UNIVERSIDADE ESTADUAL DE LONDRINA

EDUARDO GALLIANO

BANCOS DE DADOS MÓVEIS

Londrina – 2007

EDUARDO GALLIANO

BANCOS DE DADOS MÓVEIS

Monografia apresentada ao Curso de Pós-Graduação em Redes de Computadores e Comunicação de Dados da Universidade Estadual de Londrina, como requisito parcial à obtenção de Especialização.

Orientador: Prof. Dr. Alan Salvany Felinto

Londrina – 2007

EDUARDO GALLIANO

BANCOS DE DADOS MÓVEIS

Monografia apresentada ao Curso de Pós-Graduação em Redes de Computadores e Comunicação de Dados da Universidade Estadual de Londrina, como requisito parcial à obtenção de Especialização.

COMISSÃO EXAMINADORA

_____

Prof. Dr. Alan Salvany Felinto
Universidade Estadual de Londrina

_____

Prof. Dra. Linnyer Beatrys Ruiz
Universidade Estadual de Londrina

_____

Prof. Msc. Elieser Botelho Manhas Jr.
Universidade Estadual de Londrina

Londrina, ____de_____ de 2007.

AGRADECIMENTOS

A Deus, acima de tudo.

Ao meu orientador e amigo Prof Dr. Alan Salvany Felinto.

A Soraya Mayumi Tasca, por toda a paciência e carinho.

A meus pais pela minha educação.

"Os verdadeiros analfabetos são aqueles que aprenderam a ler e não lêem".

Mário Quintana

GALLIANO, Eduardo. **Bancos de dados móveis.** 2007. Monografia (Especialização em Redes de Computadores e Comunicação de Dados) - Universidade Estadual de Londrina.

RESUMO

O presente trabalho procura relatar as características de sistemas de banco de dados móveis, demonstrar as influências da computação móvel para esta tecnologia e apresentar um estudo comparativo a sistemas de bancos de dados convencionais. Visando isso, este trabalho traz uma breve explanação sobre a computação móvel e os fundamentos de sistemas de gerenciamento de banco de dados.
O trabalho aborda os principais conceitos de bancos de dados e busca demonstrar como as características da computação móvel podem afetar no tratamento de bancos de dados.
O estudo ainda apresenta pontos importantes sobre segurança em bancos de dados móveis e seus desafios.
Como conclusão, os principais bancos de dados móveis presentes no mercado atual são avaliados e relacionados às atuais pesquisas sobre o assunto.

Palavras-chave: Bancos de Dados Móveis; Computação Móvel; Banco de Dados.

GALLIANO, Eduardo. **Mobile Database Systems**. 2007. Monografia (Especialização em Redes de Computadores e Comunicação de Dados) - Universidade Estadual de Londrina.

ABSTRACT

The present work describe the characteristics of mobile database systems, demonstrating the influences of the mobile computation for this technology and to present a comparative study to systems of conventional databases. Aiming at that this work brings a short explanation on the mobile computation and the bases of systems of managing database.
The work boards the principal concepts of databases and tries to demonstrate how the characteristics of the mobile computation can affect on the treatment of databases.
The study also presents important points on security in mobile databases and its challenges.
As a conclusion, the essencials mobile databases presented nowadays are valued and related to the current researches on the subject.

Key-words: Mobile database systems; Mobile computation, Database.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

PDA	Personal Digital Assistant
SGBD	Sistema de Gerenciamento de Bancos de Dados Móveis
PCS	Personal Communication System
GSM	Global System Mobile Communication
MSC	Mobile Switching Center
PSTN	Public Switched Telephone Network
AFIM	After Image
BFIM	Before Image
CPU	Central Processing Unit
HiCoMo	High Commit Mobile
SBDM	Sistema de Banco de Dados Móvel
WAL	Write Ahead Logging
VLR	Visitor Location Register
HLR	Home Location Register
TCOT	Transaction Commit On Timeout
EFT	Expected Failure Time
ELUT	Estimated Log Unification Time

SUMÁRIO

INTRODUÇÃO

A computação móvel tem se apresentado como um novo paradigma em comunicação e processamento de dados e vem atingido em cheio conceitos e modelos tradicionais da ciência da computação. Tudo isso se deve principalmente ao crescente uso de dispositivos móveis e portáteis e ao desenvolvimento da comunicação digital sem fio.

A computação móvel permite ao usuário estabelecer comunicação com outros usuários e gerenciar seu trabalho enquanto se movimenta. São características importantes para organizações geograficamente dispersas. Podemos tomar como exemplo disso o policiamento de trânsito, serviços de previsão do tempo, serviços de táxi, comunicações sobre o mercado financeiro e aplicações sobre agências de informação (ELMASRI; NAVATHE, 2002).

O ambiente criado pela computação móvel tem se mostrado útil em muitas aplicações, incluindo aí também os sistemas de gerenciamento de bancos de dados. Os usuários em viagem usam *notebooks* para trabalhar em trânsito, algumas empresas já utilizam dispositivos móveis para controlar o serviço de entrega de mercadorias, serviços de emergências médicas podem usar os recursos do ambiente móvel para acesso e fornecimento de dados sobre a situação dos atendimentos no local da emergência atendida.

As novidades em tecnologia da comunicação sem fio (*wireless*) transformam cada vez mais os serviços de informações móveis em uma realidade. Cresce então o número de novos sistemas móveis que necessitam de acessos a bancos de dados utilizando equipamentos do tipo *Personal Digital Assistant*, computadores portáteis ou telefones celulares (CÔRTES; LIFSCHITZ, 2003). Todo

esse novo cenário cria uma nova situação onde máquinas não apresentam mais uma localização fixa. Há determinadas aplicações onde os processamentos de consultas podem ser dependentes da localização do usuário, sua direção e velocidade (KORTH; SILBERSCHATZ, 1999).

Surgem então novos desafios relacionados a hardware e software. Em se tratando de bancos de dados, os casos podem envolver a gerência de dados e transações e a recuperação do banco de dados (ELMASRI; NAVATHE, 2002).

É importante ressaltar também que a computação móvel pode ser vista como uma variação da computação distribuída e as questões relativas à gerência de dados distribuídos também podem ser aplicadas aos bancos de dados móveis. São os casos de distribuição e replicação de dados, modelos de transações, processamento de consultas, recuperação e tolerância à falhas e projeto de bancos de dados, casos estes que passam a ser avaliados com o intuito de atender os requisitos do ambiente da computação móvel.

O presente trabalho tem por objetivo avaliar os principais conceitos sobre os bancos de dados móveis, relatar os novos desafios criados pela computação móvel a esse padrão de aplicação e apresentar os principais pontos de segurança sobre o assunto.

O primeiro capítulo procura apresentar conceitos e importantes observações sobre a computação móvel. São relacionados os principais dispositivos móveis e apresentados importantes conceitos como portabilidade, mobilidade e *handoff*. O capítulo ainda oferece um estudo sobre a arquitetura da computação móvel.

O segundo capítulo apresenta os fundamentos sobre os Sistemas de Gerenciamento de Banco de Dados. Neste capítulo são apresentados os principais

conceitos sobre o assunto como transação, redundância, controle de concorrência, restrições de integridade e recuperação de bancos de dados. Esse capítulo deve servir como base para o estudo em aplicações de bancos de dados móveis do próximo capítulo.

O terceiro capítulo analisa os bancos de dados móveis e seus principais desafios. Este capítulo traz um estudo sobre a influência da mobilidade em gerenciamento de dados, o gerenciamento de transação e a recuperação em bancos de dados móveis, desconectividade e consistência de dados.

O quarto capítulo traz informações sobre as pesquisas feitas em segurança de bancos de dados móveis.

O quinto capítulo conclui o estudo, avalia os bancos de dados para ambientes móveis do mercado e aponta o crescente aparecimento de pesquisas sobre o assunto.

1 COMPUTAÇÃO MÓVEL

A computação móvel é a conjunção da computação tradicional com a tecnologia da comunicação sem fio e pode ser caracterizada pela utilização de dispositivos móveis e portáteis. Neste ambiente as informações dependem da sua localidade, surgindo então novos desafios. Por isso, questões como o gerenciamento de dados devem ser revistas (AMADO, 2002).

