente a alguma história serial (MADRIA & BHARGAVA, 1997).

5.5.3 *Pro-motion*

De acordo com WALBORN & CHRYSANTHIS (1997), *pro-motion* trabalha com acordo entre a unidade móvel e o servidor de banco de dados. Por considerar o sub-sistema móvel inteiro como uma grande transação, logo nas unidades móveis são executadas sub-transações. *Pro-motion* pode ser considerada como transações aninhadas divididas.

Segundo WALBORN & CHRYSANTHIS (1999), *Pro-motion* utiliza uma escala de 10 níveis para caracterizar a correção de uma determinada transação. O padrão para *Pro-motion* é o relaxamento baseado nos graus de isolamento definidos no padrão SQL ANSI. Cada método em um compacto (que corresponde a uma operação nos dados do compacto) é nomeado um nível de isolamento num modo de indicar se a operação é uma leitura ou gravação. Qualquer operação que possa modificar o valor de um item de dados vai ser categorizada como operação de escrita (incremento ou decremento). O compacto pode ter múltiplos métodos, podendo empregar para um item de dado, diferentes níveis de correção. Ex.: em um contrato de compacto pode ter nível 8 de isolamento (serializável), para métodos de incremento e decremento, e nível 3 para uma leitura relaxada. Também um compacto pode ter mais de um nível de isolamento para a mesma operação para prover métodos alternados. Ex.: um compacto pode prover uma operação de *read-strict*, a qual somente completa se o item for bloqueado e o método *read-relaxed,* o qual retorna um valor apesar da condição do item de dado. A Tabela 5 mostra a matriz de conflitos entre subtransções e transações.

Tabela 5 – Matriz de transações *Pro-motion*

Operações	**Bloqueio de Leitura**	**Bloqueio de Gravação**
Bloqueio de Leitura	Há compatibilidade	Não há compatibilidade
Bloqueio de Gravação	Não há compatibilidade	Não há compatibilidade

Como *Pro-motion* usa a arquitetura aninhada dividida, o protocolo de bloqueio em transações aninhadas obedecem às seguintes regras:

- antes de realizar uma operação p_i [x], uma transação deve obter o bloqueio pl_i [x];

- a transação só obtém o bloqueio de pl_i [x] se nenhuma outra transação Tj no sistema possuir um bloqueio conflitante ql_j[x]; caso contrário, a transação é posta em espera;
- ao abortar, todos os bloqueios de uma transação são liberados;
- uma transação só se confirma se todas as suas filhas já terminaram (confirmadas ou abortadas);
- ao se confirmar, todos os bloqueios de uma transação passam para sua mãe.

Quando a unidade móvel se reconecta à rede, começa a ressincronização. A ressincronização busca as mudanças executadas na unidade móvel para o servidor. Esta ressincronização requer alguns estágios, dependo da validade dos compactos contidos no *cache* da unidade móvel. São feitas tentativas para incorporar as mudanças oriundas de todas as transações *committed* localmente na unidade móvel. Porém, pode não ser possível cometer toda transação ilegível ao servidor. O procedimento contingente para cada transação que não pode cometer no final, é executar uma compensação

Segundo WALBORN & CHRYSANTHIS (1999), no primeiro estágio de ressincronização, o registro do compacto acha os IDs (identificadores) de todos os compactos que foram modificados quando a unidade móvel esteve desconectada. Se todos os compactos são válidos, a ressincronização pode ser completa e podem passar ao estágio final. Porém, caso algum compacto esteja expirado, um trabalho adicional deverá ser efetuado antes da ressincronização.

No segundo estágio, o registro de compacto transmite os IDs dos compactos modificados expirados, tenta renovar cada compacto e estende seu *deadline* (prazo final). Caso nenhuma outra entidade tenha bloqueado ou modificado os dados do compacto, o gerenciador de compacto pode restabelecer um compacto e validar do lado da unidade móvel, como se o compacto nunca tivesse expirado. Se todos os compactos expirarem, serão restabelecidos e a atualização pode ser feita na sua totalidade e a ressincronização procede para o estágio final.

Porém, se os compactos modificados não podem ser restabelecidos, os valores originais válidos ou restabelecidos terão que ser obtidos do gerente de compacto. Os compactos que não podem ser restabelecidos são inválidos. Neste momento, um evento de *log* pode ser refeito para todas as transações cometidas. Se uma transação lê ou

modifica um compacto inválido, a transação é considerada abortada e todos os compactos modificados pela transação abortada são marcados como indisponíveis, assim como as transações que lêem transações indisponíveis são abortadas e os compactos marcados como indisponíveis. Quando um evento de *log* for completamente refeito, os compactos passam a ser válidos e podem ser incorporados ao servidor de banco de dados.

No último estágio da ressincronização, o registro de compacto inspeciona o conjunto de compactos para os compactos que são válidos e modificados. Para cada compacto consultado, é gerado um evento OP que é retornado ao compacto do lado do servidor para trazer os dados do servidor em acordo com os dados da unidade móvel. Quando todas as operações forem comunicadas, a unidade móvel manda mensagem de *commit* para o compacto gerente.

No servidor, o conjunto de operações de atualização no processo de ressincronização é dividido em uma única e separada transação, que está comitada no servidor. Além disso, o gerenciador de compactos tenta manter os bloqueios em todos os ítens comitados. Se os bloqueios são retidos pelo gerente de compacto, no lado do servidor, os compactos associados e os compactos no lado da unidade móvel permanecem válidos e utilizáveis. Se os bloqueios não podem ser mantidos, o compacto no lado do servidor e os compactos no lado da unidade móvel são invalidados. Uma vez que a ressincronização é completada, os compactos são liberados e a unidade móvel volta ao estado de *hoarding*.

5.5.4 *Semantics-based*

Segundo WALBORN & CHRYSANTHIS (1995), no modelo *semantic-based*, a serializabilidade é mantida através do protocolo de controle de concorrência de bloqueio em duas fases, quando as transações locais têm acesso ao *cache* de fragmentos.

Pode ser utilizada a comutatividade para aumentar o acesso concorrente e simples recuperação de dados compartilhados na unidade móvel. Para os objetos armazenados no servidor de banco de dados, se todas as operações de um objeto comuta com cada outro em todos os estados, então este objeto pode ser colocado no *cache* e pode ser manipulado numa unidade móvel assincronamente sem qualquer coordenação do

servidor de banco de dados. Da unidade móvel é exigido a propagação periódica para o servidor das atualizações da transação móvel.

A comutatividade pode ser usada nos métodos de *caching* que empregam controle de concorrência para assegurar a coerência do *cache*.

O método de garantia transacional é projetado para melhorar o acesso concorrente e agregar ítens explorando a estrutura dos objetos, baseado no estado da comutatividade e restrições de integridade. O método de garantia explora o fato de os ítens agregados serem valores numéricos que representam a quantidade de ítens trocáveis. Considerando que todos os artigos são trocáveis, a quantidade pode ser dividida entre várias unidades móveis, baseada em seus pedidos. A consistência de dados é assegurada limitando a quantidade de ítens no *cache*, de decremento e incremento que provêem comutação e certas condições de limites que derivam das restrições de integridade.

A consistência dos dados captura a justeza da perspectiva do objeto no banco de dados. A exigência de consistência de dado varia de consistência estrita (definido para serializabilidade) e consistência eventual.

Consistência eventual denota uma divergência temporal ou de espaço, da consistência estrita e estende o que pode ser expresso como "aumento de inconsistência".

Conforme WALBORN & CHRYSANTHIS (1995), as transações móveis que podem acomodar inconsistência espacial, agrupando objetos dinâmicos, baseado em grau de inconsitência e dois tipos de operações de leitura e escrita: *weak-read, weak-write, strict-read* e *strict-write*. Onde operações de leitura e escrita estrita têm a mesma semântica que operações de leitura e escrita invocadas pelas transações tradicionais ACID. Uma leitura fraca retorna o valor do ojeto que está no *cache* local escrito por uma escrita-rígida ou uma escrita fraca. Uma operação de escrita fraca atualiza um objeto que está no *cache* local, que poderá se tornar permanente em uma fusão de agrupamento se a escrita fraca não estiver em conflito com qualquer operação de leitura estrita ou escrita estrita.

5.5.4.1 Fragmentação

Considerando que no ambiente móvel é preciso espalhar os dados entre as unidades fixas e móveis da rede, a consistência destes fragmentos se torna

indispensável. Os objetos grandes são divididos em fragmentos menores do mesmo tipo. Com a divisão apropriada, uma unidade móvel pode ter no seu *cache* uma partição de um objeto de um certo tamanho, minimizando as exigências de armazenamento na unidade móvel. Além disso, estes fragmentos podem ser unidades de reconciliações de atualizações, ou unidades de consistência.

Uma cópia-mestre de cada objeto reside em um servidor de banco de dados estacionário. A unidade móvel especifica a granularidade de um objeto a ser colocado no *cache*, restrições de uso, usando operação de divisão.Uma operação de fusão é provida para permitir que transações liberem e incorporem fragmentos no *cache*, usando a cópia mestre.

Quando uma unidade móvel requer um acesso a um objeto, a unidade móvel envia um pedido de *cache* ao servidor de banco de dados, invocando uma operação de divisão com dois parâmetros: critério de seleção e condições de consistência. O critério de seleção especifica o objeto a ser colocado no *cache* e o tamanho exigido para a partição do objeto. Quando esta partição for colocada no *cache* da unidade móvel, é *log*icamente removida da cópia mestre do objeto e é somente acessível pela transação da unidade móvel. A parte restante da cópia-mestre não é afetada e fica acessível ao servidor.

As condições de consistência especificam restrições nos fragmentos que precisam ser satisfeitas, como operações permissíveis e restrições no seu valor de entrada, condições no estado do objeto. Quanto ao suporte para consolidação unilateral de transações executadas na unidade móvel, tem que reter os efeitos das operações quando houver a fusão dos fragmentos. Freqüentemente a ordem de recombinação será ditada pela estrutura original do objeto ou as operações executadas em cada fragmento.

Esta abordagem focaliza noções semânticas que permitem execuções concorrentes em fragmentos independente de um objeto, que podem estar no *cache* da unidade móvel.

5.5.5 *Two-tier Replication*

A abordagem do modelo *two-tier* trabalha com dois tipos de transações, básicas e tentativas, e dois tipos de cópia de dados, versão mestre e versão tentativa. Transações básicas só trabalham em dados de mestre e produzem novos dados de mestre. Estas

transações envolvem no máximo um nó móvel conectado e pode envolver vários nós básicos. Transações tentativas são executadas em modo desconectado e resultam da execução sobre a cópia da versão mestre (versão tentativa) (ADIBA et al, 2001).

Segundo GRAY et al (1996), na replicação de mestre, este nomeia um dono(nó) a cada objeto. O dono armazena o valor atual correto do objeto. As atualizações são primeiro feitas pelo dono e então propagadas a outras réplicas. Objetos diferentes podem ter os donos diferentes.

Quando uma transação quiser atualizar um objeto, envia uma chamada de procedimento remota (RPC) para o nó que possui o objeto.

Para haver serializabilidade, uma ação de leitura deveria enviar uma RPC bloqueio de leitura para os mestres de qualquer objeto que lê.

Aqui é assumido que o nó que originou a transação radiodifunde as atualizações de réplica a todas as réplicas escravas depois que a transação de mestre consolidar. E também envia uma transação escrava para cada nó escravo. Atualizações de escravo são pré-ordenadas para assegurar que todas as réplicas convirjam ao mesmo estado final.

Se o registro de marca de tempo da réplica for mais novo que um registro de marca de tempo da atualização de réplica, a atualização é "passada" e pode ser ignorada. Alternativamente, cada nó de mestre envia atualizações de réplica para escravos na ordem da seqüência do *commit*. Replicação de mestre-*lazy* não é apropriado para aplicações móveis. Um nó que quer atualizar um objeto deve ser conectado ao dono de objeto e deve ser participante em uma transação atômica com o dono.

Como com sistemas não relaxados, sistemas de mestre-relaxado não têm nenhum fracasso de reconciliação, pois conflitos são resolvidos esperando pelo *deadlock*. Se for ignorada a mensagem de demora, a taxa de *deadlock* para um sistema de replicação de mestre-relaxado é semelhante a um sistema de um nó único com taxas de transação mais altas.

Transações de mestre relaxado operam em cópias de mestre de objetos. Os nós que contém tempos para muitas transações concorrentes de mestre, são calculados em aproximadamente $nó^2$ (tempos de nós ao quadrado) de tempo para muitas transações de atualização de réplica. As transações de atualização da réplica não importam, elas são transações de trabalho de *background* (fundo). Eles podem abortar e podem reiniciar sem afetar o usuário. Assim, o fator principal é o freqüente *deadlock* das transações de

mestre. Este é o melhor comportamento da replicação de grupo-relaxado. A replicação de mestre relaxado envia menos mensagens durante a transação básica e assim completa mais depressa. Mas, todos estes esquemas de replicação têm como aborrecimento a taxa de *deadlock* ou reconciliação quando há muitos nós.

Segundo GRAY et al (1996), a replicação de mestre relaxado requer contato com mestres de objeto e assim não é muito usável através de aplicações móveis.

O método utilizado para que transações possam ser executadas às atualizações, inclusive com concorrência é a pré-ordenação.

Na pré-ordenação um valor é substituído por um valor mais novo. Se o valor atual do objeto já tem uma marca de tempo maior que a marca de tempo da atualização, a atualização está descartada.

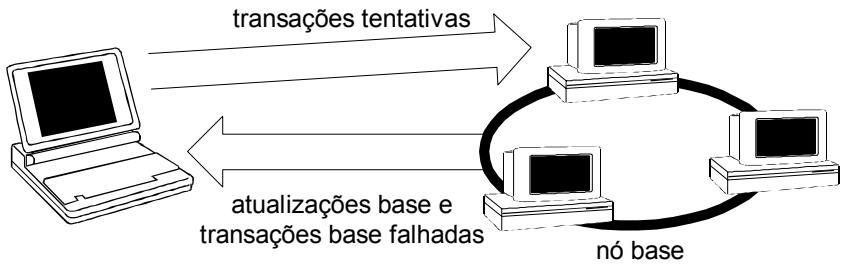

Figura 5.6 – Arquitetura de *Two-tier Replication*
Fonte: GRAY et al (1996)

No entanto, se convergência fosse a única meta, o método de marcas de tempo seria suficiente. Mas o esquema de marcas de tempo pode perder os efeitos de algumas transações porque aplica as atualizações mais recentes. E este, no controle de concorrência, é um problema chamado de problema de atualizações perdidas. Esquemas de marca de tempo são vulneráveis a atualizações perdidas.

A convergência é desejável, mas o estado convergido deveria refletir os efeitos de todas as transações comprometidas. Em geral, isso não é possível a menos que técnicas de serializações globais sejam usadas.

Em certo casos, as transações podem ser designadas a comutarem, de forma que o banco de dados termine no mesmo estado consistente, não importando que ordem de execução de transação é escolhida. Neste caso, seria marcas de tempo, com comutatividade.

Segundo GRAY et al (1996), a solução parece ser, usar semântica de dados (marcas de tempo, e transações de comutatividade), combinado com um esquema de replicação de *two-tier*. Replicação de *two-tier* apóia nós móveis e combina os benefícios de um esquema *lazy-master-replications* (replicação relaxada de mestre) e um esquema de atualização local.

5.5.6 MDSTPM

A arquitetura de gerência de processo de transações de multibanco de dados utiliza como abordagem, para gerenciar estações de trabalho móveis e as transações globais submetida, o fato de ter estas estações de trabalho móveis fazendo parte do MDS durante sua conexão com seus respectivos coordenadores de nós. Com isso, quando uma transação global é submetida, o coordenador local pode escalonar e coordenar a execução da transação global em nome da estação de trabalho móvel.

Segundo YEO & ZASLAVSKI (1994), a razão principal para esta estratégia é:

- o usuário da estação de trabalho móvel pode desconectar da rede e pode executar algumas outras tarefas sem ter que esperar pela transação global para completar;

- os computadores são conectados um ao outro com uma comunicação de transmissão segura em rede e, assim, menos suscetível a falhas da rede .

Um dos mecanismos usado para comunicação de interprocessos em um ambiente de computação distribuído é *Remote Procedure Call* (chamada de procedimento remoto (RPC)). Esta estratégia é análoga a uma subrotina chamada por um programa principal, com parâmetros que são passados ao processamento no nó para dirigir o processamento pedido. Igual a uma implementação que implicaria que os eventos que estão acontecendo com sincronismo com o visitante, teriam que esperar pelo controle a ser devolvido antes de continuar seu processo.

Uma abordagem alternativa é *Message e Queuing Facility* ((MQF - mensagem e fazendo fila) que é proposto para facilitar a implementação da estratégia global. Uma *Mobile Workstatio*n (MWS - estação de trabalho móvel) envia uma mensagem de pedido (junto com a informação requerida para processar) para seu coordenador de nó pré-determinado. As mensagens são controladas assincronamente, permitindo que a estação de trabalho móvel se desconecte da rede e execute algumas outras tarefas,

deixando o coordenador do nó para coordenar a execução das transações globais submetidas em seu lado. Além disso, como estas estações de trabalho móveis podem ter características de confiança totalmente diferentes, comparado-os aos computadores localizados nos vários locais remotos, o período de suas conexões pode ser intermitente e curto.

Por causa da natureza da computação móvel, a estratégia de MQF pode ser muito apropriada nesta implementação como:

(1) é simples para administrar a entrega e recuperação de mensagens;

(2) com tempos independentes nas estações de trabalho móveis, elas podem se desconectar da rede por um período ilimitado de tempo, enquanto as transações globais submetidas por estas estações de trabalho móveis estão sendo coordenadas e executadas pelo seu coordenador de nó respectivo;

(3) cada estação de trabalho tem habilidade para examinar o estado de suas transações globais a sua conveniência.

5.5.6.1 Gerenciamento de Filas de Mensagens

No MQF proposto, as mensagens podem ser classificadas em três tipos: (1) Mensagem de pedido; (2) Mensagem de reconhecimento; (3) Mensagem de informação.

Exemplos típicos de uma mensagem de pedido são:

Req_Reconnect (Pedido para Reconexão), ou *Req_Mws_Status* (Pedido Estado de Estação de trabalho Móvel).

Mensagens de reconhecimento são criadas com respeito a uma mensagem de pedido e são incluídas, por exemplo, *Ack_Reconnect_Mws* (Reconheça reconexão para estação de trabalho móvel).

Mensagens de informação incluem, entre outros, *Info_Msg_Queue* (estado de Fila de Mensagem) ou *Info_Mws_Status* (*Mws/Site* estado informação). Para cada uma das estações de trabalho móveis, existe uma fila de mensagem e uma fila de transação.

5.5.6.2 Gerenciamento de Estações de Trabalho Móvel

Para fazer uma conexão com o coordenador de nó, a estação de trabalho móvel executará o seguinte:

(4) verifique sua tabela de estado de mws para seu último estado de conexão onde o *status_table={node_id, mws_id, mws_timestamp, mws_status}*;

(5) envie um *Req_Connect* para o GTMC designado com o seguinte {*node_id*, *mws_id, mws_timestamp*, ação}, onde ação = {conecte};

(6) grave o pedido de conecte no arquivo de *log* da estação de trabalho móvel;

(7) atualize a tabela do estado de mws.

Após a recepção deste pedido, o GCM executará o seguinte:

- Guarde o pedido de conecte em seu arquivo de *log* e então reconheça a conexão respondendo *Ack_Connect*, para as estações de trabalho móveis. Também, confira e atualize a tabela de estado de estação de trabalho que é mantida em armazenamento.

- Faça uma varredura na fila de saída para qualquer saída importante e roteie esta informação para a estação de trabalho móvel, se necessário.

Para fazer uma desconexão de seu coordenandor de nó, a estação de trabalho móvel executará o seguinte:

- envie um pedido de *Req_Disconnect* para o GTMC designado com o seguinte {*node_id, mws_id, mws_timestamp*, ação}, onde ação = {desconecte};

- grave o pedido de registro desconecte no arquivo de *log* da estação de trabalho móvel.

Após a recepção deste pedido, o GCM executará o seguinte:

- grave o pedido de desconecte em seu arquivo de *log* e então reconheça a desconexão, respondendo *Ack_Disconnect* para a estação de trabalho móvel. Atualize sua tabela de estado de mws que é mantida em armazenamento então.

O fluxo procedural global no gerenciamento das estações de trabalho móveis é mostrado na Figura 5.7.

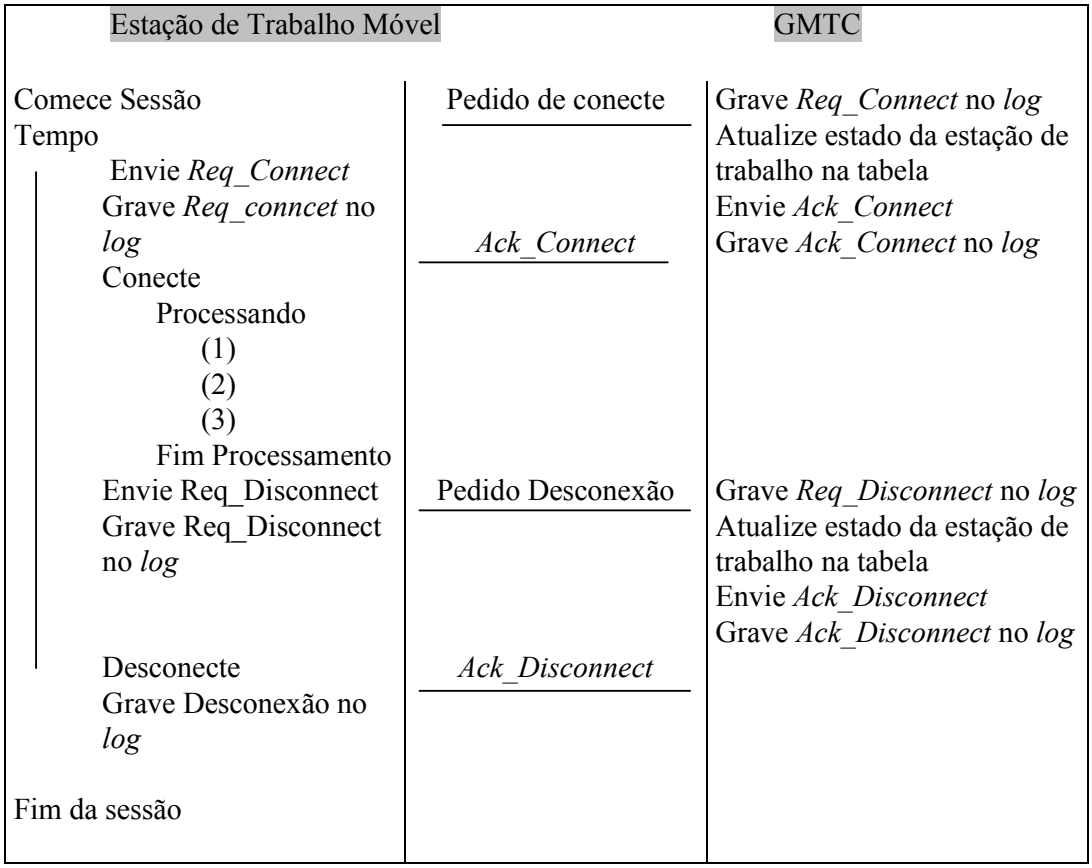

Figura 5.7 – Gerenciamento de Estações Moveis de Trabalho

Para gerenciar as transações submetidas por estações de trabalho móveis, é proposta uma transação global simples com mecanismo de fila. O princípio básico sob o mecanismo fazendo fila está o conceito de máquinas de estados finitas. Isso acontece porque pode ser definido um conjunto de possíveis estados e transições de um estado para outro durante o período de vida de uma transação global, isto é, do período de vida das ações ou sub-transações dentro das primitivas:

BEGIN_GLOBAL_TRANSACTION (Comece_transação_global) e
A*ND_GLOBAL_TRANSACTION* (finalize_transação_global)

Em conseqüência disso, a chave para implementar o modelo proposto é projetar uma fila fácil de fazer, que trace cada um destes estados. Existem cinco sub-filas de transação que são usadas para administrar transações/sub-transações globais submetidas ao local pela estação de trabalho móvel.

Fila de Entrada: esta fila contém todas as transações/sub-transações globais que chegaram primeiro ao coordenador do nó. Está monotomicamente em ordem crescente, baseada no tempo de chegada da transação.

Fila de Alocação: serão selecionadas as transações/sub-transações globais para execução baseado em *first-in first-out* (FIFO – primeiro-que-entra / primeiro-que-sai), ou uma prioridade baseada no algoritmo de escalonamento. Todos os bloqueios pedidos para transações/sub-transações globais serão adquiridos durante esta fase.

Fila Ativa: esta fila contém transações/sub-transações globais ativas atualmente.

Fila Suspende: esta fila contém todas as transações/sub-transações globais que completaram a primeira fase do protocolo *two phase committ* (consolidação em duas fases) (YEO & ZASLAVSKI, 1994).

Fila de Saída: esta fila contém todas as transações globais completadas.

Quando uma transação global está sendo submetida por uma estação de trabalho móvel, o GCM colocará a transação global na Fila de Entrada. Periodicamente, o GSS efetua uma varredura na Fila de Entrada e seleciona uma transação global para execução. Uma vez selecionada, a transação global será transferida para a Fila de Alocação onde todos os pedidos de bloqueio serão adquiridos por GCS. Então a transação global será transferida à Fila Ativa onde sub-transações globais são executadas em outros locais e serão despachadas pelo GCM. Uma vez que a transação global completou a primeira fase do protocolo *two phase committ*, é colocada, então, na Fila Suspenda e, para concluir o *two phase committ,* a transação global é colocada na Fila de Saída.

Uma vez que uma transação foi selecionada para execução pelo GSS, é necessário um controle de concorrência que é adquirido e gerenciado pelo GCS. Para assegurar a consistência dos objetos de banco de dados na execução entrelaçada de múltiplas transações concorrentes, um mecanismo de controle de concorrência é usado para isolar os efeitos de uma transação sobre as outras transações executadas simultaneamente. Segundo FERREIRA & FINGER (2000), tal protocolo é baseado na existência de tabelas de bloqueio, que faz com que haja um certo grau de replicação das informações já mantidas nos gerenciadores locais. Além disso, uma transação local só poderá emitir uma operação após obter o bloqueio de sua transação global imediatamente superior. A

serializabilidade é o critério de justeza usado habitualmente para designar o mecanismo de controle de concorrência.

De acordo com YEO & ZASLAVSKI (1994), um dos principais problemas encontrado para manter a serializabilidade das transações globais é como assegurar que a ordem de execução de transações globais é preservada pelo gerente de transação local (LTM). Alguns conflitos entre sub-transações globais e transações locais pode mudar a ordem de execução de transações globais. Um método de etiqueta pode ser usado para solucionar este problema. Todas as sub-transações globais são forçadas a obter primeiro uma etiqueta, assim, se houver conflitos adicionais causados entre eles, a ordem de suas execução é preservada .

5.5.7 Canguru ou KT

De acordo com DUNHAM et al. (1997), a relação entre movimento de uma unidade móvel entre células e a Transação Canguru é correspondente. É assumido que a transação tenha começado através da unidade móvel na célula 1 que é associada a Estação Base 1. Neste momento, o AAD nesta Estação Base criou uma Transação de Canguru nova e imediatamente criou uma Transação Filhote. Quando a unidade móvel se mudar para célula 2, um *handoff* é executado. Como parte deste *handoff*, a Transação Canguru é dividida em duas transações. A primeira parte desta transação são as sub-transações debaixo da Transação Filhote1, o resto (neste momento) fará parte da Transação Filhote2. Semelhantemente, quando a unidade móvel passar a célula 3 e célula 4, *handoffs* acontecem e novas Transações Filhotes são criadas com uma operação de divisão.

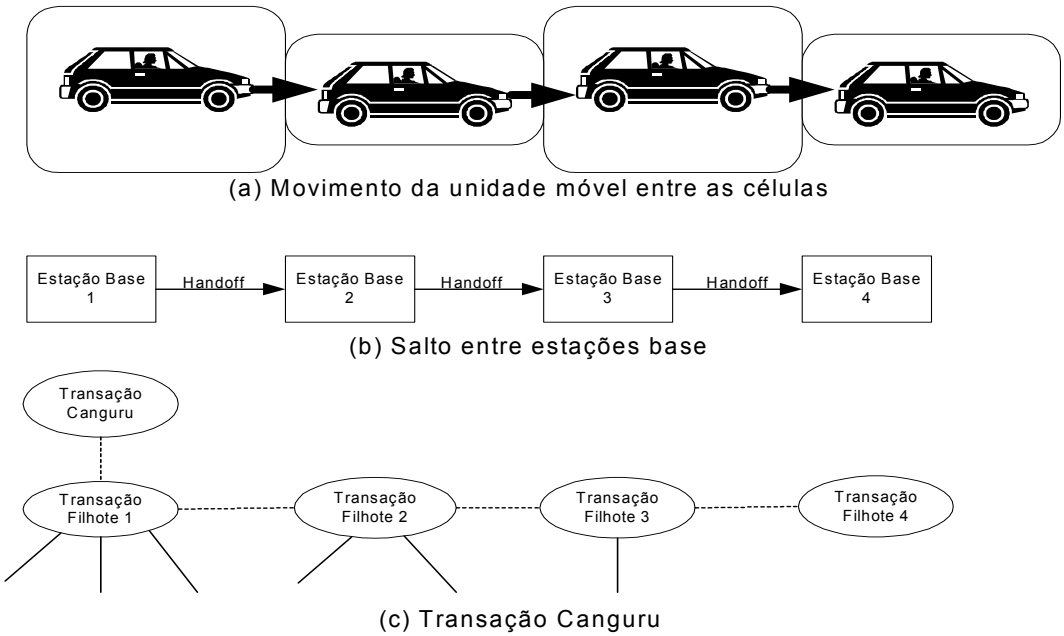

Figura 5.8 – Transação Canguru
Fonte: DUNHAM et al (1997)

Uma nova Transação Filhote só é criado quando acontecer um salto entre células. A mesma transação pedida em dois tempos diferentes pode ter estruturas diferentes. Por exemplo, uma transação com um curto tempo de vida ou com tipo de mobilidade lento ou uma transação com um longo período de vida e mobilidade rápida. Embora a estrutura de cada transação seja diferente em termos do número de Filhotes e o formato deste, o conjunto subjacente de transações globais e locais de cada é o mesmo. Assim, duas Transações de Canguru são Equivalentes se eles têm a mesma bolsa (DUNHAM et al 1997).

Conforme DUNHAM et al. (1997), as Transações Filhotes são criados por uma operação de divisão. Como parte do procedimento de *handoff*, uma operação dividida sempre é executada. Conforme a Figura 6.8(b), uma divisão é executada primeiro durante o salto da Estação Base 1 para a Estação Base 2. A Transação Canguru é então dividida em duas sub-transações: TF1 e TF2. É assumido que as sub-transações da transação móvel (isto é, as transações globais e locais) são executadas na seqüência. O usuário não pede a próxima sub-transação até que seja previamente completada. Assim, assume-se que as transações locais e globais debaixo de TF1 acontecem antes das que estão debaixo de TF2. Isto garante que TF1 preceda TF2 em ordem serializável.

Assumi-se que nenhuma sub-transação será criada debaixo de TF2 até que a TF1 que está sendo executada seja consolidada

Quando TF2 é criada, ha uma transação local ou global, provavelmente debaixo de TF1 que está sendo executado atualmente. Porém a TF2 deve ser criada quando o *handoff* acontecer. A situação apresentada na Figura 5.8 (c) mostra que TF4 foi criado, mas nenhuma sub-transação ainda existe.

Transações cangurus utilizam o mesmo mecanismo que as transações de multibanco de dados, no caso o bloqueio de cada sub-transação que está em execução.

5.6 Considerações

No decorrer deste capítulo pode-se observar os mecanismos de controle de concorrência utilizados pelos modelos de transações móveis, conforme está descrito na tabela 6. GRAY et al. (1996) descrevem que quando é utilizado pré-ordenação por marca de tempo, se o valor atual do objeto já tem uma marca de tempo maior que a marca de tempo da atualização, a atualização esta descartada. Isto pode gerar o problema de atualizações perdidas, requerendo assim, além da marca de tempo, um mecanismo a mais que garanta que as atualizações não serão descartadas.

Quanto a maioria dos modelos estudados, utilizam mecanismo de bloqueio. Porém, o mecanismo de bloqueio utilizado não é o modelo estrito. Observamos que os modelos utilizam o bloqueio para gravações. Assim que estas terminam o bloqueio é liberado. Na tabela 6, pode-se observar o resumo dos modelos propostos, tipo de transação e o mecanismo de controle de concorrência .

Tabela 6 - Mecanismos de controle de concorrência utilizados e tipos de transação.

Modelo	**Tipo de Transação**	**Mecanismo de Controle de Concorrência**
Pro-motion	Transação móvel	Tabela de bloqueio
Clustering	Transações *weak e strict*	Bloqueio em duas fases
Reporting	Sub-transações	Tabela de bloqueio
Two-Tier	Transações tentativas e básicas	Pré-ordenação

Modelo	**Tipo de Transação**	**Mecanismo de Controle de Concorrência**
Prewrite	Transações *pre-write* e *pre-read*, *read e write*	Bloqueio em duas fases
Canguru	Sub-transações	Tabela de bloqueio
MDSTPM	Transações Locais	Tabela de bloqueio
Semantics Based	Transação Móvel e *commit* local	Bloqueio em duas fases

No próximo capítulo, serão descritos cinco modos de controle de concorrência utilizados em transações de banco de dados móveis.

CAPÍTULO VI

MECANISMOS DE CONTROLE DE CONCORRÊNCIA PARA AMBIENTES MÓVEIS

Os mecanismos de controle de concorrência utilizados em bancos de dados móveis, se enquadram na classificação otimista, definida por ÖSZU & VALDURIEZ (2001). Estes mecanismos são utilizados porque usam a suposição que existirão poucos conflitos. Os algoritmos otimistas atrasam a fase de validação até um pouco antes da fase de gravação. Com isso, uma operação submetida a um escalonador otimista nunca é atrasada.

As operações de leitura, computação e gravação de cada transação são processadas livremente, sem atualizar o banco de dados real. Como passo inicial, as transações fazem atualizações sobre cópias locais dos ítens de dados. Na fase de validação, é verificado se essas atualizações mantêm a consistência do banco de dados. Em caso afirmativo, as mudanças serão efetuadas no banco de dados real. Caso contrário, a transação será abortada e seus resultados locais revertidos, e terá de ser reinicializada.

Nas próximas seções serão apresentados cinco algoritmos que prevêem o controle de concorrência sob um ambiente móvel. Estes algoritmos prevêem serializabilidade e utilizam a troca de mensagens para detectar conflitos. Um dos problemas na computação móvel é justamente o volume de troca de mensagens que os mecanismos de controle de concorrência podem gerar.

6.1 Controle de Concorrência para Sistema Móvel com Dados Difundidos - MEI-WAI et al. (1999)

Um dos métodos propostos para disseminação de dados é a difusão. Nos dados difundidos, as transações móveis não precisam informar ao servidor de difusão antes de acessar o item de dados. Estes dados podem ser adquiridos do “meio”, enquanto está ocorrendo a difusão. Porém, se as atualizações para o banco de dados terminarem simultaneamente, as transações móveis podem observar valores de dados inconsistentes.

O modelo utilizado é a difusão do servidor (Figura 6.1), onde existem vários clientes móveis e uma rede móvel do tipo sistema de rádio celular, que provê largura de banda limitada para difusão de dados. Os clientes móveis, se comunicam com o servidor difusor em canais de baixa largura de banda na rede sem fio.