A mobilidade se mostra como um novo paradigma, gerando mudanças em projetos e formas de desenvolvimento de sistemas. Isso significa dizer que os ambientes de computação móvel fizeram surgir a necessidade do desenvolvimento de novas formas de adaptação. Aplicações passam a ser projetadas considerando a mobilidade do usuário, limitação de energia do dispositivo e variações de latência e largura de banda de redes sem fio (ENDLER; SILVA, 2000).

A computação móvel traz também a possibilidade de expectativa da computação pervasiva.

1.1 Dispositivos móveis

Os dispositivos móveis são equipamentos portáteis que comumente utilizam pequenos periféricos de entrada e saída. São considerados dispositivos móveis o *Personal Digital Assistants* (PDA) ou assistente digital pessoal, o *smart phone*, o telefone móvel, o *notebook*.

O desenvolvimento tecnológico destes dispositivos trouxe significativas melhorias a importantes fatores como autonomia de bateria,

visualização, poder de processamento e compatibilidade com dados multimídia, tudo isso sem interferir na portabilidade dos mesmos (AMADO, 2002).

Em se tratando de restrições dos dispositivos móveis, se considerarmos a velocidade de processamento dos dispositivos móveis, quanto maior a latência que pode ser tolerada no processamento menor será seu consumo de energia. Assim, velocidade de processamento, custo de armazenamento em termos de energia, quantidade de dados transmitidos e recebidos e a latência tolerada são fatores a serem considerados nos vários aspectos do acesso e organização dos dados para comunicação (PIRES; REDIN; *BELUSSO*; LIMA; AUGUSTIN, 2005.).

1.2 Portabilidade

Um equipamento caracteriza-se por portável quando apresenta por completo as ferramentas necessárias ao usuário, tudo isso de forma compacta para que possa ser facilmente movido de um lugar para o outro.

Para ser portável, o dispositivo precisa ser pequeno, leve, durável, operacional em diversas condições e consumir o mínimo de energia possível assegurando longa autonomia de uso para a bateria (AMADO, 2002).

Os computadores tradicionais não foram projetados para serem carregados de um lugar para outro. Por isso, quando se trata de portabilidade existem algumas concessões ao usuário, dentre elas restrições de memória, capacidade de armazenamento, consumo de energia e interface do usuário (YAMIN; AUGUSTIN; BARBOSA; SILVA; GEYER, 2001).

A bateria é o componente mais pesado de um PDA e gera

problemas de peso e tamanho, além da necessidade de recarga. Para aumentar a portabilidade e a vida útil da bateria são utilizadas formas de minimização de consumo de energia.

O usuário de PDA usa botões e canetas como periféricos de entrada e uma tela muito pequena para saída. Com isso, muitas aplicações precisam ser adaptadas às restrições de interface.

A capacidade de armazenamento de um dispositivo portátil é restrita pelo seu tamanho físico e consumo de energia, muitos destes dispositivos nem possuem tal recurso. Algumas estratégias para solução desse tipo de problema são estudadas: compressão automática de dados, acesso a arquivos armazenados em bases remotas, compressão de páginas de memória virtual, compartilhamento de bibliotecas de código.

Dispositivos portáteis também são menos seguros e confiáveis, isso porque sua portabilidade aumenta o risco de danos físicos, de acesso não autorizado e roubo ou perda dos dispositivos móveis. Por isso o usuário deve ter uma preocupação ainda maior com rotinas de backup e armazenamento de dados em bases remotas.

1.3 Mobilidade

Mobilidade é propriedade do que é móvel ou do que obedece às leis do movimento. Em computação móvel, a mobilidade é o acesso à informação de qualquer lugar e a qualquer momento.

A mobilidade leva em conta as necessidades dos usuários nômades, usuários conectados à rede de localizações arbitrárias e que não ficam permanentemente conectados (YAMIN; AUGUSTIN; BARBOSA; SILVA; GEYER,

2001).

Mobilidade exige adaptação. Adaptação é a condição de produzir resultados em condições adversas. Assim, deve existir um equilíbrio entre a necessidade e a disponibilidade de recursos computacionais para a execução de aplicações.

A mobilidade faz aumentar a volatilidade da informação. Dados estáticos em computadores fixos podem se transformar em dados dinâmicos nos dispositivos móveis. A mobilidade levanta questões importantes que aqui devem ser citadas: em um computador móvel seu endereço de rede muda dinamicamente, sua localização passa a influir em parâmetros de configuração e respostas a consultas (AMADO, 2002).

A mobilidade cria a necessidade de gerência de localização e o custo de localizar o cliente passa a ser relevante.

1.3.1 Tipos de Mobilidade

Segundo Kumar (2006), a mobilidade pode ser definida como mobilidade terminal ou mobilidade pessoal. Não há dependência de relação entre os dois tipos. Em mobilidade pessoal a parte é livre para se mover e em mobilidade terminal a unidade de comunicação se utiliza dessa liberdade.

A comunicação de voz ou dados pode ser suportada por outros tipos de mobilidade, mas para visualizar um banco de dados móvel completo ambos os tipos de mobilidade são essenciais.

1.3.1.1 Mobilidade Terminal

Mobilidade terminal permite a uma unidade móvel (*notebook*, telefone celular, PDA) acessar serviços de qualquer lugar enquanto em movimento ou estacionado, independente de quem é o transportador da unidade.

Como exemplo, podemos dizer que um celular pode ser usado pelo seu proprietário e ele também pode ser emprestado a alguém para seu uso. Em mobilidade terminal é responsabilidade da rede *wireless* identificar o dispositivo de comunicação.

Do ponto de vista de telecomunicação, o ponto de conexão de rede (acesso de rede ou ponto de terminação) é identificado, mas não como parte chamada. Assim, a conexão é estabelecida entre dois pontos e entre duas pessoas que se chamam entre si. Este tipo de conexão em uma sessão permite o uso de dispositivos de comunicação para serem compartilhados entre os indivíduos.

Assim a mobilidade de terminal é suportada, isto significa que o mesmo terminal pode ser usado para conectar outra parte de qualquer lugar e por qualquer usuário.

1.3.1.2 Mobilidade Pessoal

Em mobilidade pessoal um usuário não precisa carregar qualquer equipamento de comunicação consigo. O usuário pode se valer de qualquer dispositivo para estabelecer comunicação com outra parte. Esta facilidade requer um esquema de identificação para verificar qual é a pessoa que deseja se

comunicar.

A mobilidade pessoal está disponível na web, um usuário pode logar na web de máquinas localizadas em diferentes lugares e acessar seu e-mail, por exemplo. O sistema móvel amplia esta facilidade para que o usuário possa usar qualquer dispositivo móvel para acessar a internet.

Em mobilidade pessoal cada pessoa tem de ser identificada unicamente. Um modo de fazer isso é através de um número único de identificação.

1.4 Handoff

Handoff é o processo de transferência de uma estação móvel de uma estação rádio-base para outra ou de um canal para outro. A mudança de canal devido ao *handoff* pode ser através de um *slot* de tempo, de uma banda de freqüência, de uma palavra código, ou de uma combinação deles, dependendo da técnica de múltiplo acesso utilizada (BOROS, 2007).

Através desse processo usuários da rede conseguem manter seus dispositivos móveis conectados ao sair de uma célula para outra vizinha. Assim, graças ao *handoff* é permitido ao usuário descartar o contato com uma estação base numa célula e estabelecer contato com outra sem haver desconexão. A troca deve ocorrer quando o *host* móvel está na região onde as células se sobrepõem (AMADO, 2002).

1.5 Arquitetura da computação móvel

Segundo Elmasri e Navathe (2002), a arquitetura da computação móvel é composta por entidades ligadas por uma rede fixa e interconectada por fios

e por entidades móveis que se comunicam com esta rede. A arquitetura geral pode ser vista como uma arquitetura distribuída, composta de *hosts* fixos, estações bases e unidades móveis.

Hosts fixos são conectados por uma rede de alta velocidade ligada por fios. O *host* fixo é um computador com objetivos genéricos e não é equipado para administrar unidades móveis, podendo ser configurado para tal gerenciamento.

Unidades móveis são equipamentos portáteis alimentados por bateria e que se movem em um domínio geográfico de mobilidade. Essa área é restrita pela limitada amplitude de banda de canais de comunicação sem fio. O gerenciamento dessa mobilidade é feito pela divisão do domínio geográfico em domínios menores (células).

A estação base é equipada com interface de comunicação para acesso de dados pelas unidades móveis que se encontram em sua célula, ou seja, que estão na área de cobertura dentro da qual uma estação base consegue captar seu sinal.

Unidades móveis e estações base comunicam-se por canais sem fio que apresentam amplitudes de banda consideravelmente menores do que uma rede cabeada.

Quando uma unidade móvel sai de uma célula e entra em outra passa a ser controlada por uma nova estação base. Como a comunicação entre eles se dá sem fio, a taxa de transferência de dados sofre grande variação.

1.5.1 Rede ad-hoc móvel

A rede *ad-hoc* tem como principal característica ausência de infra-estrutura. Os nós que integram essa rede comunicam-se entre si e funcionam como

roteadores. A mobilidade dos nós de uma rede *ad-hoc* faz com que esse tipo de estrutura apresente freqüentes mudanças de topologia de rede, ou seja, sua topologia é dinâmica. Os nós móveis também indicam que os equipamentos possuem restrições quanto ao consumo de energia para o funcionamento. Isto faz com que seja necessário o controle de consumo de energia usado para o processamento para que rotas entre nós sejam descobertas.

Nós que se conectam um ao outro sem a necessidade de outros intermediários são chamados de nós vizinhos. Cada um dos nós de uma rede *ad-hoc* pode realizar o roteamento de pacotes. Assim, se um nó precisa transmitir dados para outro nó que não seja seu vizinho será preciso o uso de uma rota para que a comunicação fim a fim seja feita. Entram em cena os protocolos de roteamento e se faz necessário um algoritmo de roteamento em cada um dos nós da rede ad-hoc (AMODEI JUNIOR; DUARTE, 2003).

2 FUNDAMENTOS DE BANCOS DE DADOS

Este capítulo apresenta os principais fundamentos sobre sistemas de banco de dados, segundo Korth e Silberschatz (1999). Os conceitos apresentados são importantes para uma compreensão de sistemas de bancos de dados convencionais, conceitos estes que também são vistos nesse trabalho em ambientes de bancos de dados móveis. Por conta dos fatores dos ambientes móveis cada fundamento apresentado nesse capítulo pode apresentar alteração de tratamento em bancos de dados móveis.

2.1 Sistema de Gerenciamento de Bancos de Dados

Um sistema de gerenciamento de banco de dados (SGBD) é formado por uma coletânea de dados que se relacionam e um conjunto de programas que oferecem acesso a tais dados. O objetivo principal de um SGBD é prover um ambiente apropriado e eficiente para uso no armazenamento e acesso à informação. Este tipo de sistema é projetado para o tratamento de grandes volumes de informação. Este tratamento ou gerenciamento de dados envolve a definição de estruturas para o armazenamento e mecanismos para manipulação de informações. Um sistema de banco de dados deve também garantir a segurança da informação armazenada e evitar possíveis resultados anômalos em caso de compartilhamento de dados.

SGBDs apresentam ao usuário uma visão abstrata dos dados. Para isso são definidos três níveis de abstração de dados: o nível físico, o nível conceitual e o nível das visões do usuário.

Os bancos de dados mudam com a inserção e exclusão de dados. O conjunto de informações armazenadas em um dado momento é chamado de instância de banco de dados. Seu projeto geral é chamado de esquema de banco de dados. O esquema de um banco de dados é definido por um conjunto de definições expressas em uma linguagem de definição de dados. O resultado da compilação desta linguagem é armazenado em um dicionário de dados. Um dicionário de dados é um repositório de metadados, que podem ser definidos como dados sobre os dados.

O conceito de modelo de dados é essencial à estrutura de um banco de dados. Trata-se de um conjunto de ferramentas conceituais para descrição, relacionamento e semântica de dados e restrições de consistência.

O gerenciador de banco de dados é um importante módulo do sistema que oferece interface entre os dados de baixo nível armazenados e os programas de aplicação com suas requisições. Dentre as principais tarefas do gerenciados de banco de dados estão: o cumprimento de integridade, o cumprimento de segurança, recuperação e controle de concorrência.

2.2 Transação

Uma transação pode ser definida como uma unidade de programa que faz o acesso ou atualização de vários itens de dados. Cada item de dado deve ser precisamente lido uma vez pela transação e gravado no máximo uma vez se houver atualização do item. É importante e fundamental que as transações não violem qualquer restrição de consistência do banco de dados. Assim, se o banco de dados era consistente quando uma transação iniciou deve ser também consistente no término da mesma.

Cabe ressaltar que durante a execução de uma transação pode ser necessário permitir temporariamente a inconsistência. Essa inconsistência pode trazer dificuldades em caso de falhas.

Dois importantes requerimentos são usados em transações: corretude ou precisão e atomicidade. Em corretude, cada transação deve ser um programa que preserva a consistência do banco de dados. Em atomicidade, todas as operações associadas a uma transação precisam ser executadas até o final ou nenhuma deve ser executada. Assegurar a atomicidade é responsabilidade do próprio sistema de banco de dados.

Por conta de falhas, uma transação nem sempre pode completar sua execução. Para assegurar a atomicidade uma transação desse tipo não pode afetar o estado consistente do banco de dados. Caso o estado seja afetado, o banco de dados precisa ser restaurado ao estado em que a transação o encontrou. Parte do esquema de recuperação é gerenciar estes casos de transações mal sucedidas. Por conta de todos esses fatores pode-se afirmar que uma transação pode estar em um dos seguintes estados: ativo, parcialmente compromissado, falhado, abortado e compromissado ou confirmado. O estado ativo pode ser considerado como o estado inicial da transação. Uma transação é dita parcialmente compromissada logo após a última instrução executada. Uma transação é dita falhada quando sua execução normal não pode prosseguir. Uma transação é abortada quando é desfeita e o banco de dados é restaurado ao seu estado anterior ao início da transação. Uma transação completada com sucesso é chamada de transação compromissada ou confirmada.

Uma transação se inicia no estado ativo. Ao alcançar sua última instrução ela entra em estado parcialmente compromissado. Isto significa que a

transação completou sua execução, mas ainda pode ser abortada já que seu resultado ainda não foi gravado em disco. Uma falha de hardware, por exemplo, pode impedir o término da transação. Então uma transação confirmada será sempre capaz de completar sua gravação no disco.

Uma transação é dita falhada quando não pode prosseguir com sua execução, talvez por problemas hardware ou software. A transação precisa ser desfeita e, nesse caso, entra no estado abortado. Nesse caso, um sistema de banco de dados pode seguir dois caminhos: reiniciar ou encerrar a transação.

2.3 Controle de Redundância

A redundância é caracterizada pela presença de um elemento de informação duplicado. Sistemas de banco de dados devem ter capacidade de garantir que os dados não sejam redundantes. Esse controle é usualmente conhecido como integridade referencial. O controle de redundância não permite incluir dois registros com a mesma chave primária e excluir um registro que possua relacionamento com outras tabelas (chave estrangeira). Com isto, o controle de redundância evita a inconsistência de dados. Este padrão de integridade é o fundamento do modelo relacional, por isso é necessário que o banco de dados tenha a capacidade de gerenciar o controle de redundância.

2.4 Controle de Concorrência

Controle de concorrência é um esquema usado para garantir que as transações são executadas de forma segura. Uma das qualidades dos sistemas desenvolvidos é a multiprogramação que permite a execução de diversas

transações visando o compartilhamento do processador. Nesse ambiente multiprogramado diversas transações podem executar concorrentemente. Por isso os sistemas precisam controlar a interação entre transações concorrentes com o objetivo de prevenir a violação da consistência do banco de dados. Esse controle é feito por um conjunto de mecanismos definidos como esquemas de controle de concorrência.

Quando as transações são executadas concorrentemente a consistência do banco pode ser violada mesmo que cada transação individual esteja correta. Cabe ressaltar que nesse ambiente é importante o conceito da seriabilidade. È necessário que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente seja computacionalmente equivalente a um escalonamento produzindo executando essas transações serialmente em alguma ordem. Diz-se que um sistema que garante esta propriedade assegura a seriabilidade.