O servidor de banco de dados mantém um banco de dados com informações úteis sobre o ambiente externo, como atualizações de ações, informações de tráfego e notícias. Estes valores podem ter alterações dinâmicas. Para manter a validade e "atualidade" dos ítens de dados, atualização das transações são geradas para *refresh* dos valores dos dados sempre que o estado dos objetos no ambiente externo mudam. Cada transação de atualização tem uma marca de tempo. A marca de tempo é utilizada para indicar em qual momento a atualização é gerada. No item de dados é registrado com o novo valor no seu número de versão.

Neste modelo, é assumido que as transações de atualizações são curtas, consistindo principalmente de uma para várias operações de *write.* Algumas transações de atualizações também podem conter operações de leitura. Além disso, um protocolo de concorrência, como o bloqueio em duas fases, é usado. Este mantém o controle de concorrência entre as transações de atualizações no servidor de banco de dados.

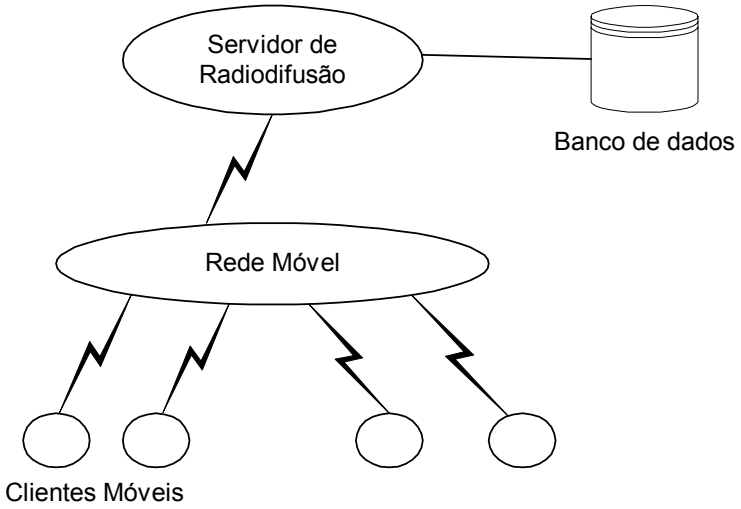

Figura 6.1 - Modelo do Sistema
Fonte: MEI-WAI et al, (1999)

O servidor de banco de dados difunde constantemente os ítens do banco de dados, um a um para os clientes móveis da rede até o fim do ciclo de difusão. O comprimento

de um ciclo de difusão pode ser fixo ou variável. Os ítens de dados podem requerer tempos diferentes por difusão devido aos seus tamanhos diferentes.

No modelo, o processo de difusão é modelado como uma *Broadcast Transaction* ((BT) transação de difusão). O conjunto de ítens de dados na transação de difusão consiste nos ítens de dados que são difundidos num ciclo. A transação de difusão e *Update Transaction* ((UT) transação de atualização), são executadas entrelaçadas. Com isso, é reduzido a demora dos bloqueios para a instalação de transações de atualizações.

Os clientes móveis emitem transações somente de leitura, chamadas de Transações Móveis (TM), para acessar os ítens de dados do servidor de banco de dados.

6.1.1 Primeira Atualização com Ordem (UFO)

O maior problema no projeto de protocolo de controle de concorrência para sistemas de computação móvel, usando difusão de dados, está no servidor de banco de dados que geralmente não percebe que ítens de dados estão sendo lidos pela transação móvel e em que momento. Isso faz com que dados conflitantes sejam difíceis de detectar. No ambiente móvel, a largura da banda dos canais móveis, especialmente para *up-link*[1], é muito limitada. Por isso, é importante que para a performance do sistema, se minimize o número de comunicações entre clientes móveis e o servidor de difusão, especialmente usando canais de *up-link*. O algoritmo UFO checa conflitos de dados entre transações de difusão e de atualizações, ao invés de detectar conflitos entre transações móveis e de atualização.

No UFO, o comprimento da transação de difusão é redefinido para o comprimento do conjunto de ítens de dados, que serão difundidos num período de tempo, tal que nenhuma transação móvel lê ítens de dados a mais do que uma transação de difusão. O princípio básico do algoritmo UFO é assegurar que se um conflito de dados ocorrer entre uma BT (*broadcast transaction*) e uma UT (*update transaction*), a serialização imponha a seguinte ordem: UT→ BT. BT→UT só será permitido se for impossível uma transação móvel, que lê dados de BT, ser diretamente dependente em UT. Considerando que transações móveis leram ítens de dados de uma transação de difusão, a ordem da

[1] Up-link – ligação do cliente móvel para o servidor.

serialização entre BT e MT sempre será BT→MT, se elas têm algum conflito. Assim, a ordem de serialização entre transações de atualização e transações móveis sempre será UT→MT, se existir um conflito. Então, o gráfico final de serialização pode ser assegurado que é serializável.

Basicamente o UFO consiste em duas partes:

- execução da transação de atualização; e
- resolução do conflito entre a transação de atualização e a transação de difusão.

A execução de uma transação de atualização é dividida em duas fases: a fase de execução e a fase de atualização. Durante a fase de execução, as operações de uma transação de atualização são executadas e qualquer dado que está em conflito com outras transações de atualizações é resolvido usando um protocolo de controle de concorrência convencional, como o bloqueio em duas fases. Os valores novos da operação de escrita são escritos em um *workspace* (espaço de trabalho) privado, ao invés de atualizar imediatamente o banco de dados. Quando todas as operações de atualização estiverem completas, entra a fase de atualização, na qual as atualizações se tornam permanentes no banco de dados. Os valores então são cópiados do *workspace* privado para o banco de dados. Os dados que estão em conflito com a transação de difusão, serão conferidos na fase de atualização, depois da conclusão da atualização. Verifica-se que as transações de atualização adotam bloqueio em duas fases para solucionar conflitos de dados com outras transações de atualizações e usa uma abordagem otimista para descobrir conflitos com transação de difusão.

Existem duas vantagens importantes em dividir a execução de uma transação de atualização em duas fases. Primeiro, isto reduz significativamente a probabilidade de bloqueio das transações de difusão. Se uma operação difundida quer ler um item de dado que está bloqueado por uma transação de atualização, a transação de difusão será bloqueada até que a transação de atualização seja consolidada, seguindo o princípio do bloqueio em duas fases. Ao mesmo tempo, a transação de atualização que está de posse do bloqueio, pode ser bloqueada devido a conflitos de dados com outras transações de atualizações. Devido ao bloqueio transitivo, o tempo de bloqueio de uma transação de difusão pode ser muito longo. Dividindo a execução da transação de atualização em duas fases, pode-se reduzir a probabilidade de bloqueio e o tempo de bloqueio da

transação de difusão. Desde que, os conflitos entre uma transação de atualização e uma transação de difusão só aconteçam quando a transação de atualização estiver na fase de atualização. Segundo, a detecção dos conflitos de dados entre uma transação de atualização e uma transação de difusão se tornará mais fácil. Na fase de atualização, o sistema sabe quais ítens de dados foram acessados pela transação de atualização, comparando o conjunto de escrita de uma transação de atualização com o conjunto de leitura de uma transação de difusão. O sistema pode determinar se há algum dado em conflito entre os conjuntos.

Quando o conflito entre uma transação de atualização e uma transação de difusão é detectado, ele é resolvido quando a transação de atualização entra na fase de atualização. Com o propósito de reverter a ordem da serialização de BT $\rightarrow$ UT ou na ordem desejável UT$\rightarrow$ BT. O seguinte algoritmo é definido no servidor de difusão. Este é executado quando uma transação entra na fase de atualização.

Se $O_{BT} \cap O_{UT} = \{\ \}$

Então BT e UT não tem dependência

Senão cada item de dado $d_i \in \{ O_{BT} \cap O_{UT} \}$

Re-difunda o item de dado d_i

Onde O_{BT} = conjunto de ítens de dados da transação de difusão, BT

O_{UT} = conjunto de ítens de dados da transação de atualização, UT.

Para re-difundir ítens de dados conflitantes, a ordem de serialização entre a transação de difusão e a transação de atualização é invertida. O conjunto de ítens de dados nas transações de difusão não são fixos desde que estes dados consistam na BT que sejam difundidos no período corrente. Depois da difusão dos ítens de dados, o último item de dados difundido será incluído na BT e o último item de dado na BT será removido.

6.1.2 Método de Checagem de Serialização (SCM)

O problema de performance encontrado no algoritmo UFO é a sensibilidade quanto a probabilidade de conflitos de dados entre transação de atualização e transação

de difusão. Se a probabilidade de conflito é alta e o tamanho dos ítens de dados são grandes, a re-difusão pode provocar uma troca de mensagens significativamente alta, degredando a performance no sistema global. No SCM, este problema é aliviado porque a consistência dos valores de dados observado para uma transação móvel é assegurado se for executada uma checagem no gráfico local de serialização, em vez de confiar somente nos dados difundidos.

O principal problema da inconsistência de dados no ambiente de difusão, é que ítens de dados capturados no "meio", podem ser utilizados depois por uma transação de atualização. Se a transação de atualização tiver adicionais e estes estiverem em conflito com a transação de difusão, pode causar escalonamento cíclico com BT. Outra razão para o escalonamento cíclico é devido a dependência transitiva sobre a transação de atualização. Estes dois problemas podem proporcionar para o cliente móvel, conflito de dados de informação das transações de atualizações, para a construção de seus gráficos locais de serialização. No caso de um escalonamento cíclico ser formado, o cliente móvel disporá de um item de dado de leitura antes da transação de atualização. Isto causa um escalonamento não serializável. Para resolver o primeiro caso, o SCM re-difunde as identidades dos ítens de dados que são atualizados por uma transação no servidor de banco de dados. Para controlar o caso das dependências transitivas, o SCM simplesmente difunde as transações de atualizações que foram difundidas no último período.

6.1.2.1 - Algoritmo para o Servidor

O SCM divide a execução das transações de atualizações em duas fases, similar ao algoritmo UFO. Quando houver um conflito de entre uma transação de atualização e uma transação de difusão, este será descoberto quando a transação de atualização entrar na fase de atualização. No caso de dados em conflito, a dependência entre a transação de atualização e a transação de difusão será dividida em vez de ser re-difundidos os ítens de dados em conflito.

Depois da conclusão da transação de atualização, o seguinte algoritmo é executado:

Se $D_B \cap D_U \neq \{\ \}$

Então difunda a mensagem: D_U está atualizado

Senão Se $D_B \cap D_U \neq \{\ \}$ onde $U_I \in T_B$

Então difunda a mensagem: D_U está atualizado e UT's ID

Onde D_B = conjunto de ítens de dados da transação de atualização, UT

T_B = conjunto de ítens de dados da transação de atualização no último período depois da queda

6.1.2.2 Algoritmo para Transação Móvel

Transações movéis executam transações de difusão para seus ítens de dados requeridos e também executam transações de atualização para construir gráficos de serialização local. Uma vez que o gráfico de serialização é atualizado, uma procura no gráfico será executada. Se uma escala cíclica se formar, a transação móvel resolve isto, dispondo dos ítens capturados antes da transação de atualização, o que causa um escalonamento não serializável. Os ítens de dados dispostos serão capturados do "meio" quando houver a próxima difusão. O processo continuará até que todos os dados exigidos sejam capturados ou o período de queda da transação for expirado .

6.2 *Bounded* Bloqueio para Controle de Concorrência Otimista – PHATAK & BADRINATH (1995)

Neste modelo foi considerado um banco de dados que consiste em ítens de dados. Estes ítens de dados poderiam ser tuplas ou páginas, que dependem da granularidade do controle de concorrência. Um estado do banco de dados é consistente se está conforme o conjunto de regras. Uma transação opera no estado do banco de dados em um estado consistente e dá origem a um novo estado consistente ao banco de dados na consolidação. Se um estado consistente não for realizável, devido aos efeitos de outras transações concorrentes, a transação deve abortar e desfazer qualquer atualização passageira executada.

A transação, antes de acessar um item de dados, deve adquirir um bloqueio. Além disso, uma transação não deve liberar um bloqueio adquirido até sua consolidação,

similar ao bloqueio em duas fases recuperável. Transações de longa duração têm um pedido ativo de bloqueio no item de dados.

Uma transação de escrita não vai para o banco de dados, os dados ficam em uma área privada, até sua consolidação. No ponto de consolidação, a transação é validada e tudo o que a transação escreveu é escrito no banco de dados, modificando o estado do banco de dados. A validade da escrita deve acontecer em sinal de operação atômica. No entanto, pode existir um encarecimento nessa técnica. Neste modelo assume-se que cada escrita é precedida por uma leitura destes ítens de dados. Assim, o conjunto de gravação de uma transação é um subconjunto de seu conjunto de leitura.

O gerente de bloqueio mantém um *buffer* finito de bloqueio. Cada *slot* (ou *frame*) no *buffer* de bloqueio segura bloqueios e pedidos de bloqueios pendentes para um único item de dados. Assim, o número de ítens de dados com pedido de bloqueios ativos não podem exceder ao número de *slots* (aberturas) no *buffer*. O número de *slots* no *buffer* seria uma ordem de magnitude menor que o número de ítens de dados no banco de dados, mas seria maior que o número esperado de conflitos de ítens de dados.

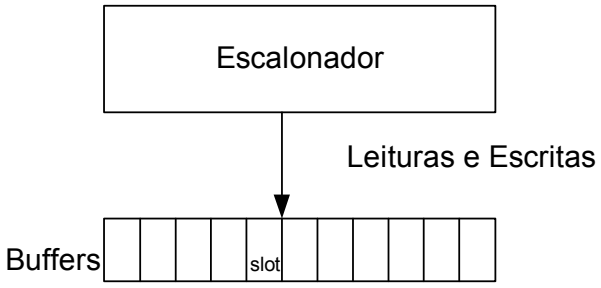

Figura 6.2 – O papel do Escalonador

Sempre que um bloqueio de um item de dados x é recebido pelo gerente de bloqueio, este primeiro tenta localizar x no *buffer* de bloqueio. Caso este item for achado, o gerente de bloqueio tenta postar o bloqueio no *slot* correspondente. O pedido de bloqueio pode ser concedido ou pode ser bloqueado, da mesma maneira pode ser colocado como bloqueio pendente, no estado de bloqueios existentes em x. Assim, para a leitura (compartilhda) seria concedido um bloqueio, se não houvesse nenhum bloqueio exclusivo existente em x. Um bloqueio de escrita exclusiva seria concedido, se não houvesse nenhum bloqueio existente em x. Se x não ficar situado no *buffer* de bloqueio, um *slot* deve ser locado para a postagem do pedido de bloqueio. Se existir um *slot* livre,

será usado, caso contrário um *slot* vítima deve ser selecionado. Qualquer *slot* sem bloqueio ou pedido de bloqueio é considerado livre. A marca de tempo de um *slot* é o último tempo em que o bloqueio pediu um *slot* para um item de dado. O item de dados y no *slot* vítima é então despejado. Todos os bloqueios em y são removidos e todas as transações bloqueadas são liberadas em y são desbloqueadas.

Se o tamanho do *buffer* de bloqueio é zero, então todas as requisições de bloqueio são rejeitadas e nunca há qualquer bloqueio ativo. Neste cenário, todas as transações ficam otimistas com respeito a todos os ítens de dados de seu conjunto de leitura e escrita e o sistema fica puramente otimista. Por outro lado, se o número de *slot* é menor ou igual ao número de ítens de dados do banco de dados, então cada pedido de bloqueio pode ser concedido sem despejar qualquer item de dado que exista bloqueio. Assim, o sistema é garantido seguir o bloqueio em duas fases, desde que cada transação consolidando possua bloqueio para todos os ítens de dados que acessa. Porém, o tamanho do *buffer* de bloqueio se for igual ao número de ítens de dados no banco de dados é o suficiente, mas não uma condição necessária para que esta técnica reduza ao bloqueio em duas fases (2PL). A única condição necessária é que ítens de dados nunca sejam despejados do *buffer* de bloqueio (um bloqueio nunca seja rejeitado). Porém, se isso ocorrer, o sistema poderá ficar muito pessimista, com *buffer* de bloqueio menor que o número de ítens de dados.

Na consolidação, a transação deve ser validada. Para uma transação ser validada, ela deve ser validada com respeito a todos os ítens de dados dos conjuntos de leitura e escrita. Um transação T é válida com respeito a um item de dados x, seguindo três condições :

- uma transação concorrente T′ de T, cuja execução sobrepõe T;
- ou T′ começou antes que T entrasse na fase de validade;
- ou T começou ante de T′ mas antes de T′ entrar na fase de validade.

Detalhando mais essas condoções, teremos:

1 – T segura um bloqueio em x (o bloqueio adquirido por T não foi despejado)

2 – x não está no conjunto de escrita de T, T não segura o bloqueio em x e ambos seguem as seguintes condições

(a) se outra transação T′ pedir um bloqueio em x, então não é um bloqueio exclusivo;

(b) x não foi escrito elaboradamente depois que foi lido por T (não está no conjunto de escrita de qualquer transação concorrente e consolidada).

3 – x está no conjunto de escrita de T, T não consegue um bloqueio em x e ambos seguem as seguintes condições ,

(a) nenhuma outra transação consegue um bloqueio em x (ou compartilhado ou exclusivo);

(b) x não foi escrito elaboradamente depois que foi lido por T.

Depois que a validação tiver sucesso, o conjunto de escrita de uma transação é escrita no banco de dados e todos os bloqueios segurados pela transação são liberados.

A exigência da condição 2 e 3, que T não consegue um bloqueio em x, é desnecessária, desde que a condição 1 impeça qualquer subcondição da condição 2 e condição 3. Se x não esta no conjunto de escrita de T então T′ consegue um bloqueio (compartilhado) em x. Isto implica que nenhuma outra transação poderá conseguir bloqueio exclusivo em x. Além disso, desde que T's bloqueiam x durante o tempo de vida de T, nenhuma outra transação poderá escrever em x. Assim, nem a condição 2a nem a 2b podem conseguir. Similar, se x está no conjunto de escrita de T e, T conseguir um bloqueio (exclusivo) em x, então nenhuma outra transação pode conseguir um bloqueio em x (desde que todos os bloqueios conflitem). Assim, nem a condição 3a nem a 3b podem conseguir.

Estas condições estão incorporadas no algoritmo de validação que segue.