Os escalonamentos podem ser seriais e não-seriais. Escalonamentos seriais consistem de uma seqüência de instruções de várias transações onde as instruções pertencentes a uma única transação aparecem juntas. No escalonamento não serial as operações de uma transação são executadas intercaladas com operações de outra transação.

Os mecanismos de controle de concorrência são classificados em mecanismos otimistas e pessimistas. Os mecanismos de controle de concorrência pessimistas são aqueles que buscam impedir antecipadamente os tipos de acesso concorrente a dados que podem gerar inconsistências. Isso pode ser feito bloqueando temporariamente o acesso de dados a algumas aplicações enquanto outra está acessando. Os mecanismos de controle de concorrência otimistas, ao

invés de tentar evitar antecipadamente acessos inconsistentes permitem o livre acesso. No final da execução das aplicações é iniciado um processo que examina a incidência de inconsistência nos dados graças ao acesso concorrente.

2.5 Restrições de Integridade

As restrições de integridade oferecem meios de assegurar que mudanças feitas no banco de dados por usuários autorizados não resultem em perda de consistência dos dados. As restrições de integridade podem resguardar o banco de dados contra danos acidentais. Uma restrição de integridade pode ser um predicado arbitrário que reaplica ao banco de dados. No entanto, os predicados arbitrários podem ser custosos para testes. Dessa forma é preciso limitar-se em restrições de integridade que possam ser testadas com um mínimo custo adicional.

A forma mais elementar de restrição de integridade é a restrição de domínio. Em um sistema é possível que diversos atributos tenham um mesmo domínio. A definição de restrição de domínio não somente permite testar os valores inseridos no banco de dados como possibilita testar as consultas assegurando que as comparações façam sentido.

A integridade referencial é usada para garantir que um valor que aparece em uma relação para um dado conjunto de atributos também apareça para um certo conjunto de atributos em outra relação.

No modelo E-R (Entidade – Relacionamento), o esquema do banco de dados relacional derivado de diagramas E-R resulta em um conjunto de relacionamentos que possui regras de integridade referencial. Outro ponto de origem de restrições de integridade referencial são os conjuntos de entidades fracas. Isto porque o esquema relacional para cada conjunto de entidades fracas

inclui uma chave estrangeira que leva a uma restrição de integridade referencial.

As modificações em banco de dados podem violar as regras de integridade referencial.

2.6 Recuperação

Todo e qualquer sistema está sujeito à falhas. É responsabilidade do gerenciador do banco de dados detectar as falhas e restabelecer o banco de dados ao estado anterior à ocorrência da falha. Para isso, são usados procedimentos de recuperação.

Usualmente diversas operações do banco de dados formam uma única unidade de trabalho. Um exemplo que pode ser citado é a transferência de valores entre contas bancárias onde uma conta é debitada e outra creditada. É fundamental que ocorram os dois casos ou nenhum deles, a transferência deve acontecer por inteiro. Esta necessidade é chamada de atomicidade.

As falhas mais simples de se tratar são aquelas que não ocasionam perda de informação no sistema. As falhas mais difíceis são as que ocasionam perda de informação ao sistema. Para definir a recuperação é preciso identificar os modos de falhas e, assim, identificar como esses modos afetam o conteúdo do banco de dados.

Os esquemas de recuperação normalmente são compostos de duas partes: ações tomadas durante o processamento normal da transação que asseguram a existência de informações usadas para a recuperação e ações tomadas após as falhas garantindo a consistência do banco de dados e a atomicidade da transação.

Podem ser considerados os seguintes tipos de falhas: erros lógicos,

erros de sistema, queda de sistema e falha de disco.

Em erros lógicos, a transação não pode prosseguir com a execução normal devido a alguma condição interna (entrada com erro, dado não encontrado, *ovewrflow*). O erro de sistema significa que o sistema entrou em algum estado indesejável e a transação não pode prosseguir sua execução normal, mas pode ser executada depois. A queda do sistema pode ser uma falha de hardware que causa perda de conteúdo do armazenamento volátil. A falha no disco pode ocorrer durante uma transferência de dados para este tipo de dispositivo.

Cabe ressaltar a importância do log em recuperação do banco de dados. Cada registro de log descreve uma gravação no banco de dados e pode guardar as seguintes informações: nome da transação, nome do item de dado, valor antigo e novo valor. Outros registros de log podem ainda gravar eventos de processamento de transações como o início, o compromisso ou a abortagem de uma transação. É importante que os registros de log sejam criados antes que o banco de dados seja modificado. É importante destacar também que o log deve ser guardado em algum dispositivo estável de armazenamento.

3 BANCOS DE DADOS MÓVEIS

Este capítulo apresenta os principais fundamentos dos bancos de dados móveis segundo Kumar (2006). A integração de dispositivos portáteis com as mais contemporâneas tecnologias de comunicação celular, redes de comunicação sem fio e serviços via satélite vem permitindo que usuários de dispositivos móveis mantenham a conexão com a rede enquanto se movimentam livremente, tendo acesso a recursos, serviços e informações compartilhadas. Este conceito é chamado de computação móvel e este ambiente dispõe aos usuários o acesso a informações e recursos compartilhados independente de onde estejam localizados e de sua mudança de localização (mobilidade). A computação móvel permite o desenvolvimento de novas aplicações em banco de dados. Para a efetiva execução dessas aplicações tornam-se necessárias mudanças no gerenciamento e nos mecanismos de garantia de consistência dos dados. Essa necessidade vem das restrições impostas pelos ambientes de comunicação sem fio: a limitação da largura de banda dos canais de comunicação sem fio, a mobilidade e as freqüentes desconexões dos dispositivos móveis, a mobilidade dos dados e o grande número de usuários.

Bancos de dados móveis geralmente funcionam em redes “sem fio” (*wireless*). Os bancos de dados móveis podem ser distribuídos em dois possíveis cenários. No primeiro cenário, toda a base de dados está distribuída principalmente entre os componentes ligados por fiação, possivelmente com replicação total ou parcial dos dados. Uma estação base gerencia sua própria base de dados com um SGBD com funcionalidades adicionais para localizar unidades móveis e para

gerenciar consultas e transações do ambiente móvel. No segundo cenário a base de dados é distribuída pelos componentes com e sem fio. A responsabilidade do gerenciamento dos dados é compartilhada entre as unidades móveis e as estações base.

Por conta da influência de todas as características, tanto da computação móvel quanto de redes wireless, o gerenciamento de dados e o controle de concorrência em ambientes de computação móvel tem sido objeto de pesquisas. Novos modelos computacionais, arquiteturas, modelos transacionais, protocolos e algoritmos têm sido propostos. Tudo isso vem trazendo um novo universo relacionado aos sistemas de bancos de dados móveis.

3.1 Mobilidade em gerenciamento de dados

O conceito de conectividade contínua em espaço móvel permite a execução ativa necessária de tarefas independente do estado da unidade (móvel ou estática). A mobilidade pessoal e terminal têm sido componentes indispensáveis, e assim tem sido comum a abordagem onde qualquer lugar em qualquer situação passa a ser um escritório. Esse tipo de ambiente de trabalho prove um grande crescimento de produtividade.

Os recentes avanços em tecnologias *wireless* tem tornado possível o alcance da conectividade contínua em espaços móveis. As redes *wireless* passaram a ser usadas com mais freqüência em ambientes de negócios para a conexão de *notebooks*, *laptops* e outros dispositivos portáteis. Cabe ressaltar também o crescimento da funcionalidade, poder, capacidade de armazenamento de tais dispositivos. Por conta disso, os usuários passam a usá-los não só como dispositivos de comunicação, mas também para gerenciar suas atividades e

informações.

3.1.1 Efeito da mobilidade em gerenciamento de dados

Os sistemas convencionais de bancos de dados possuem uma característica em comum: todos os componentes, especialmente as unidades de processamento, são estacionários. Em sistemas distribuídos, dependendo dos tipos de dados, a distribuição de dados pode migrar de um nó para outro, mas esta migração é determinística. Isso significa que os dados se movem de uma origem fixa para um destino fixo. Tal migração de dados não satisfaz quaisquer critérios de mobilidade.