Algoritmo de Validação

```
//Entradas: Transação T
// Loop principal:
    para todos x ∈ ao READSET (T) faça
        se x ∉ ao LOCKSET (T) então
            se x ∉ WRITESET (T) e ∃ T′: x ∈ WLOCKSET(T′) então
                aborte
                retorne
```

```
        fim se
        se x ∈ WRITESET (T) e ∃ T′: x ∈ WLOCKSET(T′) então
            aborte
            retorne
        fim se
        se ∃ consolidado T″ concorrente de T: x ∈ WRITESET (T″) então
            aborte
            retorne
        fim se
    fim se
fim para
Commit T
Fim
```

Onde, para uma transação T, READSET(T) é o conjunto de ítens de dados por T, WRITESET (T) é o conjunto de dados de escrita por T e *LOCKSET*(T) é o conjunto de todos os ítens de dados que T possui bloqueios e, *WLOCKSET*(T) $\subseteq$ *LOCKSET* é o conjunto de todos os ítens de dados que T possui bloqueios exclusivos no momento de validação. Uma transação T' concorrente de T, é qualquer transação que era concorrente ativo com T (sobreponha seus tempos de vida). Qualquer transação que possui bloqueios quando T é validado, é automaticamente concorrente de T (desde seja ativa para segurar bloqueios).

Estas condições de validação que gerenciam *read/write* e *write/write* estão em conflito entre várias transações em um modo complexo. Para qualquer item de dados x, se todas as transações seguram bloqueios em x ou são bloqueadas para espera de bloqueios em x, então puro 2PL previne conflitos pela condição 1. Por outro lado, se todas as transações estão puramente otimistas com respeito a x, então as condições 2b e 3b (que são idênticas) previnem conflitos, enquanto usando um modelo estritamente otimista. Através da condição 1 e condições 2a e 3a, uma transação que tem um bloqueio existente em x sempre é determinada a "prioridade" sobre uma transação que não faz. Assim, o algoritmo trabalha priorizando acessos pessimistas a dados sobre acessos otimistas a dados. A condição de validação previne qualquer transação que é

otimista com respeito a x de escrever para x, quando outra transação segurar um bloqueio em x (compartilhado ou exclusivo). Semelhantemente, se uma transação possui um bloqueio exclusivo (escreva) em x, então nenhuma outra transação que leu x pode consolidar até que este bloqueio seja libertado (nenhuma outra transação desbloqueada pode possuir uma bloqueio em x enquanto esta transação tem bloqueio exclusivo). Conflitos puramente otimistas são controlados usando condições 2b e 3b, que também acontecem por serem as únicas condições relevantes quando buffer de bloqueio é de tamanho zero e o sistema é puramente otimista. Um detalhe fundamental do algoritmo é que, desde que bloqueio e bloqueios pedidos podem ser despejados, a transação pode somente requerer um bloqueio uma vez no item de dados. Caso contrário, o cenário seguinte pode acontecer: transação T_1 bloqueia x e y, escreve y. x e y são então despejados do *buffer* de bloqueio. Outra transação T_2 bloqueia x e y, lê y e consolida. T_1 novamente bloqueia x e y, escreve x e y e consolida. T_1 poderá consolidar desde que tenha segurado o bloqueio em x e y. Porém, o escalonamento não é serializável. O pedido de atualização é tratado parecido com um pedido de bloqueio, espera-se que no caso do item de dados não estar no *buffer* de bloqueio, o pedido é "rejeitado" sem selecionar um *slot* vítima, e a transação tem que continuar seja otimista com respeito a este item de dados (se a atualização falhasse, então o seria bloqueio previamente rejeitado e a transação se tornaria otimista com respeito a este item de dados).

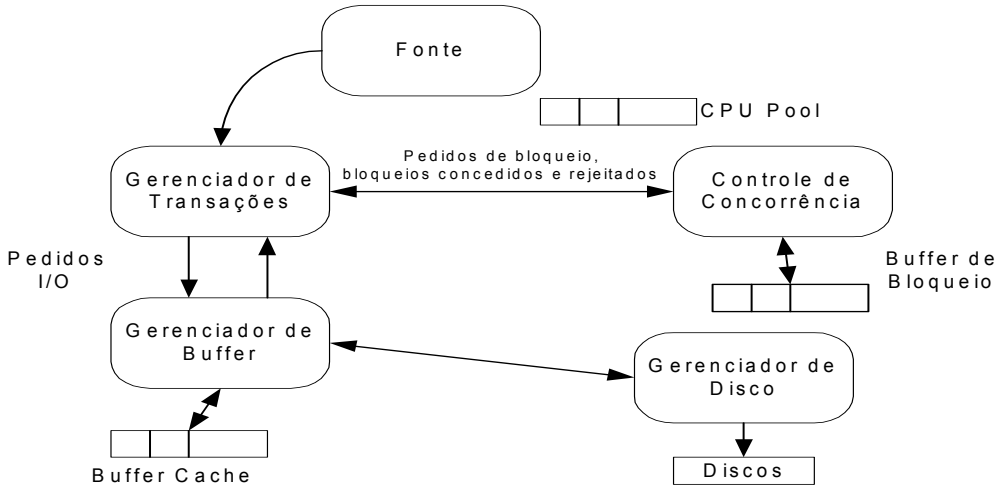

Figura 6.3 – Simulação do diagrama de Bloqueio

6.3 Gerenciamento Distribuído de Bloqueio para Transações Móveis – JIN JING et al, 1994)

Considere o impacto da mobilidade de unidades (*hosts*) de transação no gerenciamento com o protocolo de controle de concorrência de bloqueio/desbloqueio de leitura, em um ler-um escrever-todos. O presente esquema de bloqueio permite que uma leitura desbloqueie, para um item ser executado em qualquer cópia local daquele item. O desbloqueio pode ser feito em local diferente do local da cópia no qual o bloqueio de leitura foi fixado. Neste esquema proposto, são utilizadas as cópias reproduzidas de ítens de dados para reduzir os custos de mensagens incursas pela mobilidade das unidades de transação. Este algoritmo de bloqueio otimista é chamado de O2PL-MT (*Optomistic Two Phase Locking – Mobile Transaction*) .

Partindo do algoritmo O2PL (otimista 2PL), que foi apresentado dentro para um sistema de banco de dados distribuído convencional, o O2PL-MT concede bloqueio de leitura imediatamente em demanda e adia bloqueios de escrita até tempo de consolidação. Porém, em um ambiente móvel onde são reproduzidos ítens de dados, O2PL-MT requer menos mensagens que O2PL.

O método também pode ser usado para melhorar a eficiência de outro protocolo de bloqueio distribuído (por exemplo bloqueio pessimista), em um ambiente móvel onde operações de leitura excedem em número as operações de escrita. Em um sistema de computação móvel, o fator de mobilidade é de importância extrema no projeto de um algoritmo distribuído.

O método de controle de concorrência otimista é geralmente satisfatório para aplicações de baixa contenção de dados. O O2PL-MT assume que bloqueios de leitura são executados imediatamente, quando são executadas operações em servidores de dados. As transações têm, assim, garantidas, a busca de cópias consistentes de ítens de dados durante a sua execução. Esta vantagem é diluída um pouco pelo fato de que ítens de dados podem estar sujeitos a bloqueios prolongados durante os períodos de desconexão das unidades móveis. Porém, este problema pode ser focalizado por um mecanismo de intervalo que permite servidores de dados abortarem transações unilateralmente, cujas unidades estiveram desconectados por um determinado período de tempo, porque a distância física entre dois pontos necessariamente reflete a distância

de rede. Nesse caso, o caminho de comunicação pode cultivar desproporcionalidade ao movimento atual.

O mecanismo pode aceitar parâmetros de intervalo de um usuário-especificado, de forma que as unidades móveis possam apoiar as aplicações dos usuários efetivamente durante a desconexão.

Neste modelo, são considerados um servidor de banco de dados e um banco de dados para cada unidade fixo. Um servidor de banco de dados suporta operações de transação básicas como **leitura, escrita, prepare, conslide e aborte**. Cada MSS tem um coordenador, que recebe operações de transação das unidades móveis e seus monitores executam em servidores de banco de dados dentro das redes fixas. As operações de transações são submetidas por uma unidade móvel ao coordenador em sua MSS, que em troca as envia ao banco de dados distribuído dentro das redes fixas para execução. Por exemplo, o coordenador enviará uma operação a um servidor local, se a cópia que será lida estiver no neste local. Semelhantemente, para uma operação de consolidação, o coordenador monitora a execução de protocolo 2PC sobre todos os servidores envolvidos na execução da transação.

Algumas unidades móveis podem submeter transações em um destes dois modos:

1 - uma transação inteira pode ser submetida em uma única mensagem de pedido; a transação inteira se torna uma unidade de submissão. A unidade móvel também entrega o controle de execução a seu coordenador e espera o retorno dos resultados da execução da transação.

2 - por outro lado, as operações de uma transação podem ser submetidas a múltiplas mensagens de pedido. Uma unidade de submissão consiste em uma operação (por exemplo leia) ou grupo de operações; a unidade móvel interativamente submete as operações de uma transação a seu coordenador. Uma operação subseqüente pode ser submetida somente depois que tenha sido previamente executada e os resultados retornarem do coordenador.

Enquanto a primeira abordagem envolve um único coordenador para todas as operações de uma transação, a segunda abordagem pode envolver múltiplos coordenadores, por causa da mobilidade das unidades. Por exemplo, uma unidade móvel pode passar a uma célula nova, depois disto obter os resultados de operações previamente submetidas. Na célula nova, submeterá o resto das operações de transação

ao coordenador no MSS novo. O modelo proposto emprega a segunda abordagem para a submissão de transação. Esta abordagem suporta a execução interativa de transações o que oferece um aumento de flexibilidade em computações de transação.

É assumido que uma unidade móvel só pode mover-se de sua célula atual depois que receber resultados para todas as operações submetidas naquela célula. Porém, uma unidade móvel pode mover-se a qualquer hora. Pode mover-se para longe de sua célula atual depois que submeteu uma operação e antes de receber uma resposta do coordenador. Neste caso, são precisos procedimentos adicionais para localizar a unidade móvel e carregar os resultados de operações submetidas para ela. Pela simplicidade, este modelo ignora este caso. Porém, acredita-se que o método discutido poderá ser aplicável com a incorporação destes procedimentos.

É assumido que só uma transação pode ser inicializada por uma unidade móvel a qualquer hora. Ou seja, uma unidade móvel só pode inicializar uma transação depois que a transação prévia terminar. A transação submetida pela unidade móvel é terminada numa unidade de transação móvel. Uma transação móvel consiste em um conjunto de operações de leitura e escrita que são encapsuladas por uma declaração de COMEÇAR_Transação e uma declaração de FINALIZAR_Transação.

6.3.1 Algoritmo O2PL-MT para transações móveis.

O algoritmo O2PL usa uma abordagem de controle de concorrência "otimista," ler-um e escrever todos. Um bloqueio deve ser obtido imediatamente do local ou o mais próximo ao local da cópia para cada operação de leitura: bloqueio de escrita para cópias replicadas são adiadas até o começo da fase do *commit* ser alcançado. A idéia básica sobre O2PL é o conjunto de bloqueios imediatamente dentro do local (isto é possível se os ítens de dados são replicados). Com isso fica mais barato se for tratado com uma a abordagem mais otimista. Portanto, se usa menos mensagem intensiva por limite de local.

No ambiente de computação móvel, a mobilidade de unidades com transações também pode aumentar o número de mensagens e o custo no gerenciamento de bloqueios, que é a abordagem usada no O2PL.

Este algoritmo emprega um método simples para restringir o número de mensagens extras de desbloqueio de leitura. No lugar de enviar uma mensagem de desbloqueio de leitura ao servidor remoto da cópia, onde o bloqueio de leitura é feito, o O2PL-MT simplesmente permite a execução da operação de desbloqueio no local ou mais próximo ao servidor da cópia do item. Um exemplo: em vez de pedir o desbloqueio de (Xa) e desbloqueio de (Yb) nos servidores A e B, o O2PL-MT executará desbloqueio (Xc) e o desbloqueio (Yc) no servidor local C, a tempo consolidar, não necessitando transmitir mensagens.

Este método não é suficiente para assegurar a justeza de uma abordagem de um ler-um e escrever-todos. Antes de ser aplicada à redução de mensagens extras de desbloqueio, deve ser considerado o seguinte:

- Bloqueios de leitura devem ser garantidos para permanecer os efeitos em locais remotos quando o coordenador decide liberar os bloqueios e consolidar as transações localmente;
- Uma transação de atualização deve ser capaz de determinar o item para ser atualizado, que não tenha sido bloqueado por outra transação para leitura se o bloqueio de leitura e o desbloqueio são executados em diferentes locais. Tal checagem deve ser de custo baixo em termos de mensagem;
- Um mecanismo deve ser provido para remover os bloqueios de leituras pendentes em locais remotos no próprio momento, pois se tais bloqueios continuarem afetarão a execução de outras transações.

Focalizando a primeira consideração, começa com a observação que, em 2PL, um bloqueio deveria ser segurado até que nenhum novo pedido de bloqueio fosse preciso. Sem esta estipulação, a serializabilidade na execução da transação não poderia ser assegurada. Enquanto isto pode ser obrigado, os bloqueios são mantidos até o momento da consolidação de uma transação, esta condição não pode ser confirmada para bloqueios de leitura fixadas em locais remotos, sem o uso de transmissões de mensagem extras. Porém, ainda pode se assegurar a semântica correta dos bloqueios de leitura, se a versão da cópia do item a ser desbloqueado estiver inalterada de seu estado no momento do bloqueio. Um número de versão de cópia inalterado insinua que o item não foi atualizado através de outras transações desde que foi bloqueado. No momento da consolidação, pode ser comparado o número da versão da cópia de um item no seu

local. Isso pode ser obtido de outro local de cópia, no momento de bloqueio. Se os dois números combinam, pode-se concluir que a ordem de serialização de transação é igual a um esquema de bloqueio/desbloqueio convencional, no qual o desbloqueio da leitura sempre é executado no mesmo local da cópia onde o bloqueio é fixo.

Um algoritmo tradicional lê-um e escreve-todos como o O2PL, pede um bloqueio de escrita (e um novo valor é atualizado) é enviado para todas as cópias locais. Cada cópia local então determina se o bloqueio de escrita pode ser garantido. Se nenhum bloqueio de leitura está pendente, este fixa na cópia, a cópia local concede o pedido imediatamente e envia uma mensagem de resposta ao coordenador de transação (ou unidade). Caso contrário, o pedido é bloqueado na cópia local. O coordenador de transação coleciona mensagens de resposta destes locais. Uma vez que todos os locais respondem ao pedido, o coordenador de transação conclui que não há nenhum bloqueio de leitura pendente, em qualquer local da cópia e concede bloqueio de escrita. Assim, só uma troca de mensagem é preciso, entre o coordenador e qualquer local de cópia que pediu um bloqueio de escrita.

Porém, se o bloqueio e o desbloqueio de uma operação de leitura são executados em locais diferentes (ou servidores), uma única troca de mensagem entre o coordenador e um local de cópia é insuficiente. Na realidade, se um bloqueio de leitura foi fixo na cópia de um local, não pode ser concedido o pedido de bloqueio imediatamente nem pode ser bloqueado neste local. Esta complicação surge porque o local de cópia não pode saber se o bloqueio de leitura foi libertada e o desbloqueio da leitura foi executado em um local diferente. O local de cópia deve primeiro obter informação sobre o estado da cópia antes que possa libertar o bloqueio de leitura e pode conceder o pedido de escrita. Para colher esta informação, o coordenador coleciona informações de desbloqueio de todos os locais de cópia envolvidos na primeira-rodada de troca de mensagem. Se qualquer local indica que o desbloqueio foi executado, o coordenador pode enviar uma mensagem então ao primeiro local de cópia, enquanto permitindo que um pedido de bloqueio de escrita seja concedido.

Assim, para permitir a distribuição de operações de bloqueio/desbloqueio de leituras em locais diferentes, são exigidas duas rodadas de trocas de mensagem. Estas mensagens trocam permissões, o coordenador de transação decide se um pedido de bloqueio de escrita pode ser concedido pelos locais de cópia. Porque O2PL permite que

todos os bloqueios de escrita possam ser adiados até o tempo de *commit*, estas trocas de mensagens podem ser fundidas de fato no protocolo 2PC regular. Quer dizer, na primeira fase de 2PC, o coordenador coleciona a informação de bloqueio/desbloqueio de leitura de todos os locais de cópia. O coordenador não pode entrar na segunda fase de 2PC até que todos os locais votam "sim", e cada bloqueio de leitura seja igualado por um desbloqueio de leitura no item a ser atualizado. Na segunda fase, o bloqueio de escrita é fixo e as atualizações são obrigadas.

No método anterior, embora bloqueios de escrita não serem concedidos durante a primeira fase, todos os locais de cópia ainda têm que executar todas as outras funções requeridas pelo padrão 2PC (por exemplo a verificação de restrições de integridade para cada item de dados de atualização). Como no padrão 2PC, estas funções deveriam ser executadas no *workspace* temporário de cada local de cópia. Estas ações preparam servidores para obrigar operações de escrita (inclusive o conceder de bloqueios de escrita) em todos os locais de cópia durante a segunda fase do 2PC.

Para garantir e obrigar operações de escrita, um novo modo de bloqueio, chamado "*write intend*" (pretende escrever) ou *W_INTEND*, deve ser usado para bloquear a cópia em todos os locais antes destes locais responderem a um pedido de escrita na primeiro-rodada de troca de mensagem. Um *W_INTEND* pedido é compatível com um *R_LOCK* concedido (bloqueio de leitura) mas não com um *W_INTEND* concedido ou *W_LOCK* (bloqueio de escrita). Porém, um *W_INTEND* concedido não é compatível com um pedido *R_LOCK* ou um *W_INTEND* pedido ou *W_LOCK*. Quando um pedido de leitura é concedido em um local de cópia, *R_LOCK* é fixo na cópia. Quando a primeira fase de um pedido de escrita chega a um local de cópia, um *W_INTEND* é fixo na cópia se nenhum outro *W_INTEND* ou *W_LOCK* esta aplicado àquela cópia. Caso contrário, a primeira fase do pedido de escrita é bloqueado no local. Este cenário insinua que duas escritas diferentes estão pedindo o bloqueio. Se há um *R_LOCK* na cópia, a primeira fase do pedido de escrita fixará o *W_INTEND* imediatamente na cópia. Depois que o coordenador decidir consolidar a transação na segunda fase, o *W_INTEND* é atualizado a *W_LOCK* e a atualização é obrigada à cópia. Assim, o W_INTEND concedido prevenirá qualquer *R_LOCK* pedido novo ou *W_LOCK* de ser concedido na cópia. Caso contrário, a decisão do coordenador para conceder o bloqueio de escrita não será válido. A matriz de compatibilidade de bloqueios aparece na Tabela 7.

Tabela 7 – Matriz de Compatibilidade de Bloqueio

Tj (lock)	*W_INTEND*	*R_LOCK*	*W_LOCK*
W_INTEND	Não	Sim	não
R_LOCK	Não	Sim	não
W_LOCK	Não	Não	não

Fonte: JIN JING et al, 1994

Observando o terceiro ponto das considerações. Depois que os desbloqueios de leitura são executados no local de cópia, outros pedidos de bloqueios podem estar pendentes nestes locais. Os bloqueios de leituras pendentes podem ser removidos quando outra transação se preparar para atualizar a cópia. Como *two phase commit*/escreva o protocolo de bloqueio descrito sobre uma transação pode obter um bloqueio de escrita e só atualizar um item se achar que cada bloqueio de leitura em um local de cópia é igualado por um desbloqueio de leitura em algum local. Então, é seguro remover o bloqueio de leitura pendente antes de uma transação de atualização obter o bloqueio de escrita naquele artigo. O período pendente começa na consolidação das transação de leitura e termina com o primeiro bloqueio de escrita por outra transação. O bloqueio de leitura pendente não produz este efeito sobre qualquer outro pedido de leitura ou escrita. Assim, não há nenhuma desvantagem em se ter um bloqueio de leitura pendente na cópia de um item até que a cópia é atualizada.

6.4 Marcas de Tempo para Detectar Conflitos R-W em Computação Móvel – MADRIA S K (1997).

A abordagem de marca de tempo usada neste modelo permite que operações de leitura e escrita possam ser executadas independentemente na unidade móvel e no MSS. Assim, permite que aconteçam possíveis inconsistências às custas de mais disponibilidade. É assumido que atualizações que acontecem são escritas cegas. Em computação móvel, arquivos que mantêm dados como índices acionários, taxas da moeda corrente etc., podem ser atualizados sem necessariamente ler anteriormente os valores. Isto é considerado escrita cega. É permitido escrita cega e leitura em aplicações

de computação móveis para melhorar a disponibilidade. Porém, são validadas leituras que acontecem em ambos os lugares, enquanto as unidades estão desconectadas e na reconexão.

Observa-se que a oportunidade de operações de leitura na computação de banco de dados móvel é mais importante que o tempo de resposta (consolidação final). Ou seja, o *commit* final das transações de leituras pode ser demorado até que uma conexão é estabelecida entre o MH e MSSs, mas as operações devem ser permitidas acontecer em ambos, na unidade móvel e no MSS. As operações de leitura e escrita conflitam quando uma conexão é estabelecida entre MH e MSS. Se só for considerado escrita cega, conflitos de escrita-escrita não surgem. A abordagem deste modelo usa marcas de tempo para leitura e escrita junto com alguma informação adicional para detectar e reconciliar conflitos de leitura-escrita para um único arquivo. A informação adicional está na forma de um *row-vector* (vetor-fila) e cada marca de tempo de leitura e escrita é fixo com este fila-vetor. Estes *row-vector* também são comparados para descobrir conflitos, enquanto comparando marcas de tempo de leitura e escrita. A técnica para solucionar conflitos de leitura-escrita não leva em conta a semântica das operações que manipularam o arquivo e a semântica dos dados que são armazenados.

Considerações:

- transações não lêem o item de dados antes da escrita; considera-se só escritas cegas. Em ambiente de computação móvel, escrita cega ocorre com muita freqüência;
- que transações somente de leitura só consolidem depois que uma conexão seja estabelecida entre o MH e MSS;
- que nenhuma partição de rede ocorra, pois pode isolar estes MSSs fisicamente conectado. Assim, a qualquer hora, não há nenhum conflito entre cópias do arquivo da MSSs;
- os relógios físicos são sincronizados em cada MH e MSS;
- assumi-se a abordagem de “ler-um e escrever-todos” entre MSSs e, assim, não é permitido conflito de qualquer tipo entre MSSs;
- essa técnica de detecção de conflito trabalha somente para um único arquivo.

6.4.1 Estrutura dos Dados do Modelo

Definição 1: Duas operações em duas diferentes transações são ditas dentro de um conflito de leitura-escrita, se ambas as operações acessarem o mesmo arquivo de dados concorrentemente e, um das duas operações é uma operação escrita e a outro é uma leitura.

Definição 2: O primeiro tempo de atualização é o tempo quando a primeira atualização acontece na unidade móvel(MSS), depois que ela esteja desconectada do MSS (MH). Quando a unidade móvel(MSS) está desconectada, o tempo da primeira atualização na unidade móvel (MSS) é armazenado. Isso define o primeiro tempo de atualização na unidade móvel (MSS). Para próxima atualização, é armazenado seu tempo, mas é atualizado para atualizações subseqüentes. O segundo tempo ou atualizações subseqüentes será a último tempo de atualização na unidade móvel(MSS), antes de mover-se à célula nova e começar a receber as atualizações consistente com o MSS.

Definição 3: Um vetor de W-marca-de-tempo para um arquivo "f" está definido como uma sucessão de n + 1 de elementos de marca de tempo para a escrita, onde "n" é o número de MSSs no sistema. Assumi-se que há uma unidade. Cada elemento de marca de tempo pode ter no máximo duas tuplas, onde o primeiro valor é o primeiro tempo de atualização e o segundo valor é a última vez de atualização na unidade móvel ou MSS. Depois de uma desconexão, um vetor de W-marca-de-tempo novo será formado em ambos, na unidade móvel e na MSSs, provendo algumas atualizações na cópia do arquivo na unidade móvel e na MSSs. Quando uma atualização acontece na MSS (UM), só os elementos de marcas de tempo que correspondem a MSSs (MH) presente são atualizados e o outro correspondendo a unidade móvel (MSSs) permanece o mesmo. O vetor de W-marca-de-tempo é armazenado na forma do modelo seguinte: <{MSS1}, {MSS2},…. {MSSn},{UM}>.

Definição 4: Um *row-vector* de desconexão para uma cópia do arquivo "f" é uma tupla ordenada de valores de bandeira "0" ou "1". Inicialmente, o valor da bandeira é fixado a "1" para toda MSS e UM. Assumi-se que toda MSS está conectada o tempo todo. Então, as entradas no *row-vector* que correspondem a MSS sempre será "1". Sempre que uma atualização acontece na UM durante uma desconexão, o valor da

bandeira que corresponde a UM é mudado para “1” e para todos os outros é fixado “0”. Semelhantemente, se uma atualização acontecer no MSSs, o fila-vetor tem 0 entrada para a UM e “1” para os outros. Estes vetores movem-se com a unidade móvel de uma célula para outra e são fixos com cada vetores de W-marca-de-tempo e marca-de-tempo de leitura. O *row-vector* mantém a informação na forma do modelo seguinte: <MSS1, MSS2,…. MSSn, UM>.

Definição 5: Um gráfico de desconexão móvel PG(f) para qualquer arquivo “f” é um gráfico dirigido onde o nó de fonte é rotulado com os nomes de todas as MSSs e a UM no sistema tendo uma cópia do arquivo “f” e todos os outros nós são rotulados com um subconjunto deste conjunto de nomes. Cada nó (diferente da fonte) só pode ser rotulado com um dos nomes da MSS e a UM que se aparecem nos nós antecessores no gráfico; reciprocamente toda MSS ou UM têm que aparecer em exatamente um nó de seus descendentes.

Seguem algumas propriedades do Modelo:

- para a detecção de conflitos de leitura-escrita, é preciso somente a primeira e a última marca de tempo da escrita em cada vetor de W-marca-de-tempo;

- é preciso de todas as marcas de tempo que correspondem às operações de leituras executadas na unidade móvel e na MSS. Então, é armazenada uma marca de tempo de leitura por operação de leitura tanto na unidade móvel quanto na MSSs;

- é preciso armazenar no máximo dois vetores de W-marca-de-tempo: o da unidade móvel e o da MSSs; inicial e um pode ser depois da desconexão;

- inicialmente, o vetor de W-marca-de-tempo consiste na primeira e na última marca-de-tempo da escrita que corresponde a MSSs e à unidade móvel enquanto conectado. Depois, a unidade móvel pode ter um novo vetor de W-marca-de-tempo e toda a MSSs coletivamente pode ter outro vetor novo de W-marca-de-tempo;

- se uma operação de escrita acontece depois de uma desconexão da unidade móvel(MSSs), então as entradas de marca-de-tempo da escrita na unidade móvel (toda MSSs) é atualizada. Isso dá um novo vetor de W-marca-de-tempo a unidade móvel e outro a MSSs;

- marcas-de-tempo de leitura e escrita são mantidas na unidade móvel e na MSSs até que a unidade móvel estabeleça a conexão com a MSS.

Observações:

- Unidade móvel e cada MSS terão um novo vetor de W-marca-de-tempo que corresponde a cada desconexão, contanto que o valor do arquivo seja atualizado durante aquela desconexão. Quer dizer, cada desconexão corresponde a um vetor de W-marca-de-tempo novo no caso de existir uma atualização.

- Se não houver nenhuma atualização durante uma desconexão, não haverá nenhum vetor novo de W-marca-de-tempo. Assim, o valor retornado pela operação de leitura será o último valor atualizado quando a unidade móvel e a MSS estavam conectados.

6.4.2 Processo de *Row-Vector*

Associa-se um *row-vector* na desconexão com cada vetor de W-marca-de-tempo e com cada marca-de-tempo de leitura na unidade móvel e MSSs. O *row-vector* dá alguma informação adicional e é associado com a marca-de-tempo de leitura e escrita, tanto na MH quanto na MSS. A primeira atualização ao ser desconectado de uma célula velha, a célula nova atualizará a entrada no *row-vector* que corresponde a unidade móvel para 1 e para todas as MSSs para 0. As atualizações subseqüentes na unidade móvel enquanto desconectado não mudará as entradas de *row-vector*. Em uma célula nova, se não houver nenhuma atualização nova, uma operação de leitura devolverá o último valor atualizado na unidade móvel na célula anterior. Então, o *row-vector* fixo com a marca-de-tempo de leitura será o *row-vector* fixo ao vetor de W-marca-de-tempo correspondente a célula velha. Por outro lado, se houver uma atualização na célula nova, uma operação de leitura devolverá o valor novo e o *row-vector* associado com a marca-de-tempo da leitura será o *row-vector* fixo ao vetor de W-marca-de-tempo da nova célula.

6.4.3 Primeira e Última Marca-de-tempo de Escrita

Quando uma operação executa a primeira atualização no arquivo (o arquivo pode ter algum valor inicial antes) na MSSs e na unidade, o tempo desta operação de escrita é armazenado. Deste ponto para frente, a marca-de-tempo de escrita da próximo escrita

será armazenada, mas será atualizada para subseqüente escrita. Geralmente, quando uma desconexão é detectada e uma operação de escreva é executado independentemente na unidade móvel ou na MSSs, será a primeira operação que atualizará o arquivo na unidade móvel ou na MSSs. Então, seu tempo será armazenado como o primeiro tempo de atualização. Para a próxima escrita ou na MSSs ou na unidade móvel, seu tempo de escrita será armazenado, além de ser atualizado para a escrita subseqüente. Isto determinará a primeira e a última vez de atualização na unidade móvel ou na MSSs.

Quanto à descoberta de conflitos de Leitura e Escrita:

Suponha que o primeiro R-W está em conflito entre as marcas-de-tempo de leitura armazenado na unidade móvel e os elementos de marca-de-tempo de escrita dos vetores de W-marca-de-tempo armazenados na MSS. Podendo haver dois vetores de W-marca-de-tempo no máximo na unidade móvel ou na MSS, um é inicial quando todas as MSSs e a unidade móvel estavam conectadas e o outro pode ser criado na desconexão. Primeiro, compara-se a mais recente marca-de-tempo de leitura disponível na unidade móvel com a marca-de-tempo da escrita do arquivo do último vetor de W-marca-de-tempo na MSS. Suponha que a mais recente marca-de-tempo de leitura do arquivo na unidade móvel é menos que os correspondentes elementos da marca-de-tempo da escrita do arquivo na MSS. Neste caso, compara-se a marca-de-tempo de leitura com os elementos da marca-de-tempo escrita dos vetores de W-marca-de-tempo prévios na MSS.

Condições para detectar conflitos de leitura-escrita

Condição 1:

Se $\text{Min}\{wM_m, wM_n\}_{mss} < [rM_k]_{um} < \text{Max}\{wM_m, wM_n\}_{mss}$

e o *row-vector* fixo com as marcas-de-tempo de leitura e escrita forem ≠

então L-E conflitam

senão L-E não conflitam

Condição 2:

Se $[rM_k]_{um} = \text{Max}\{wM_m, wM_n\}_{mss}$ ou $[rM_k]_{um} = \text{Min}\{wM_m, wM_n\}_{mss}$

então L-E conflitam

Condição 3:

Se $[rM_k]_{um} > Max \{wM_m, wM_n\}_{mss}$ e o *row-vector* fixo com as marcas de leitura e escrita diferem e o *row-vector* fixo com a marca-de-tempo não é o *row-vector* inical

então L-E conflitam

senão L-E não conflitam

Condição 4:

Se $[rM_k]_{um} > Max \{wM_m, wM_n\}_{mss}$ e o *row-vector* fixo com as marcas-de-tempo de escrita e leitura são o *row-vector* inical

então E-L não conflitam.

A condição 4 é diferente da 3, desde que leve em consideração o caso de quando houver uma atualização na MSS depois que uma leitura tenha ocorrido na unidade móvel. Neste caso, os *row-vectors* são diferentes, mas não ocorre conflito e as leituras retornam valores antes que a primeira atualização na MSS, depois da unidade móvel, ter desconectado.

6.5 Considerações

Como visto no decorrer deste capítulo, o algoritmo que utiliza marcas de tempo para manter serializabilidade sem perder atualizações, utiliza, além das marcas de tempo, um vetor de fila (MADRIA, 1997). Porém, nesse modelo o autor considera que aconteçam escritas cegas e que transações de leitura só consolidem depois que haja uma conexão entre unidade móvel e MSS.

Como no mecanismo UFO e SCM, a inconsistência de dados ocorre pelo fato da captação dos dados no “meio” a qualquer momento, o que pode ocasionar a leitura de dados que não seria necessária a uma transação de atualização e que pode levar a uma execução cíclica. Neste ponto, o mecanismo SCM para resolver tal problema, redifunde as identidades dos dados que são atualizados no servidor de banco de dados. O UFO gera um *overhead* quanto a resolução de conflitos, pois re-difunde todos os ítens em conflitos. Enquanto que o SCM re-difunde apenas as identidades dos dados em conflito. Por outro lado, o SCM gera um *overhead* para manter o gráfico de serialização atualizado.

Já o algoritmo citado por PHATAK & BADRINATH (1995), utiliza bloqueio em duas fases para resolver conflitos. Mas estes bloqueio estão vinculados ao tamanho do *buffer* de bloqueio existente. Se o número de *slots* do *buffer* for de número igual ao tamanho de ítens do banco de dados, os conflitos serão resolvidos de forma otimista, seguindo apenas condições impostas a cada transação. Quanto menor a capacidade do *buffer*, maior será a utlização do bloqueio pessimista. A transação só é validada no momento da consolidação. Mas para isso todo o seu conjunto de leitura e escrita deve ser validado. Só após esta validação o conjunto de escrita da transação é escrito no banco de dados.

No mecanismo citado por JIN JING et al (1994), mais uma vez o bloqueio em duas fases é utilizado e, por sua vez, concede bloqueios de leitura imediatamente, e adia bloqueios de escrita até o tempo de consolidação. A novidade neste mecanismo é o fato da possibilidade do desbloqueio em local diferente da cópia no qual o bloqueio de leitura foi fixado. Com isso, o algoritmo restringe o número de mensagens extras de desbloqueio de leitura. Este mecanismo utiliza número de versão de cópia para verificar suas atualizações.

O que é percebido dentre os modelos é que os algoritmos que utilizam bloqueio não utilizam somente o bloqueio pessimista, mas sim um mecanismo híbrido com mecanismo otimista. Já o mecanismo otimista (pré-ordenação), necessita de mais uma funcionalidade para garantir a serializabilidade das operações.

Seguem as características dos algoritmos estudados neste capítulo.

Tabela 8 – Características dos algoritmos

Algorítmo	**Mecanismo**	**Resolução de Conflito**	**Atualização**	**Troca de Mensagem**
UFO p. 103	Bloqueio em duas fases e marcas de tempo nas transações de atualizações.	Utiliza a redifusão dos dados – tanto da transação de atualização quanto a de difusão.	Utiliza um espaço de trabalho, para depois atualizar o banco de dados real.	Caso a probabilidade de conflito for alta, a troca de mensagens será alta também, pois redifunde os ítens de dados em conflito.

Algorítmo	Mecanismo	Resolução de Conflito	Atualização	Troca de Mensagem
SCM p.105	Bloqueio em duas fases e marcas de tempo nas transações de atualizações.	Divide a transação de atualização e a de difusão, ao invés de redifundir todos os dados e atualização de gráficos de serialização na unidade móvel.		Quanto aos conflitos, é menor por só re-difundir a identificação dos ítens. Porém, para manter os gráficos atualizados a um aumento na troca de mensagens.
Bloqueio em Duas Fases p. 107	Bloqueio em duas fases. Mantém um buffer de bloqueio, sempre em tamanho maior que o possível numero de conflitos.	Prioriza acessos pessimistas a otimistas	Utiliza um espaço de trabalho, para depois atualizar o banco de dados real.	Cada escrita é precedida por uma leitura dos ítens de dados, isto encarece a troca de mensagens.
O2PL-MT p. 113	Bloqueio em duas fases.	Há troca de mensagens de permissões com o coordenador de transações que decide se um pedido de bloqueio de escrita pode ser adiado até o tempo de *commit*.	Utiliza um bloqueio com pretensão de escrita.	Coordenador coleciona mensagens de respostas e informações de desbloqueio dos locais. Com isso, a troca de mensagens se torna menor.