A integração da mobilidade geográfica é um excelente modo de usar o tempo em viagens, por exemplo. Entretanto, há um crescimento no número de problemas relacionados à manutenção das propriedades ACID (Atomicidade – Consistência – Isolamento – Durabilidade) na presença das mobilidades pessoal e terminal. As propriedades ACID de uma transação devem ser mantidas em todas as atividades de gerenciamento de dados. Os mecanismos de controle de concorrência e os esquemas de recuperação de banco de dados asseguram isso. Em plataformas *wireless* e móveis a natureza do processamento de dados permanece a mesma, mas as situações sob as quais os dados são processados podem mudar. Por isso é importante entender o efeito da mobilidade na distribuição de dados e nas propriedades ACID das transações.

3.1.1.1 Categorização de dados

A distribuição de dados em sistemas convencionais de bancos de

dados distribuídos pode ser feita de três modos: particionada, replicação parcial e replicação completa. A presença da mobilidade adiciona outra dimensão à convencional distribuição, introduzindo o conceito de dados dependentes de localização.

Os dados dependentes de localização são classificados como dados onde seus valores são fortemente ligados à localização geográfica específica. Há o mapeamento 1:1 entre o conjunto de valores de dados e a região que ele serve.

A requisição dependente de localização é um tipo de requisição que recebe resultados ou valores que estão sujeitos à localização da unidade móvel que a iniciou. Se as coordenadas da localização não são conhecidas a requisição não pode ser processada. Uma requisição idêntica pode obter respostas diferentes, porém corretas já que a localização de origem da requisição pode mudar constantemente.

No processamento de uma requisição dependente de localização, os dados dependentes de localização e a localização geográfica da origem da requisição devem ser conhecidos. O sistema deve mapear a localização com dados para obter os corretos dados dependentes de localização. Um conjunto de provedores de serviços tem facilidade em descobrir a localização que pode ser usada para acessar os dados dependentes de localização.

A requisição ciente de localização é um tipo de requisição que inclui referência a uma localização particular por nome ou coordenadas geográficas. Por exemplo, a requisição que busca a informação da distância entre as cidades de Londrina e São Paulo. Isto porque é uma requisição que se refere diretamente a localizações pelo nome.

3.1.1.2 Distribuição de dados dependentes da localização

O mapeamento 1:1 entre um dado e sua localização geográfica restringe as três abordagens de distribuição de dados. A fragmentação horizontal e a fragmentação vertical de um banco de dados relacional devem incluir a informação de localização implicitamente ou explicitamente. A partição do banco de dados, entretanto, torna-se mais fácil por que a decisão é exclusivamente baseada no parâmetro de localização. O conceito de região de dados é importante para entender a distribuição de partições de bancos de dados em bancos de dados móveis.

Uma região de dados é uma região geográfica ou uma célula geográfica. Cada ponto geográfico desta região satisfaz o mapeamento 1:1 com o dado. Como exemplo, uma cidade pode ser uma região de dados. Esta região de dados inteira pode estar inclusa em uma célula. O banco de dados inteiro da cidade está particionado em subdivisões. Se cada subdivisão mantém seu próprio banco de dados então em cada subdivisão uma partição de banco de dados pode ser armazenada. Uma unidade móvel que se move entre as subdivisões verá o mesmo valor consistente de um item de dados.

O exemplo de uma cadeia de hotéis pode ser usado para demonstrar o problema de replicação de dados e sua consistência para bancos de dados móveis. Um hotel em particular tem um número de filiais distribuídas pelo país. Cada filial oferece serviços idênticos; entretanto, sua locação de quartos, política, facilidades, etc, dependem da localização da filial. Assim, o mesmo tamanho de suíte pode custar mais em São Paulo ou em Londrina. As restrições de consistência de dados em São Paulo devem ser diferentes daquelas em Londrina

por conta das taxas locais e políticas do ambiente. Cada filial deve compartilhar o mesmo esquema, mas suas instanciações (valores para os dados) devem diferir.

Em uma abordagem de replicação parcial a mesma partição pode ser replicada em mais de uma subdivisão. Por exemplo, na subdivisão 1 e na subdivisão 2, as partições 1 e 2 podem ser replicadas sem afetar a consistência. Em uma replicação total, também o banco de dados inteiro pode ser replicado e usado em todas as subdivisões em um modo consistente.

A situação não muda se a região de dados é coberta por múltiplas células. Uma unidade móvel pode ser mover de uma subdivisão para outra e usar o mesmo item de dados em ambas subdivisões. Entretanto, a situação muda quando uma célula cobre duas ou mais regiões de dados. Os dados de uma região não podem ser replicados para outra região. Por exemplo, a porcentagem das taxas de mercado de uma região 1 não podem ser replicadas para uma região 2. Esta restrição exige que uma requisição dependente de localização na região 2 deva ser processada na região 2 antes do cliente entrar na região 1. Esta restrição também afeta o *cache* de dados móveis. Uma unidade móvel deve limpar seu *cache* antes de entrar em outra região de dados para manter consistência global.

Desde que a distribuição de dados dependentes de localização é dependente em localizações geográficas, sua distribuição é definida como distribuição espacial para distinguí-la da distribuição convencional que é chamada de distribuição temporal. Em distribuição espacial e em distribuição temporal a replicação espacial e a replicação temporal são usadas, respectivamente.

Replicação espacial se refere a cópias de objetos de dados que devem ter diferentes valos de dados corretos em qualquer ponto no tempo. Cada valor é correto dentro de uma área de localização. Uma destas cópias é chamada

de uma réplica espacial. Replicação temporal se refere a cópias de todos objetos de dados que tem somente um valor de dados consistente em qualquer ponto no tempo. Uma destas cópias é chamada de uma réplica temporal.

A distribuição temporal considera principalmente a disponibilidade local de dados e o custo da comunicação, mas para distribuição espacial a localização geográfica deve também ser incluída. A identificação de dados como espaciais e temporais afetam a definição de consistência.

A propriedade da atomicidade garante que os resultados parciais de uma transação não existam no banco de dados. Se uma transação falha ao confirmar, então todos seus efeitos são removidos do banco de dados. A mobilidade não altera a definição de atomicidade, mas torna sua aplicação mais difícil. O *log* de execução de transação é requerido para implementar atomicidade. Em um sistema convencional o *log* é guardado em um servidor e é facilmente disponível. Em um sistema móvel, a abordagem de *log* convencional não funciona satisfatoriamente porque uma unidade móvel se mantém conectada ou desconectada a diversos servidores enquanto ela é móvel. Há um número de modos para gerenciar um log de transação em sistemas móveis.

Em um ambiente centralizado ou distribuído existe somente um valor correto para cada objeto de dados. O termo consistência mútua é usado para indicar que todos os valores do mesmo item de dados convergem para este valor correto. Um banco de dados replicado é dito como em estado mutuamente consistente se todas as cópias tem o mesmo valor exato. Em adição, um banco de dados é dito como em um estado consistente se todas as restrições de integridade identificadas pelo banco de dados forem obedecidas.

Em um sistema banco de dados móvel a presença de dados

dependentes de localização define dois tipos de consistência: espacial e temporal.

A consistência espacial indica que todos os valores de item de dados de uma replicação espacial estão associados a uma e somente uma região de dados e eles satisfazem as restrições de consistência definidas pela região. Assim há um mapeamento 1:1 entre valores de dados e a região que eles servem.

Cada unidade móvel que inicia uma transação em uma região deve ter uma visão consistente da região e o banco de dados deve garantir que o efeito da execução das transações seja durável naquela região. Para alcançar este estado, a região deve satisfazer também a consistência temporal.

A consistência temporal indica que todos os valores do item de dados devem satisfazer um dado conjunto de restrições de integridade. Um banco de dados é temporariamente consistente se todas as réplicas temporais (replicação de itens de dados em múltiplos sites) de um item de dados tiverem o mesmo valor.

O isolamento de transações assegura que uma transação não interfere na execução de outra transação. O isolamento é normalmente reforçado por algum mecanismo de controle de concorrência. Assim como atomicidade, o isolamento é necessário para assegurar que a consistência é preservada.