Algorítmo	Mecanismo	Resolução de Conflito	Atualização	Troca de Mensagem
R-W p.120	Pré-ordenação por marca de tempo com vetor de fila (informação adicional).	Marcas de tempo e vetor de fila são comparados para descobrir conflitos. As marcas de tempo são armazenadas na UM e na MSS.	A transação só consolida na reconexão com a rede.	Há uma rodada de troca de mensagens entre UM e MSS, verificando as marcas de tempo em cada conexão com a rede.

No próximo capítulo será concluído o estudo sobre controle de concorrência em ambiente móvel, colocando ainda as perspectivas para trabalhos futuros.

CONCLUSÃO

Os SGBDs (Sistemas Gerecniadores de Banco de Dados) são os mais utilizados para gerenciar informações e nesta nova modalidade de computação, "computação móvel", continua ganhando espaço, mas enfrenta problemas como a descontinuidade do fluxo de informações para seus clientes. Com isso, as propriedades ACID, que foram propostas para garantir a consistência do banco de dados, sofrem algumas alterações como inconsistência temporária entre unidade móvel e unidades fixas, ou servidores. Além de que, quando falamos em envio e recepção de dados sem fio, uma das formas mais comuns é a difusão dos dados. Com isso, as unidades móveis recebem ítens de dados independentemente de suas necessidades, e podem ler dados do "meio" a qualquer momento, além dos conflitos na execução das transações. A propriedade de isolamento também sofre alteração, pois em certos modelos a transação não precisa estar consolidada por inteiro para que outras operações enxerguem seus resultados.

No decorrer do trabalho foram expostos alguns tipos de transações móveis que tentam suprir as necessidades do banco de dados, assim como suportar a desconexão, a reintegração e a mobilidade. Percebemos nos estudos destes modelos, que os mecanismos de controle de concorrência utilizados nestes (Tabela 3 – capítulo VI), são principalmente bloqueio em duas fases e pré-ordenação por marca de tempo.

No capítulo VI, foram descritos 5 mecânismos de controle de concorrência. Podemos salientar que um dos ítens mais preocupantes quanto ao controle de concorrência no gerenciamento de transação, é o volume de mensagens trocadas entre as estações para manter o banco de dados consistente e atualizado. Considerando que o custo do tráfego de mensagens é bastante alto na computação móvel, e ainda mais caro na direção unidade móvel para unidade fixa. Outro item de grande ênfase é a resolução de conflitos entre transações móveis, transações de atualizações, transações de difusão, além das de leitura e escrita. Percebemos, no decorrer deste trabalho, que para manter um controle de concorrência que satisfaça os problemas da computação móvel, ele é composto de mecanismos híbridos, pessimistas e otimistas. Além do que, os mecanismos de pré-ordenação necessitam de um requisito a mais para não cair no problema de atualizações perdidas.

Como visto no capítulo VI, os mecanismos podem ser classificados, segundo a quantidade de mensagens trocadas entre estações, o tipo de resolução de conflito adotada e como são feitas as atualizaçoes no banco de dados servidor, ou nos demais banco de dados, caso o sistema seja distribuído ou mutibanco de dados.

7.1 Relevância do Trabalho

A relevância deste trabalho para a área de sistemas gerenciadores de bancos de dados móveis é apresentar as necessidades de se adotar mecanismos de controle de concorrência que suportem a mobilidade, desconexão e reintegração dos ítens de dados pertencentes às estações no ambiente móvel. Levando sempre em consideração a perfomance e consistência do banco de dados.

7.2 Perspectivas Futuras

Tendo em vista que um dos fatores primordiais para o controle de concorrência em ambientes móveis é a minimização de mensagens trocadas, a garantia de consistência do banco de dados e a performance do banco. Um estudo que poderia contribuir para o desenvolvimento da utilização dos bancos de dados em largura de faixa estreita, seria a análise comparativa de algorítmos de controle de concorrência, levando em consideração os fatores citados. Como parâmetros para esta verificação, poderiam ser utlizados os seguintes:

- Tamanho do Banco de Dados
- Número de Clientes
- Largura do Canal de Difusão
- Tamanho do Item de Dados
- Tamanho do ID do Item de Dados
- Número de Ítens nas Transações, tanto móveis quanto de atualizações,
- Intervalo entre atualizações
- Tipo de arquitetura do banco de dados

Por meio dessa análise proporcionar a indicação de que algorítmo contribuiria para a melhor performance para o banco de dados e em que tipo de arquitetura.

REFERÊNCIAS BIBLIOGRÁFICAS

1. ACHARYA A., BADRINATH B. R. **Delivering Multicast Messages in Networs With Mobile Hosts.** 13th International Conference on Distributed Computing Systems, 1993.

2. ADIBA, M.; SERRANO-ALVARADO, P.; RONCACIO, C. L. **Mobile Transaction Supports for DBMS: An Overview.** LSR-IMAG Laboratory, 2001.

3. ALONSO RafaeL; FRANKLIN M.; ZDONIK Stanley; ACHARAYA SWARUP. **Broadcast Disks: Data Management for Asymmetric Communication Environment***s.* Proceedings of the ACM SIGMOD Conference, San Jose, CA, Maio 1995.

4. ANINDYA Datta; DEBRA E; VANDER Meer; ASLIHAN, Celik; VIJAY, Kumar. **Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users** . Electronic Edition. V. 24, N. 1, pp. 1-79, Março, 1999,

5. ARAÚJO, Luciano V. De; FERREIRA, João Eduardo. **Cachê semântico para Computação Sem Fio Baseado na Abstração de Composição dos Dados**. WorkSIDAM, Workshop de Sistemas de Informação Distribuída de Agentes Móveis. São Paulo, Outubro, 2000. p. 83-89

6. AU Mei-Wai, CHAN Edward, LAM Kam-Yiu**. Concurrency Control for Mobile Systems with Data Broadcast**. Journal of Interconnection Networks, 1999.

7. BADRINATH B. R.; IMIELINSKI T.; **Replication and Mobility,** 2nd IEEE Workshop on Management of replicated data, Novembro. Pp. 1-12, 1992,

8. BADRINATH, B. R.; PHATAK, SHIRISH. H.. **An Architecture for Mobile**

Databases. Relatório Técnico DCS-TR-351, pp. 1-8, 1998.

9. BADRINATH, B. R.; PHATAK, SHIRISH. H.. **Bounded Locking for Optimistic Concurrency Control**. Department of Computer Science, University Rutgers, Piscatway, 1995

10. BARBARÁ, D. **Mobile Computing and Databases – A Survey.** IEEE Transactions on Knowledge and Data Engineering, vol 11, n. 1, Fevereiro, 1999

11. BERENSON H., BERNSTEIN P.A., GRAY J., MELTON J., O'NEIL E.J., O'NEIL P.E. **A Critique of ANSI SQL Isolation Levels**, Proceedings of ACM SIGMOD Conference, pp 1-10, 1995.

12. BERNSTEIN P.A, GOODMAN N., **Concurrency Control in Distributed Database Systems, ACM Computing Surveys**, 13 (2), pp 185-221, 1981

13. BRODIE, M.; STONEBRAKER, M.; **Migrating Legacy Systems: gateways, interfaces & the incremental approc**h. São Francisco, CA: Morgan Kaufmann Publishers, Inc., 1995.

14. CASANOVA, Marco Antionio, MOURA, Arnaldo Vieira. **Princípios de Sistemas de Gerência de Banco de Dados Distribuídos** Ed. Campus, 3ª ed. Rio de Janeiro , 1985.

15. CHRYSANTHIS Panos K,. **Transaction Processing in Mobile Computing Environment**, in Proceedings of the IEEE, Workshop on Advances in Parallel and Distribuited Systems, pp 77-82, Outubro, 1993

16. CRYSANTHIS, P. K., RAMAMRITHAM K., **Syntesis of Extended Transaction Models Using ACTA.** Technical Report 93-05, University of Pittsburg, 1993

17. DATE, C.J. **Introdução A Sitemas De Banco De Dados**, Ed. Campus, Rio de

Janeiro, 7ª Edição, 2000

18. DAVIDSON S.B**., Optimism and Consistency in Partitioned Distributed Database Systems**, ACM Transactions on Database Systems, vol. 9, n 3, pp 456-481, 1994

19. DIRCKZE A Ravi , LE Gruenwald., **A Pre-Serialization Transaction Management Technique for Mobile Multidabases**. To appear in ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications, pp 311-211, 2000

20. DUNHAM M. H., HELAL A., AND BALAKRISHNAN S., **A Mobile Transaction Model That Captures Both the Data and Movement Behavior**. To appear in ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications, 1997

21. FAIZ M., ZASLAVSKY A**, Database Replica Mangement Strategies in Multidatabase Systems with Mobile Hosts**, in 6th International Hong Kong Computer Society Database Workshop, 1995.

22. FERREIRA, E. JOÃO; FINGER, MARCELO. **Controle de concorrência e distribuição de dados: a teoria clássica, suas limitações e extensões modernas**. Escola de computação 2000

23. GRAY J., HELLAND P., O'NEIL P. E., SHASHA D., **The dangers of replication and a solution**. Proceedings of ACM SIGMOD, pages 173-182, Junho 1996.

24. HUANG Y., SISTLA P, WOLFSON O. **Data Replication for Mobile Computers**, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1994.

25. IMELINSKI T., BADRINATH A. **Mobile Wireless Computing: Solutions And Challenges in Data Management.** Technical Report, Rutgers University, 1993

26. IMIELINSKI, T., VISHWANATH, S., and BADRINATH, B. R. **Data on air: Organization and Access**. IEEE Trans. Knowl. Data Eng. 9, 3 (May/June), 353–372. . 1997

27. IMIELINSKI, T.; BADRINATH, B. R. .**Data Management Issues in Mobile Computing** , *Commun* ACM, 37 (10): 18-28 , Outubro, 1994

28. JING, Jin, BUKHRES, Omran, ELMAGARMID, Ahmed**, Distributed Lock Management for Mobile Transactions**, Departament of Computer Sciences, Purdue University, Technical Report, 1994

29. KUMAR V. DUNHAM M. H., **Defining Location Data Dependency,Transaction Mobility and Commitment**. TECHNICAL REPORT 98 CSE-01, Dallas, Southern Methodist University, Fevereiro, 1998

30. KUNG H.T., ROBINSON J.T., **On Optimistic Methods of Concurrency Control**. ACM Transactions on Database Systems, 6(2), pp 213-226, 1981

31. LAM Kam-Yiu, CHAN Edward, AU Mei-Wai, **,. Broadcast of Consistent Data to Read-Only Transactions from Mobile Clients,** in Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Aplications, New Orleans, 1999.

32. LEE Sangkeun, HWANG, Chong-Sun, YU, Heonchng, **Supporting Transactional Cache Consistency in Mobile Database Systems,** MobiDE Seattle WA USA, ACM 1999.

33. LEE Y.,MOON S, **Commit-Reordering Validation Scheme For Transaction Scheduling in Client-Server Based Teleputing Systems: COREV,** Proceedings of the 6th International Conference on Information and Knowledge Management,

pp 50-66, 1997

34. LU Q., SATYANARAYANAN M. **Isolation-Only Transaction for Mobile Computing.** ACM Operation Systems Review, 28(2): 81-87, 1994

35. LUBINSKI A., HEUER A., **Configured Replication for Mobile Applications**. Proc. Der 12. GI – Workshop "Grundlagen Von Datenbanken", plön 13-16. Junho, 2000

36. MADRIA S.K., **Timestamp Based Detection and Resolution of Mutual Conflicts in a Distributed Systems**, in the proceedings of IEEE for the 8th International Conference and workshops for the Database and Expert System Applications, 1997.

37. MADRIA S. K., BHARGAVA B., **On the Correctness of a Transaction Model for Móbile Computing**. 9th International Conference and Workshop on Database and Expert Systems (DEXA), 1998

38. MADRIA, S. K., **Transaction Models for Mobile Computing**, in Proceeding of 6th IEEE Singapore International Conference on Network, World Scientific, Julho, 1998

39. MATEUS, Geraldo Robson; LOUREIRO, Antonio A. Ferreira. **Introdução a Computação Móvel**. DCC/IM, COPPE/Sistemas, NCE/UFRJ, 11ª. Escola de Computação, 1998

40. MOLINA, H. G.; ULLMAN, J. D.; WIDOM, J. **Implementação de Sistemas de Bancos de Dado**s. Rio de Janeiro, Campus, 2001.

41. ÓSZU Tamer M., VALDURIEZ Patrick,**Pirncipios de Sistemas Distribuidos** – Ed. Campus – Tradução(2ª ed. Americana) ,Rio de Janeiro, 2001

42. PHILIP S Yu, DIAS Daniel M E LAVENBERG STEPHEN S **On the Analytical of Database Concurrency Control.** ACM –Jornal of the Association Computing Machinery, vol 40 nº4 p831-872 , Setembro 1993.

43. PITOURA EVAGGELIA; BHARGAVA BHARAT. **Revising TransactionConcepts for Mobile Computing**, in Proceedings of the IEEE, Workshop on Mobile Systems and Applications, Canadá, Dezembro, 1994

44. PITOURA, Evagelia; SAMARAS, George. **Data Management for Mobile Computing**, Kluwer Academic Publishers, 1998

45. PITOURA, Evaggelia; BHARGAVA, Bharat. **Maintaining Consistency of Data in Móbile Distributed Environments**, in Proceedings of the 15th International Conference on Distributing Com putting Systems, Vancouver, Canadá, Maio, 1995

46. PITOURA, Evaggelia; BHARGAVA, Bharat., **Building Information Systems for Mobile Environments**, Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, ACM Press. p. 371-378, Novembro, 1994

47. RASHEED A., ZASLAVSKY A. **Ensuring Database Availability in Dynamically Changing Mobile Computing Environment**, in Proceedings of the 7th australian Database Conference, Melborne, 1996

48. RAVINDRAN K., SHAN K. **Casual Broadcasting and Consistency of Distributed Data**, in 14th International Conference on Distributed Computing Systems, pp 40-47, 1994.

49. S. ACHARYA, R. ALONSO, M. FRANKLIN, and S. ZDONIK, **Broadcast Disks: Data Management for Asymmetric Communications Environments**, in: Proc. ACM SIGMOD International Conf. on Management of Data, Junho,

1995.

50. SEYDIM, Ayse Yasemn. **An Overview Of Transaction Models In Mobile Environments**, Department of Computer Science and Engineering, Southe Methodist University, Dalas, 1999.

51. SHETH, A.P.; LARSON, J. A.. **Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases**. ACM Computing Surveys, vol. 22, n. 3, Sept, 1990.

52. SHNMUGASUNDRAM J., NITHRAKASHYAP A., SIVASANKARAN R. RAMAMRITHAM K**. Efficient Concurrrency Control for Broadcast Environments**, Technical Report, 1997-062, Departament of Computer Science, University of Massachusetts, Amherst, 1998.

53. SILBERSCHATZ Abraham, KORTH Henry, SUDARSHAN S, **Sistemas de Banco de Dados -** ed. Makron Books - São Paulo - 3ª edição – 1999.

54. SILVA, S. D.. **Sistemas de Bancos de Dados Heterogêneos: Modelo de Execução de Gerência de Transaçõe**s. Tese de doutorado em informática, Dept. de Informática PUC-Rio. Rio de Janeiro, 1994.

55. SONG Siang Wun et al**. SIDAM – Sistemas de Informação Distribuídos para Agentes Móveis** . Departamento de Ciência da Computação, IME São Paulo e Departamento de Ciencia da Computação e Estatistica ICM São Carlos, São Paulo, Junho, 1999.

56. TEWARI, R.; GRILLO, P.. **Data Management for Mobile Computers in Internet**, Proceedings of the 23rd ACM Computer Science Conference, Março, p. 246-252, 1995.

57. THOMASIAN A., **Distributed Optimistic Concurrency Control Methods for**

High-Performance Transaction Processing, TKDE, pp 173-189, 1998

58. WALBORN Gary, CHRYSANTHIS Panos .K.SUPPORTING **Semantics-Based Transaction Processing in Mobile Database Applications**. in Proceeding of 14th IEEE Symposium on Reliable Distributed Systems, pp.31-40, setembro, 1995

59. WALBORN, G. D.; CRYSANTHIS, P. K. **Transaction Processing in PRO-MOTION.** In 14 th ACM Annual Symposium on Applied Computing, San Antonio Tx, Fevereiro, 1999.

60. WALBORN, GARY. D.; CRYSANTHIS, PANOS. K. **PRO-MOTION: Management of Mobile Transaction** In n Proceedings of the Symposium on Applied Computing, pp 101-108, 1997.

61. WU K. L., YU P.S., CHEN, **Energy-Efficient Caching For Wireless Mobile Computing,** Proceedings of the 12th International Conference on Data Engineering, pp 336-343, 1996

62. XUAN P., GONZALEZ O,. FERNANDES J., RAMAMRITHAM K**. Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments**. in Proceedings of 3rd IEEE Real-Time Technology Application Symposium, 1997

63. YEO L. H., ZASLAVSKY A., **Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment**, Proceedings of Distributed Computing Systems, pp 372-379, Junho,1994

APÊNDICE A – COMPUTAÇÃO MÓVEL

COMPUTAÇÃO MÓVEL

A computação móvel dirigi-se para migrar o mundo da computação para um ambiente móvel e este contexto é caracterizado, principalmente, por dois componentes: portabilidade e conectividade. Portabilidade, é a habilidade para portar computadores de ambiente de mesa convencional para os mais variados ambientes. A conectividade, é a habilidade para se conectar a recursos externos e ter acesso a dados externos. Essa tecnologia de comunicação de dados sem fios tem um papel significante nos dias atuais e maior terá num futuro próximo porque oferecerá o que se está chamando de conectividade onipresente ou ubíqua, isto é, conectividade em qualquer lugar a qualquer hora.

Segundo TEWARI & GRILLO (1995), computação móvel é um tipo de computação distribuída, em que a visão de diversos computadores autônomos é transparente aos usuários e a distribuição das atividades é feita automaticamente pelo sistema. Computação móvel amplia o conceito tradicional de computação distribuída, pois a comunicação sem fio elimina a necessidade do usuário manter-se ligado através de cabos a uma infra-estrutura fixa (MATEUS & LOUREIRO, 1998). Um sistema de Computação Móvel é um tipo particular de sistema distribuído, onde existem computadores fixos e computadores móveis. Os computadores fixos constituem a parte tradicional, formada por uma estrutura de comunicação fixa com computadores estáticos, e esta é interligada a uma parte móvel, representada por uma área ou célula onde existe a comunicação sem fio dos elementos computacionais móveis.

Os sistemas móveis de comunicação se baseiam, em sua grande maioria, por meio de ondas eletromagnéticas, principalmente em transmissão via rádio ou sinais em freqüência muito alta. Exemplos de tecnologias usadas atualmente em comunicação móvel sem fio são: redes de telefonia celular, rádio, satélite, *bluetooth* etc.

Neste capítulo são discutidos conceitos de computação móvel, arquitetura e modelos computacionais. Também são apresentados aspectos importantes em comunicação sem fio como os conceitos de *handoff*, desconexão e mobilidade.

1.1 Uma Arquitetura de Computação Móvel

Um conjunto de computadores de propósito gerais (PC, estações de trabalho, etc.) é interconectado por uma velocidade alta, em uma rede interligada com cabos. Esta é a arquitetura tradicional. Na arquitetura móvel, considera-se, além dessa estrutura tradicional, computadores que estão em mobilidade, aqui definidos como unidades móveis , computadores móveis ou *mobile hosts*.

Em uma possível arquitetura móvel, são encontradas algumas classificações de computadores. Computadores são classificados como *hosts* fixos (FH - *Fixed Hosts*) e estações base (BS - *Base Stations*) ou estações de apoio móvel (MSS - *Mobile Support Stations*).

Vários equipamentos de computação móvel (*laptops, handhelds*, PDAs, etc), chamados de *hosts* móveis (MH - *Mobile Hosts*) ou unidades móveis (MU – *Mobile Units*), são conectados a uma estação base por canais de comunicação de uma rede sem fios. Os computadores fixos não se comunicam diretamente com as unidades móveis. Ao invés disso, os computadores fixos se comunicam com as estações base que, por sua vez, estão ligadas às unidades móveis como mostrado na Figura 1.1. As estações base são equipadas com uma interface para rede sem fios e provêem um portal de comunicação entre a rede fixa e a rede sem fios (KUMAR & GUNHAM, 1998).

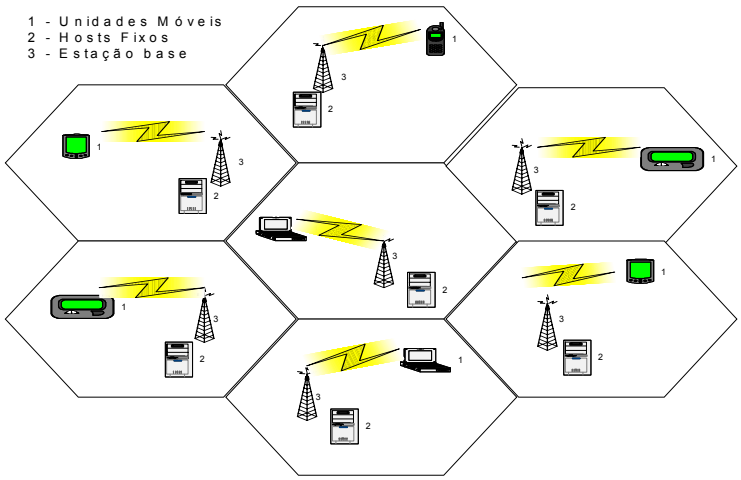

Figura A1.1 – Arquitetura de Computação Móvel

1.1.1 Elementos Computacionais Móveis

Computadores móveis ou unidades móveis (PDAs, *handhelds, Laptops*, celulares) têm como principal característica o fato de serem leves, pequenos e dependentes de bateria. Isso faz com que os elementos móveis tenham, normalmente, menos recursos do que computadores convencionais. Dentre este recursos citamos:

Elementos móveis são equipados com pouca memória RAM, processadores mais lentos e memória auxiliar pequena; a interface com o usuário é mais limitada. O monitor é menor ou tem menos capacidade (*buffer* de vídeo) e os dispositivos de entrada de dados são menores e limitados; dependem de energia fornecida por bateria, que tem uma capacidade limitada.

1.1.2 Computadores Fixos

Os computadores ou estações fixas fazem parte de uma rede fixa cuja localização e conectividade não mudam. Algumas destas máquinas estáticas fazem o papel de um transceptor estacionário chamado de Estação de Suporte Móvel ou Estação Base.

Estas estações base são utilizadas para gerenciar a localização das unidades móveis dentro de uma célula e assim permitir a comunicação entre as unidades móveis ou fixas.

1.2 Tecnologias Sem Fio

Uma rede de comunicação pessoal (PCN, *Personal Communication Network*) representa a infraestrutura da transmissão em rede. Esta infraestrutura visa prover cobertura de comunicação sem fio onipresente. Tal cobertura habilita o acesso aos recursos da rede, usando diferentes tipos de mídia de comunicação e independe da localização do usuário (PITOURA & SÂMARAS, 1998).

Uma arquitetura de rede de comunicação pessoal, normalmente, está baseada parcialmente numa arquitetura de telefonia celular existente. Em geral, a configuração da rede consiste em um *backbone* de rede fixa estendida a um número de elementos móveis que se comunicam diretamente com o transceptor. O tamanho da célula é

definido pela área coberta pelo sinal de um transceptor. Este tamanho varia muito, dependo do dispositivo utilizado para transmitir e receber informação. Por exemplo: um feixe do satélite cobre em média 400 milhas (1 milha = 1.609 metros), uma antena celular cobre poucos kms, e uma LAN sem fio *(wireless LAN)* cobre a comunicação de um edifício.

Nas seções seguintes serão apresentados alguns conceitos de algumas das tecnologias sem fio existentes atualmente.

1.2.1 Tecnologia Celular

Sistemas de telefonia celulares são projetados para prover serviço de voz para indivíduos movimentando-se em ruas e rodovias, geralmente usando alto poder de transmissão. No entanto, atualmente estes sistemas fornecem poucos serviços para unidades móveis (PDAs, *handhelds, Laptops* etc). Os equipamentos de comunicação normalmente usados em telefonia celular, possuem baixo poder de transmissão e suas antenas são pouco eficientes.

A primeira geração de sistemas de telefonia celular era baseada em tecnologia analógica AMPS (*Advance Mobile Telephone System*). Nesse modo, a voz é transmitida em canais de banda estreita em circuito de chaveamento (*switched*) que usam modulação de freqüência (FM). Esta geração de sistemas celulares fornece fala analógica e taxa baixa e ineficiente de transmissão de dados.

A segunda geração de sistemas celulares emprega modulação digital e avançada capacidade de processamento. Nessa arquitetura, o sinal de voz é digitalizado, armazenado em um *buffer* na estação e então, transmitido pela alocação de *slots* de tempo, com intervalos distintos para transmissão e recepção. Nesta segunda geração, são oferecidos uma variedade de serviços de dados como *paging*, fac-símile e outros. Porém, geralmente as taxas de erros são altas e as taxas de dados são baixas. Mas nesta geração, as estações móveis excutam mais controle que a primeira geração, inclusive o controle do processo de *handoff*.

1.2.2 Pacote Público de Rádio (*Public Packet Radio*)

Pacote público de radio prevê eficiência de canal por transmitir os dados em rajadas longas. Esses sistemas podem ser caracterizados por proverem alta mobilidade, velocidade baixa na taxa de transmissão de dados digitais, longo alcance tanto para pedestres quanto para veículos, contrastando com pacote de serviço de *switch*, que transmitem dados em rajadas curtas e que são seguidos freqüentemente por períodos de inatividade de transmissão.

Os componentes são múltiplos transmissores em estações base conectados a redes locais, por sua vez interconectados a um nó nacional. Dispositivos móveis são conectados a rede ARDIS (*Advanced Radio Data Information Service*) através de *modens* próprios. Os servidores e os *hosts* são conectados ao nó central.

1.2.3 LAN's Sem Fio (*Wireless LANs*)

Redes locais sem fio *(Local Area Network Wireless)* são caracterizadas pela comunicação de alta velocidade, mas com clientes de baixa mobilidade. O propósito das redes locais sem fio não é suportar mobilidade, mas prover pontes sem fio entre redes convencionais. Estas redes são projetadas para operarem em uma região limitada.

Em geral, a alta velocidade das redes locais sem fio se enquadram em duas categorias: uma desenvolvida para operar em freqüência de rádio e outra desenvolvida para região de ondas infravermelho.

Os componentes essências das redes locais sem fio são os mesmos ou similares aos das redes locais convencionais. A diferença está na substituição de interface de redes *Ethernet* e *Token Ring* por cartões de interface de rede – NIC's (*Network Interface Cards*). Estes são normalmente na forma PCMCIA (ou outro tipo de interface para equipamentos portáteis) para notebooks. Outro componente importante é a antena para captar sinas de rádio, que pode ser Direcional ou Unidirecional, e pontos de acesso ou módulos controladores de cartões NIC.

1.2.5 Sistemas de Satélite

Sistemas de comunicação via satélite possibilitam a cobertura de amplas áreas que se estendem, inclusive sobre oceanos e áreas mais remotas. Apresentam uma alta capacidade para transmissão em *broadcast*. Por outro lado, o problema de segurança é bastante grave, uma vez que qualquer unidade receptora pode captar o sinal.

Basicamente os satélites se estabelecem em três níveis. Os satélites de baixa órbita LEO (*Low Earth Orbit*), posicionados em torno de 1000 km de altitude. Os de órbitas médias MEO (*Medium Earth Orbit*) que estão a 10000 km de altitude. E os satélites de órbita elevadas ou geoestacionário GEO (*Geosynchronous Earth Orbit*) estão situados à aproximadamente 36000 km de altitude e em regiões próximas a linha do equador.

1.2.6 *BlueTooth*

A tecnologia *BlueTooth* permite a comunicação sem fio entre aparelhos eletrônicos que podem ser computadores, telefones celulares, PDA´s (*Personal Digital Assistans*), equipamentos de escritório e dispositivos móveis.

Um microchip muito pequeno, possuindo um transmissor de rádio, é inserido em um dispositivo digital. A tecnologia *BlueTooth* realiza todas as conexões instantaneamente sem um único centímetro de cabo. Isso facilita a rápida e segura transmissões de dados e voz ; até mesmo quando os dispositivos não estão numa linha direta de visão.

Esta comunicação se realiza através de um dispositivo de enlace de rádio na frequência de 2.4 GHz, que não necessita de licença e está disponível em quase todo o mundo.

A tecnologia *BlueTooth,* quando usada na conectividade de um ambiente interno, proporciona uma área personalizada, com conexões sem fio tão simples quanto acender as luzes. Todos os dispositivos eletrônicos desse ambiente irão se comunicar espontaneamente, para assim oferecer três grandes vantagens:

- **Ponto de acesso de voz e dados.** A tecnologia *BlueTooth* simplifica o acesso a outras redes. Isso é feito reconhecendo e conectando diferentes tipos de

redes através de uma conexão *BlueTooth*. É possível se conectar de modo fácil e instantâneo à internet, via telefonia móvel, assim como qualquer outro dispositivo de conexão *BlueTooth* do gênero.

- **Substituição dos cabos.** A tecnologia *BlueTooth* elimina a necessidade das problemáticas conexões a cabo. Pode-se simplesmente enviar e receber e-mail do seu computador móvel através do seu telefone móvel, mesmo quando não estão em linha direta de visão.

- **Redes ad-hoc personalizadas**. Redes *ad-hoc* se caracterizam pela ausência da necessidade de uma infra-estrutura de re-transmissão de dados de um ponto ao outro, ou seja, a comunicação entre dois dispositivos pode ser feita de modo direto, sem uso de intermediários.

Quanto a velocidade e segurança, a tecnologia *BlueTooth* foi projetada para ser totalmente funcional mesmo em ambientes com altos níveis de ruídos, e sua transmissão de voz é audível sob severas condições. Essa tecnologia provê uma taxa de transmissão muito alta e todos os dados são protegidos por avançados métodos de correções de erros, tais como códigos criptografados e autenticação de rotinas para a privacidade do usuário.

1.3 Características da Computação Móvel

Os sistemas móveis apresentam, como grande vantagem, a mobilidade, permitindo ao usuário o acesso à informação ou serviços e à independência de cabeamento. Porém, existem algumas características que devem ser ressaltadas, como desconexão, sincronização, handoff, baixa conectividade. Essas características influenciam amplamente a usabilidade dos sistemas móveis, pois podem impor severas restrições sobre a qualidade de transmissão de dados (MATEUS & LOUREIRO, 1998).

1.3.1 Desconexão

Segundo MATEUS E LOUREIRO (1998), na comunicação sem fio, as desconexões são freqüentes e ocorrem quando uma unidade móvel se movimenta fora da área de cobertura de todas as células. Essas desconexões podem ser caracterizadas de formas diferentes. As desconexões podem ser voluntárias ou involuntárias. As voluntárias são no caso do usuário ou o computador móvel impedirem intencionalmente o acesso à rede. Isso pode ocorrer para diminuir o custo da tarifa da comunicação, para diminuir o consumo de energia ou pela largura da banda não condizer com a necessidade de transmissão (PITOURA & SAMARAS, 1998). E involuntárias, quando a unidade móvel entra em uma região onde não existe acesso à rede fixa por falta de um canal de comunicação ou cobertura neste local.

Quando esta desconexão é voluntária, considera-se uma falha planejada que pode ser antecipada e preparada. Quando este planejamento acontece, dados são movidos para a unidade móvel para que ela possa ter autonomia de operação durante o período que estiver desconectada. A ação de pré-carregar, no sentido de acumular os dados antecipadamente, é chamada de ***hoarding*** .

A operação de desconexão pode ser descrita como a transição entre três estados (PITOURA & SAMARAS – 1998). São eles: Reintegração, *hoarding* e desconexão, como mostrado na Figura 1.2.

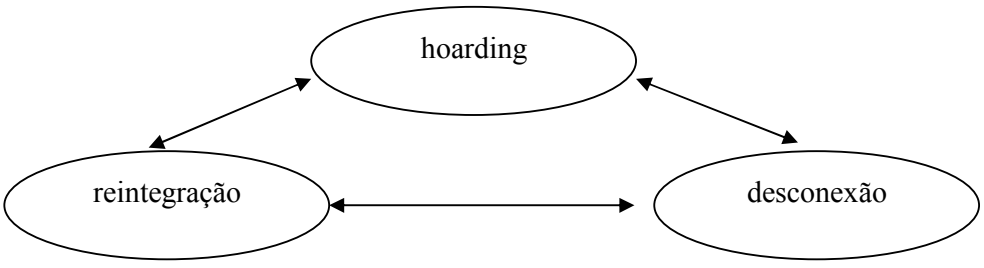

Figura A1.2 – Transição de Estados
Fonte: PITOURA & SAMARAS, (1998)

Antes da desconexão, a unidade móvel entra em um estado de *hoarding*. Nesse estado, os dados utilizados durante a operação serão pré-carregados para a unidade móvel, tornando-se inacessíveis para outros *site*s. Os dados que serão transferidos para

a unidade móvel podem ser determinados pelo usuário, ou de acordo com o passado histórico de acesso aos dados e à necessidade futura que eles terão, em função das aplicações que rodam nas unidades móveis.

No momento em que uma unidade móvel se desliga da estação base, ela entra em estado de desconexão. Nesse estado, apenas dados locais podem ser utilizados. Caso alguma solicitação seja feita e os dados não estejam disponíveis localmente, uma mensagem de erro é retornada. Estas solicitações podem ser inseridas em uma fila e serem atendidas na reconexão.

A partir do momento em que a unidade móvel se reconecta com a estação base, ela entra no estado de reintegração. Nesse estado, atualizações efetuadas na unidade móvel são reintegradas com as de outros *site*s. Geralmente esta operação é realizada pela re-execução do *log*[2] na rede fixa.

1.3.2 *Handoff*

Com a movimentação dos componentes móveis, esses podem atravessar o limite de uma célula e entrar na área coberta por uma estação base diferente. Essa transição entre células provoca um processo chamado *handoff*. Esse processo entre duas bases adjacentes garante a realização de uma transação enquanto usuários se movimentam de uma célula para outra, mantendo a conectividade de uma unidade móvel com a rede fixa (PITOURA & SAMARAS, 1998).

[2] O termo *log* pode ser entendido aqui como um diário, ou seja, um repositório de anotações sobre atividades, no caso, do banco de dados.

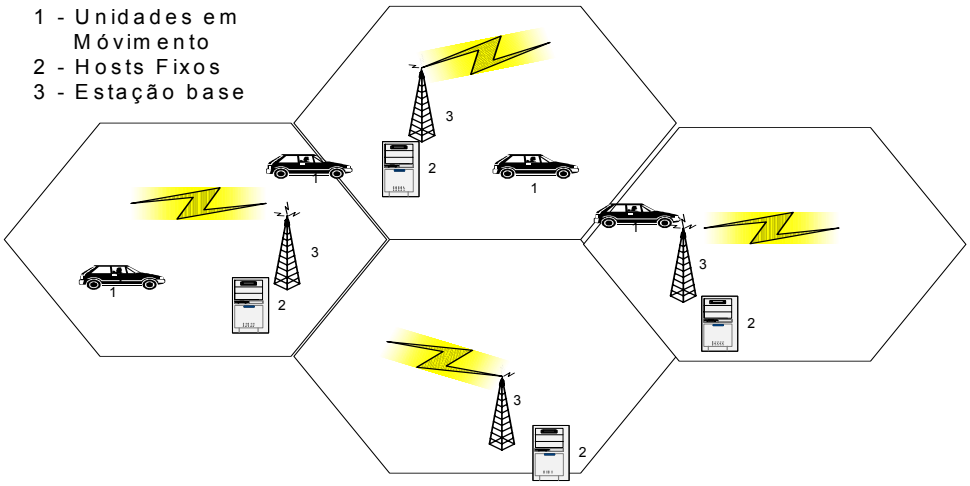

Figura A1.3 - *Handoff*

Isso implica que uma estação base pode detectar qualquer entrada e qualquer saída de usuários dentro e fora da sua célula, bem como as informações pertencentes a cada unidade móvel devem ser transferidas para a estação base da nova célula, conforme a Figura 1.3. Durante o processo de *handoff*, as estações base são responsáveis pela comunicação sem fio das unidades móveis, sendo este processo realizado de forma transparente ao usuário.