Em sistemas de banco de dados móveis uma unidade móvel pode visitar múltiplas regiões de dados e processar dados dependentes de localização. A idéia importante é assegurar que fragmentos de execução satisfaçam o isolamento em nível de fragmento de execução. Eles estarão então sob algum mecanismo de controle de concorrência que deve reconhecer as relações entre um item de dados. O mecanismo deve reforçar o isolamento em cada região separadamente, mas alcançar o isolamento para a transação toda. Isto é completamente diferente de um sistema de banco de dados distribuído convencional que não reconhece replicação

espacial e assim não reforça o isolamento regional.

A durabilidade garante a persistência de itens de dados confirmados no banco de dados. Em sistemas de bancos de dados móveis a durabilidade é regional bem como global. Para réplicas espaciais e réplicas temporais; as durabilidades regional e global são, respectivamente, reforçadas.

A confirmação de transação não é afetada pela mobilidade, entretanto, por causa da presença de dados dependentes de localização, uma confirmação de localização é definida. Um *commit* de localização obriga um *commit* de transação para uma região. Por exemplo, um gerente de departamento inicia a seguinte transação em sua unidade móvel: “reservar 5 assentos para um restaurante vegetariano localizado a 20 quilômetros daqui”. Isto é uma transação de atualização dependente de localização e ela deve ser processada na região onde o restaurante está localizado. A confirmação dever ser enviada o mais rápido possível para o gerente. O servidor de banco de dados responsável por processar esta transação deve primeiro mapear a localização da requisição e a localização do restaurante e então acessar o banco de dados correto para fazer a reserva. A execução inteira fica confinada na região até a transação ser confirmada. Assim o processo de confirmação é idêntico à noção convencional de confirmação de transação (*commit*), entretanto os requerimentos para a confirmação são diferentes Isto é chamado de confirmação dependente de localização para diferenciá-lo da noção convencional de confirmação. É possível afirmar então que um fragmento de execução E_i satisfaz uma confirmação dependente de localização se as operações de fragmento terminarem com uma operação de confirmação e uma localização para o mapeamento de dados existir. Assim todas as operações em E_i operam em réplicas espaciais definidas na localização identificada pelo mapeamento de

localização. A confirmação é assim associada a uma única localização *L*.

Em um ambiente móvel a unidade móvel pode processar sua carga de trabalho em um modo continuamente conectado, desconectado ou em conexão intermitente.

No modo conectado a unidade móvel é continuamente conectada ao servidor de banco de dados. A unidade móvel tem a opção de *cache* de dados solicitados para melhorar a performance ou para solicitar dados do servidor a qualquer momento durante o processamento da transação. Se necessário, ela pode entrar em modo cochilo para economizar energia e tornar-se ativa novamente. Entretanto, este modo é caro para se manter e não é necessário para o processamento de carga de trabalho de usuários.

No modo desconectado uma unidade móvel se desconecta voluntariamente do servidor depois de renovar o *cache* e continua a carga de trabalho localmente. Em um tempo fixado ela conecta e manda seu *cache* inteiro para o servidor usando *wireless* ou uma ligação *wired*. O servidor instala os conteúdos do *cache* como um meio para que a consistência global seja mantida.

A conexão intermitente é similar ao modo desconectado, mas aqui a unidade móvel pode ser desconectada a qualquer momento pelo sistema ou voluntariamente pelo usuário. A desconexão pelo sistema pode ser por ausência de canal, bateria fraca, segurança, etc. O usuário pode desconectar a unidade móvel para economizar energia ou para processar dados localmente, ou nenhuma comunicação com o servidor é solicitada por algum tempo. Diferente do modo desconectado, o modo intermitente não tem um tempo fixo para conectar e desconectar uma unidade móvel. A conexão em demanda é também uma forma de conectividade intermitente porque a necessidade de dados de um usuário é

A consistência do banco de dados em modos conectado ou intermitentemente conectado é difícil de se definir e manter. Manter o processamento das transações ACID em estado conectado é fácil e isso pode ser tratado de maneira convencional. Entretanto, seu processamento em modo desconectado ou intermitentemente conectado requer novas abordagens de consistência de *cache*, novas abordagens de *lock*, novo protocolo de *commit*, novos *rollback* e esquemas de abortar; e, o mais importante, um novo modelo de transação ou novo meio de processamento de transações ACID.

3.2 Gerenciamento de transação em bancos de dados móveis

Um sistema de banco de dados móvel fornece um completo sistema de banco de dados e as funcionalidades da comunicação móvel. Ele ainda permite ao usuário móvel iniciar transações de qualquer lugar a qualquer hora e garante sua consistência de execução. Em caso de algum tipo de falha (transação, sistema ou mídia), o sistema de banco de dados móvel garante a recuperação da base de dados. Segundo a arquitetura de um sistema banco de dados móvel, suas propriedades essenciais são:

- Mobilidade geográfica;
- Conexão e desconexão (com qualquer servidor e a qualquer hora);
- Capacidade de processamento de dados;
- Comunicação *wireless*;
- Transparência;

- Escalabilidade;

Um sistema de banco de dados móvel é um sistema cliente/servidor multibase distribuído baseado em *Personal Communication System* (PCS) ou *Global System Mobile Communication* (GSM). Existem algumas diferenças nas arquiteturas GSM e PCS, entretanto elas não afetam o sistema de banco de dados móvel. A funcionalidade do banco de dados é fornecida por um conjunto de servidores de banco de dados que são incorporados sem interferir em qualquer aspecto da rede móvel genérica.

Em bancos de dados móveis um conjunto de computadores com propósitos gerais são interligados por uma rede de alta velocidade. Os computadores são classificados como *hosts* fixos e estações base ou estações de suporte móvel. Os *hosts* fixos não são equipados com *trasceivers*, portanto eles não se comunicam com unidades móveis. Uma ou mais estações base são conectadas a uma estação base controladora que coordena a operação de estações base usando seus próprios softwares armazenados quando comandados pelo *Mobile Switching Center* (MSC). Para coordenar os servidores de banco de dados algumas capacidades adicionais e simples de processamento de dados são incorporadas às estações base. A mobilidade irrestrita em PCS e GSM é suportada pelo link *wireless* entre estações base e unidades móveis tais como PDA, laptop e telefones celulares. Estes dispositivos móveis são definidos como *hosts* móveis ou unidades móveis. As estações base são equipadas com *transceivers* e comunicam-se com unidades móveis por canais *wireless*. Cada estação base serve uma célula cujo tamanho depende do poder dessa estação base. Na realidade uma estação base com alta capacidade não é usada por causa de um conjunto de fatores. Preferencialmente um conjunto de estações base de baixa capacidade são

estendidas para gerenciar o movimento das unidades móveis.

Para incorporar uma completa funcionalidade ao banco de dados é preciso incorporar servidores de banco de dados ao PCS ou GSM. Eles podem ser instalados em estações base ou *hosts* fixos. Existem, porém, alguns problemas com esta configuração. Estações base ou *hosts* fixos são *switches* e têm tarefas específicas para executar que não incluem funções de banco de dados. Para adicionar as funções de banco de dados toda a arquitetura de uma estação base (hardware e software) deve ser revisada, inaceitável do ponto de vista de comunicação móvel. Por conta disso a configuração não será modular ou escalável e qualquer mudança em componentes de banco de dados irá interferir na comunicação de dados ou voz. Por essas razões os servidores de banco de dados são conectados em sistemas móveis por linhas *wired* como nós separados. Cada servidor de banco de dados pode ser alcançado por qualquer estação base ou *host* fixo, novos servidores podem ser conectados e os outros antigos podem deixar a rede sem afetar a comunicação móvel. O grupo de MSCs e *Public Switched Telephone Network* (PSTN) conecta o sistema de banco de dados móvel para o mundo externo.Um servidor de banco de dados se comunica com uma unidade móvel somente por estações base.

Um usuário móvel deve procurar sempre economizar energia de sua unidade móvel. Para isso, uma unidade móvel pode ser alterada para os modos desligado, cochilo (*idle* ou *doze*) e ativo. Em modo desligado, uma unidade móvel não escuta ativamente a estação base. Quando em modo cochilo a unidade móvel não está se comunicando com a estação base, mas está ouvindo-a continuamente. Em modo ativo a unidade móvel está se comunicando e processando dados. A unidade pode mover-se de uma célula para outra em qualquer um desses modos e

as unidades móveis só encontram *handoff* em modo ativo. Os sistemas de banco de dados móveis têm múltiplos servidores de banco de dados e um banco de dados pode ser distribuído por partição ou parcial ou replicação completa. As replicações espaciais devem seguir as restrições dependentes de localização e dados livres de localização. Os dados livres de localização podem ser replicados em todas as regiões e têm os mesmos valores. Esta é uma das razões pela qual as réplicas temporais podem ser processadas usando a versão *read-one write-all* do mecanismo de controle de concorrência *two-phase locking*, que não é relevante para replicação espacial. Os dados dependentes de localização podem ser replicados para todas as regiões, mas cada região terá diferentes valores e corretos. Os três tipos básicos de replicação do banco de dados são: não-replicação, replicação distribuída tradicional (replicação temporal) e replicação dependente de localização. As réplicas distribuídas são cópias de um ao outro que podem ter diferentes valores temporariamente, mas existe somente um valor correto. O dado dependente de localização tem múltiplas cópias e valores corretos. O valor correto é determinado pela localização. Cada região então tem um valor correto para esse dado e ele pode ter réplicas temporais em regiões. O valor de um dado que existe em uma estação base de uma determinada região é replicado para a unidade móvel, mas ele tem o mesmo valor daquele no servidor de banco de dados. Trata-se de uma replicação temporal de dados dependente de localização. Cada objeto de dado determina unicamente o tipo de replicação (não replicação, temporal ou espacial) e a granularidade celular dos dados (CEP, área metropolitana).

Tabela 1- Replicação de dados

Arquitetura	Cópias	Valores Corretos	Mobilidade
Centralizada	Uma	Um	Não
Distribuída	Múltiplas	Um por tempo	Não
Móvel	Múltiplas	Um por localização e tempo	Sim

A mobilidade não tem efeito na replicação temporal, mas tem na replicação espacial. Isto significa que como uma unidade móvel se móvel de uma região de dados para outra, diferentes valores das réplicas espaciais podem ser necessários para o processamento de requisições.

3.2.1 Execução de transação

Sistemas de bancos de dados distribuídos utilizam o máximo de paralelismo no processamento da sua carga de trabalho para melhorar a performance do sistema. Eles implementam um processamento paralelo ao fragmentar uma transação em um conjunto de subtransações que são então executadas por múltiplos nós. A execução inteira requer um módulo de software chamado coordenador que gerencia um conjunto de atividades conduzindo até o fim da transação. São três os principais modos de se implementar um coordenador: centralizado, parcialmente replicado e totalmente replicado. No modo centralizado um dos nós do sistema de banco de dados distribuído serve como coordenador. Em modo parcialmente replicado, um subconjunto de nós que servem como coordenadores. Por último, todos os nós servem como coordenadores no modo totalmente replicado. Para identificar um coordenador em sistemas de banco de dados móveis algumas importantes características devem ser observadas. Um coordenador deve ter uma comunicação direta e contínua com os outros nós, alta

confiabilidade e disponibilidade, teoricamente uma fonte de energia contínua e ilimitada e grande espaço de armazenamento.

Ao contrário dos convencionais bancos de dados distribuídos existem vários tipos de nós de processamento em sistemas de banco de dados móveis, mas somente alguns podem assumir o papel de coordenador.

As unidades móveis não fornecem conectividade contínua porque são sujeitas a imprevisíveis *handoffs*. Elas têm a capacidade de armazenamento e suporte de energia limitados e são recursos pessoais de um usuário que podem deixar a rede a qualquer momento. Por tudo isso não servem como coordenador.

Os servidores de bancos de dados estão continuamente disponíveis e têm capacidade suficiente de armazenamento, porém não têm comunicação direta com outros nós e não são equipados com *trasceivers* para comunicação *wireless*. Desse modo não servem como coordenador.

Os *hosts* fixos não são equipados com *trasceiver*, por isso não podem se comunicar com dispositivos móveis e não servem como coordenador.

Uma MSC possui todos os requerimentos para ser um coordenador, mas tem de usar os serviços de uma estação base para alcançar uma unidade móvel e, assim, não servem como um coordenador eficiente.

As estações base satisfazem todos os requisitos necessários para um coordenador e, assumindo esse papel, não se sobrecarrega. Por isso, uma estação base é o nó mais indicado para servir como coordenador.

3.2.2 Processamento de transação

Uma transação em um sistema de banco de dados móvel pode ser iniciada a partir de um servidor de banco de dados, a partir de uma unidade móvel

ou ambos. Ela pode ser processada inteiramente pela unidade móvel onde ela foi iniciada ou inteiramente pelo servidor de banco de dados ou pela combinação dos dois. Se ela é processada inteiramente por um nó (nó origem ou qualquer outro) então o coordenador assume somente o menor papel na execução e confirmação da transação.

Com mais de um nó envolvido na execução algumas situações podem ser observadas. Quando uma transação tem origem em uma unidade móvel, três cenários são possíveis: a transação é executada na unidade móvel e em um grupo de servidores de banco de dados e o resultado final vai para a unidade móvel; a transação é processada somente em um grupo de servidores de banco de dados e o resultado final vai para a unidade móvel; a transação é processada somente na unidade móvel: Cabe ressaltar que nenhuma outra unidade móvel pode ser envolvida na execução porque: uma unidade móvel é um recurso de processamento pessoal de um cliente que pode ser desconectado ou desligado por seu proprietário a qualquer momento; o proprietário de uma unidade móvel pode não querer dividir seu recurso com carga de trabalho de outros. O resultado final vai para a unidade móvel que iniciou a transação. Quando uma transação tem origem em um servidor de banco de dados, ela é executada em um conjunto de servidores de banco de dados e o resultado final vai para o servidor que iniciou essa transação.

Quando uma unidade móvel inicia uma transação ela checa se a transação pode ser inteiramente executada localmente (em unidade móvel). Neste caso a unidade móvel executa e confirma a transação usando seus dados em *cache*. No fim da confirmação (*commit*) a unidade móvel envia atualizações de transação para os servidores de bancos de dados instalá-las no banco de dados e

manda o resultado para o usuário. Se ela não pode ser executada inteiramente há duas opções: transferir os itens de dados requeridos dos servidores de bancos de dados para a unidade móvel ou distribuir a transação para um grupo de nós.

No primeiro caso, a unidade móvel identifica um grupo de dados requeridos para processar a transação. Ela se comunica com as estações base que encontram os servidores de bancos de dados e movem seus itens de dados para um *cache* de unidade móvel. A unidade móvel executa e confirma (*commit*) a transação e envia o resultado final ao usuário. Ela também envia as atualizações através das estações base para os servidores de bancos de dados para instalá-las no banco de dados. Este esquema não requer os serviços do coordenador, mas gera uma quantia significativa de overhead de comunicação.

No segundo caso, a unidade móvel divide a transação em um número de subtransações. Ela mantém as subtransações que pode executar e, com a ajuda do coordenador, distribui o restante para um subgrupo de servidores de bancos de dados para execução. Uma estação base corrente da unidade móvel torna-se o coordenador dessa transação e gerencia a execução inteira.

O movimento da unidade móvel dificulta a tarefa de coordenador quer precisa coordenar a execução e o movimento da unidade móvel. Com a mobilidade os seguintes cenários de processamento são possíveis: a unidade móvel não se move, a unidade móvel se move, a unidade móvel se move e processamento distribuído.

No primeiro caso uma transação chega e completa seu processamento inteiramente na unidade móvel. A unidade móvel não se move durante a execução da transação. Este é um caso similar ao convencional processamento de dados centralizado.

No segundo caso uma transação chega e completa sua execução inteiramente na unidade móvel também. Os itens de dados requeridos são movidos para outros nós. Durante a execução da transação a unidade móvel se move para diferentes células. Este tipo de execução é chamado de *local atomic* e é bastante comum em computação móvel.