1.3.3 Mobilidade

Denomina-se mobilidade a capacidade de uma unidade móvel se mover pela rede. Além da unidade móvel mover-se, os seus pontos de acesso à rede fixa movem-se também (PITOURA & SAMARAS, 1998). Com isso, ocorre a inexistência de um endereço fixo na rede, o que traz a necessidade de realização de um gerenciamento de localização.

Com essa mobilidade entre células, há ainda a possibilidade de descontinuidade e interferência na comunicação, assim como uma largura de banda reduzida, o que pode ocasionar severas restrições sobre a qualidade de transmissão de dados (SONG. et al 1999).

Além desses fatores sobre a mobilidade, temos a heterogeneidade das redes e equipamentos a que esta unidade móvel irá se conectar, as limitações de recursos como energia por tempo limitado, devido à utilização de baterias nas unidades móveis, baixa capacidade de processamento e armazenamento.

Quanto à mobilidade, serão apresentados os seguintes aspectos relevantes: localização, heterogeneidade, protocolos e algoritmos.

1.3.3.1 Localização

O grande problema da localização das unidades móveis é a adição do custo à comunicação. A eficiência no atendimento de uma chamada em um sistema de comunicação móvel está diretamente ligada a rápida localização do usuário chamado dentro do sistema. Para que ocorra esta localização, é executada uma monitoria sobre a mobilidade dos usuários através das áreas de registro de localização. Nessas áreas, são mantidos os registros de identificação e tarifações do usuário. A manutenção das áreas de registro de localização depende de duas operações: atualização da localização e a pesquisa da unidade móvel. Porém, o processo para manter esta constante atualização e pesquisa pelo usuário é custoso e gera uma sobrecarga no sistema.

Visando a otimização da localização, surgiram duas políticas: uma atualize-sempre e outro nunca-atualize.

- *Atualize-sempre* - cada unidade móvel realiza uma operação de atualização de localização toda vez que entra em uma nova célula.
- *Nunca-atualize* - uma unidade móvel nunca atualiza sua posição.

Porém, a primeira política gera uma taxa de atualização alta, mas a operação de pesquisa se torna baixa. Quanto a segunda política, o custo com a localização da unidade é nulo, entretanto o processo de pesquisa é bastante elevado.

Várias políticas de monitoração sobre a localização de unidades móveis têm sido propostas (MATEUS & LOUREIRO, 1998). Algumas dessas propostas exploram a velocidade de movimentação, distância percorrida e tempo (ver a Figura 1.4). No entanto o problema de localização é bastante complexo e ainda não há um padrão que

resolva todos os problemas encontrados, ou que as aplicações possam usar para a localizar as informações. Essa é uma área de grande pesquisa ainda.

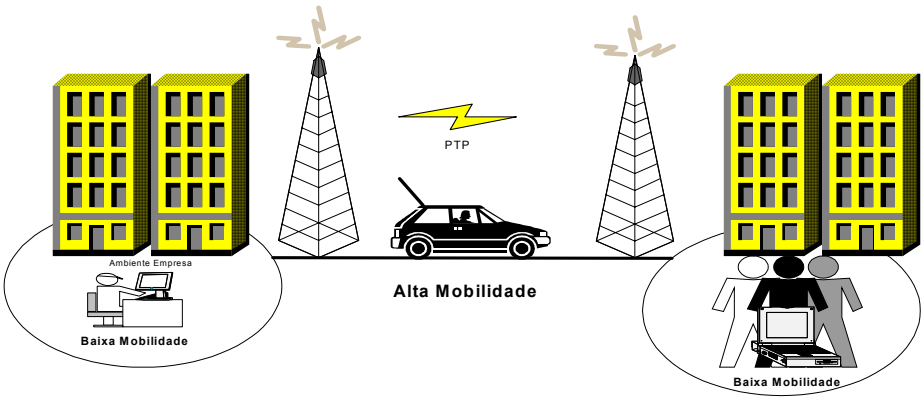

Figura A1.4 – Mobilidade Baixa e Alta

Para tentar resolver os problemas com a localização, se deve tentar minimizar o custo, projetando algoritmos e estruturas de dados eficientes, planos de execução de consultas para localização de unidades móveis, e que levem em consideração as características do ambiente (MATEUS & LOUREIRO, 1998), (PITOURA & SAMARAS, 1998).

1.3.3.2 Heterogeneidade

Com a mobilidade da unidade móvel pela rede ou células, a conectividade , pode sofrer variações, pois nem todos os elementos que compõe a rede são iguais. Existe a heterogeneidade de desempenho e confiança entre elementos computacionais.

Um exemplo de heterogeneidade é quanto à transmissão de dados: a conectividade em um ambiente interno é mais confiável e a largura de banda é maior. Já em ambiente externo, ao ar livre, a largura de banda é menor e existe a propensão a interferências, o que resulta em uma conexão não tão confiável.

Outra característica é a mudança no número de elementos constantes em uma determinada célula em um determinado instante. Isso implica na carga de uma estação base, bem como na largura de banda.

Existe ainda a variabilidade na provisão dos serviços específicos como impressoras e outros, ou de recursos disponíveis a uma unidade móvel (MATEUS &

LOUREIRO, 1998), (PITOURA & SAMARAS, 1998). A heterogeneidade dos elementos móveis ainda é um desafio para que haja uma comunicação de dados confiável e estável.

1.4 Modelos de Computação Móvel

A mobilidade gera estas restrições, que induz os recursos computacionais a uma disponibilidade variada. Estas restrições têm um grande impacto na determinação e estrutura das aplicações de computação móvel. Isso motiva o desenvolvimento de modelos de computação novos que são projetados considerando os tipos de funcionalidades associados às unidades móveis. Alguns fatores devem ser considerados quanto aos modelos computacionais: características do canal de comunicação, tipo de comunicação e classes de falhas que podem ocorrer nos elementos de processamento

1.4.1 Modelo Cliente Móvel/Servidor

No modelo Cliente Móvel/Servidor, o papel de cliente é desempenhado pela unidade móvel que requisita serviços de uma outra unidade que está fixa na rede, chamada de servidor. Existem alguns casos em que os dados e as funcionalidades são distribuídas entre vários servidores fixos que, para atenderem a uma requisição do cliente, podem estabelecer uma comunicação entre si (PITOURA & SAMARAS, 1998).

Um mecanismo muito utilizado é a replicação do servidor, pois aumenta a disponibilidade do servidor no caso de falhas da rede ou do servidor. Com isso, também aumenta a escalabilidade e o desempenho para acomodar clientes com alta mobilidade.

Existem alguns problemas que devem ser considerados e tratados. A desconexão do cliente móvel é um deles. Quando ocorre esta desconexão, o cliente não pode parar a operação, deve emular a funcionalidade do servidor, até que seja conectado novamente.

Outro problema é a necessidade da definição do tipo de mecanismo de comunicação utilizado. Dentre estes mecanismos, existe a possibilidade de se executar a troca de informações direta entre cliente e servidor (ver Figura 1.5). No entanto esta

abordagem não é adequada para redes lentas e não confiáveis. A abordagem mais adequada para esse tipo de rede é um mecanismo menos direto, ou seja, mecanismo de indireção onde as mensagens são enfileiradas nos dois pontos, tanto no servidor quanto no cliente.

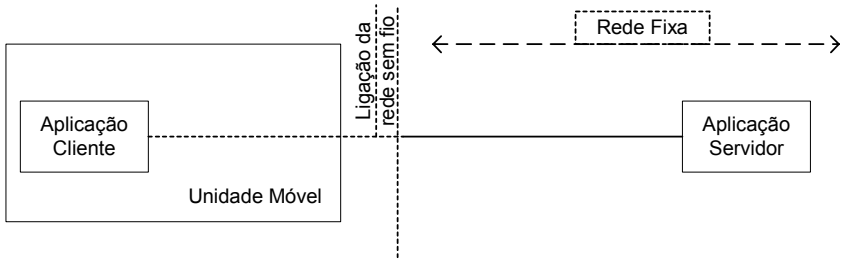

Figura A1.5 Cliente/Servidor

A uma segunda opção para este tipo de rede quanto ao mecanismo de comunicação, é a invocação da chamada remota de procedimento (RPC). Um cliente chama um procedimento que é executado em um servidor remoto. Porém RPC síncrono não é adequado para clientes móveis, pois esse trabalha com desconexão, que bloqueia a recepção do envio de dados. Existem algumas propostas sobre RPC assíncrono. O RPC assíncrono recebe a chamada feita pelo cliente e armazena em um *log* estável e imediatamente o controle é retornado à aplicação. Esse mecanismo permite que o software de suporte à comunicação e, utilize diferentes canais de comunicação para enviar requisições e receber respostas de forma transparente a aplicação (MATEUS & LOUREIRO, 1998).

Existem duas propostas de extensão do modelo cliente/servidor: Modelo Cliente/Agente/Servidor e Modelo Cliente/Interceptador/Servidor. Na seqüência são apresentadas suas características.

1.4.1.1 Modelo Cliente/Agente/Servidor

Este é um modelo computacional divido em três partes: cliente/agente/servidor., por meio das quais é utilizada a estrutura de enfileiramento de mensagens de cliente móvel para agente, e de agente para servidor. Um agente ou *proxy* se torna substituto do cliente na rede fixa. O agente está constante na rede fixa, mantendo, assim, a presença

do cliente móvel. Esta arquitetura alivia o impacto do largura de banda limitada e a baixa confiabilidade da rede. A interação ocorre entre cliente móvel e agente e agente servidor, (Figura 1.6). Os protocolos utilizados entre as partes para interação podem ser diferentes. Uma parte pode ser executada independente da outra.

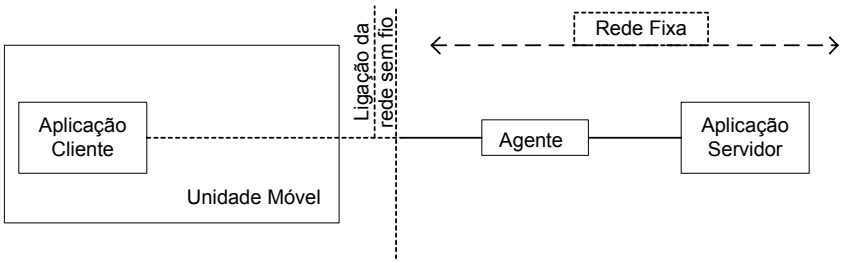

Figura A1.6 Cliente/Agente/Servidor

O agente pode ser projetado para tratar de clientes ou serviços. No caso de clientes, o agente pode representar na rede fixa uma única aplicação em diferentes computadores móveis. O agente pode ser responsável por um serviço específico como acesso a um servidor Web ou a um banco de dados. O agente de serviço pode atender mais clientes simultaneamente (PITOURA & SAMARAS, 1998) (MATEUS & LOUREIRO, 1998).

Definido o papel do agente, há possibilidade de se colocar o agente o mais próximo do cliente móvel, ou seja, na estação base onde se encontra o cliente. Conseqüentemente, se o cliente mudar de célula, o agente também deverá se mover.

No entanto, ainda existem alguns problemas que este modelo deixa em aberto. Considerando que o agente se encarrega de toda interação com o cliente, para o servidor não existem problemas de comunicação. Já o cliente precisa ser modificado para interagir com o agente e não mais com o servidor. Podemos considerar um problema que merece atenção, quando já existe uma aplicação cliente/servidor desenvolvida e se deseja disponibilizá-la num ambiente móvel. Outro problema refere-se ao agente que, estando localizado na rede fixa, só pode otimizar a transmissão de dados para o cliente móvel, a não ser que o cliente também execute uma função semelhante (PITOURA & SAMARAS, 1998), (MATEUS & LOUREIRO, 1998).

1.4.1.2 Modelo Cliente/Interceptador/Servidor

Para tentar solucionar os problemas com o modelo cliente/agente/servidor, foi dividido o agente em duas partes: uma que fica no cliente móvel e outra que permanece na rede fixa. Esses dois elementos são chamados de interceptadores. O elemento que está do lado do cliente intecerpta chamadas do cliente e, juntamente com o interceptador do servidor, executa a otimização para reduzir a transmissão de dados no canal de comunicação e tenta prover a ininterrupção da computação no cliente móvel. Ambos são vistos como *proxy*, do lado do cliente o interceptador é um proxy do servidor e do lado do servidor o interceptador é um *proxy* do cliente que reside na rede fixa (Figura 1.7).

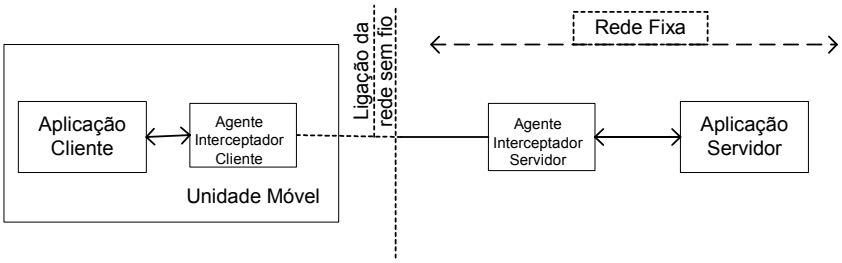

Figura A1.7 - Cliente/Interceptador/Servidor

O modelo interceptador é transparente tanto para clientes quanto para servidores. O protocolo de comunicação entre os dois agentes pode facilitar a redução de dados altamente e o protocolo de otimização não limita a funcionalidade ou interoperabilidade do cliente. A cooperação dos dois agentes permite otimização mais eficientes das ligações sem fios, trazendo benefícios para aplicações diferentes, oferecendo, assim, flexibilidade controlando desconexões. Por exemplo, um *cache* local pode ser mantido no agente do cliente móvel. O *cache* pode ser usado para satisfazer as exigências do cliente para dados durante desconexões. No *cache* podem ser enfileirados pedidos ao agente do cliente móvel a ser fornecido na reconexão. Similar pode ocorrer quanto ao agente do servidor. Também pode ser controlada a conectividade fraca.

O problema deste modelo é a necessidade de se desenvolver um par de interceptores para cada tipo de aplicação já que a funcionalidade e a otimização, em geral, são dependentes do tipo do problema.

1.4.2 Modelo *Peer-to-Peer* (Par-Par)

Na arquitetura par-par, não há distinção entre cliente e servidor. Cada estação tem a funcionalidade completa do cliente e do servidor. Mapeando esse modelo para a computação móvel, os computadores se tornam parceiros idênticos numa computação distribuída (MATEUS & LOUREIRO, 1998).

Como neste caso, um serviço pode ser disponibilizado em um computador móvel, que para economizar energia trabalha em modo *doze* (cochilo) ou pode ser desligado, o que não é conveniente, considerando a computação distribuída.

1.4.3 Modelo Agente Móvel

Agentes móveis são processos que mudam de um *site* para o outro para realizar uma tarefa especificada. Cada agente móvel possui junto, dados, instruções e estado de execução. O agente móvel executa de forma autônoma e independente da aplicação que o invocou. Quando o agente alcançar o seu destino, é autenticado, preparado para execução num ambiente de execução na entidade destino e, finalmente, executado. Para realizar sua tarefa, o agente móvel pode se transportar de uma estação para outra, pode gerar os agentes novos, ou pode interagir com outros agentes. Em conclusão, o agente móvel ao concluir sua tarefa entrega os resultados para a aplicação que o invocou ou para um servidor especificado inicialmente.

As características principais de agentes móveis são:

- habilidade de um agente móvel interagir e cooperar com outro agente;
- autonomia no sentido de que sua execução se realiza com ou sem nenhuma intervenção da entidade que disparou esse agente;
- alto grau de interoperabilidade, executada em diferentes plataformas de hardware e software;
- capacidade de responder a eventos externos;
- mobilidade do agente, habilidade de se mover entre estações.

Agentes móveis são projetados para tomar decisões e resolver problemas. Existem duas linguagens envolvidas com um agente móvel, segundo PITOURA & SAMARAS (1998). Uma é usada para expressar a tarefa do agente, geralmente uma linguagem tipo *script*. A segunda está associada a representação do conhecimento.

A segurança é um dos principais obstáculos quanto a aceitação dos agentes móveis. Neste aspecto é incluído proteção contra vírus, autenticação, privacidade e utilização dos recursos locais da estação.

Agentes móveis provêem um método assíncrono e eficiente para pesquisar por informações ou serviços em redes que se alteram com rapidez. São convenientes para tratarem de desconexões. Neste modelo, a mobilidade é implícita.

1.5 Considerações

Na computação móvel, o ambiente é dinâmico. Uma unidade móvel pode estar desconectada em uma certa estação base e em outra estação, uma outra unidade pode estar totalmente conectada. Porém, ao longo do tempo esta situação pode inverter-se, assim como a quantidade de unidades móveis e o número de recursos disponíveis pode variar de um momento para o outro em uma determinada rede.

Foram abordadas aqui as principais características de um ambiente de computação móvel e as tecnologias mais utilizadas quanto a computação sem fio. No decorrer deste apêndice também foram descritos os aspectos da computação móvel como desconexão, reintegração, *handoff*, mobilidade e as possíveis arquiteturas quanto aos modelos de computação móvel. Estes dados foram estudados com intuito de melhor visualizar as necessidades que o banco de dados em um ambiente móvel terá que suportar.