No terceiro e último caso uma transação que tem origem em uma unidade móvel é fragmentada. As subtransações são distribuídas entre a unidade móvel e um grupo de servidores de bancos de dados. Um coordenador é identificado para gerenciar a execução. A unidade móvel se move para diferentes células durante a execução da transação

O serviço de um coordenador para uma unidade móvel deve ser continuamente disponível durante a execução de uma transação. Esta ligação deve ser quebrada quando a unidade móvel atravessa o limite da célula e entra em uma célula diferente. O link pode ser mantido de dois modos: método estático ou método dinâmico.

No método estático, quando uma transação se origina em uma unidade móvel então a estação base desta unidade móvel torna-se o coordenador da transação e continua coordenador até a transação se confirmar (*commit*). A unidade móvel deve continuar a migrar de uma célula para outra enquanto processa sua subtransação, mas o coordenador não muda. É de responsabilidade da unidade móvel informar a nova identidade da estação base a seu coordenador. Isto pode ser feito quando a unidade móvel registra a nova estação base que se comunica com o coordenador já que ela sabe sua identidade. O problema deste simples método é que o coordenador e cada unidade móvel podem ter que passar através de um número de estações base para atualizar uma à outra. Isso torna a

confirmação (*commit*) de transação e o *logging* muito complexo.

No método estático a parte do coordenador se move com a unidade móvel quando esta vai para outra célula e sua estação base se torna o coordenador da transação que está sendo executada pela unidade móvel. Uma vez que uma transação está sendo processada por múltiplos servidores de bancos de dados, eles devem saber quando um novo coordenador é designado para uma transação existente. Isso é feito pelo novo coordenador que informa a todos os servidores de bancos de dados desta transação a identidade do novo coordenador.

Embora no método dinâmico uma unidade móvel é diretamente conectada a seu coordenador o tempo todo ele se torna um problema para uma unidade altamente móvel. Considerando uma unidade móvel que se move rapidamente entre duas células várias vezes, é possível que em tal situação o processo de mudança do coordenador e o processo de *handoff* possam sair de sincronia. Assim, uma estação base anterior pode continuar sendo coordenador enquanto a unidade móvel está em outra célula. Nesta situação altamente móvel a mudança do coordenador não é desejável. Dois esquemas podem ser aplicados no tratamento de coordenador para um cenário altamente móvel: o limite residencial e a migração não adjacente.

No esquema de limite residencial uma duração máxima de residência para uma unidade móvel (quanto tempo uma unidade móvel deve ficar em uma célula antes do coordenador ser mudado) é definida. O coordenador migra para a estação base corrente somente se a residência da unidade móvel em uma célula for maior que o limite residencial predefinido. O valor para um limite de residência deve ser aplicado para todas as unidades móveis ou deve ser único para uma unidade móvel específica. Um conjunto de parâmetros podem ser usado para

definir um limite de residência, tais como: comportamento de movimento, congestionamento de tráfego, o período do dia e assim por diante. O movimento físico e o congestionamento de tráfego podem variar significativamente, porém eles provavelmente exibem alguns padrões. Em um determinado período do dia (hora do rush) uma unidade móvel pode ser altamente móvel e o volume de tráfego na rede pode ser alto. Assim, o valor de residência pode ser computado dinamicamente caso a caso. A taxa de mudança de coordenador pode ser controlada pelo parâmetro de limite de residência.

Na migração não adjacente o coordenador migra somente se a unidade móvel se mover para uma célula não adjacente. As células não adjacentes são aquelas que não tocam diretamente a célula onde a unidade móvel reside atualmente.

3.2.3 Modelos de transação móvel

É plenamente reconhecido que o modelo de transação ACID convencional é incapaz de gerenciar satisfatoriamente as tarefas de processamento de dados móveis. Algumas das razões para isso: presença de *handoff*; alternância de modos (desconectado, cochilo e ativo), escassez de recursos, dados dependentes de localização. Para gerenciar o processamento de dados na presença desses novos desafios é necessário um modelo mais forte de transação ou um modelo de execução de transação ACID que pode manter a mobilidade durante o processamento de dados.

O tópico inteiro de modelagem de transação móvel é altamente orientado a pesquisa e, embora um significativo número de esquemas vem sendo proposto, nenhum se tornou um método comumente aceito. Há basicamente dois

modos de controlar pedidos transacionais em sistemas de bancos de dados móveis: o modelo de execução baseado em *framework* de transação ACID e o modelo de transação móvel e sua execução.

A primeira abordagem cria um modelo de execução baseado no *framework* de transação ACID. Na segunda abordagem uma requisição de usuário é mapeada para um modelo de transação móvel e executada sob restrições ACID móveis. A abordagem de modelo de execução conseguiu tratar de mobilidade de informação de localização, mas seu escopo era um pouco limitado. Isto rendeu um crescimento para o desenvolvimento de modelos de transação móvel que assimilou a mobilidade e a propriedade de localização em suas estruturas.

O modelo de transação convencional traz a certeza que as propriedades ACID de uma transação são mantidas durante o processamento do banco de dados. A introdução da mobilidade mudou significativamente a arquitetura de banco de dados e o paradigma de gerenciamento, tornando claro que os rigores forçosos das propriedades ACID não mantiveram necessariamente a consistência de banco de dados. A mobilidade mudou e, em alguns casos, relaxou a noção de consistência porque em sistemas de banco de dados móveis essa noção é fielmente relacionada a localizações em domínios geográficos.

O domínio geográfico *G* é a área geográfica total coberta por todas as unidades móveis de um sistema celular. Assim, $G=(C_1+C_2+...+C_n)$, onde C_i representa a área de uma célula. Uma localização é um ponto preciso dentro do domínio geográfico fixo. Ela representa a menor posição identificável no domínio. Cada localização é identificável por um *id* específico *L*. Na realidade uma localização é identifica pela referência à estação base. É importante entender a complexa relação entre dados, entre as operações a serem executadas nos dados

e as conclusões das execuções para o desenvolvimento de um modelo de execução.

Em sistemas legados, a freqüência de acesso dos itens de dados é usada na distribuição de dados (partição e replicação parcial) e não sua associação com localizações geográficas. Em sistemas de bancos de dados móveis essa associação tem uma importante função nos seu processamento assim como em sua distribuição.

Uma requisição dependente de localização é um tipo de requisição que processa dados dependentes de localização. A localização da requisição deve ser determinada e disponível ao processador para obter resultados válidos. Um processo chamado *location binding* é usado para associar a localização da requisição com a requisição em si e para determinar os resultados corretos. As definições corretas de dados dependentes de localização e replicação espacial podem ser vistas como baseadas sobre um mapeamento de um objeto para uma localização. Assim, tendo um objeto, este mapeamento identifica as corretas regiões de dados para aquele objeto.

O grupo de regiões de dados para um objeto de dados é um particionamento do domínio geográfico. No caso onde nenhuma replicação espacial de um objeto de dados existe o número de regiões de dados para aquele objeto é 1. Tendo uma localização específica em um domínio geográfico, cada objeto é associado a uma única região de dados.

Uma região de dados é identificada pelo objeto de dados e localização. Assim que uma unidade móvel se move cada localização identifica unicamente a região de dados para a qual cada dado pertence. Isto nos dá a base para executar requisições transações dependentes de localização. Uma requisição

dependente de localização normalmente retorna os valores de dados para a localização na qual ele executa. Observe que essa abordagem é consistente com aquela de um ambiente centralizado ou distribuído onde somente uma região de dados existe para todos os objetos de dados.

Em um ambiente centralizado ou distribuído existe somente um valor correto para cada objeto. O termo consistência mútua é usado para indicar que todos os valores convergem para este mesmo valor correto. Um banco de dados replicado é dito em estado mutualmente consistente se todas as cópias têm o mesmo valor exato. Além disso, um banco de dados é dito em estado consistente se todas as restrições de integridade identificadas pelo banco são satisfeitas. Este conceito básico de consistência é fundamentado em replicação temporal. A idéia de consistência é complexificada com a replicação espacial. Um método convencional de assegurar a consistência mútua é chamado seriabilidade de uma cópia. Isto implica intuitivamente que o comportamento de transações distribuídas se apresenta para prover valores de réplicas somente se não houver replicação. Assim, o agendamento de operações de atualização nas réplicas assegura que a consistência mútua é eventualmente alcançada. Em um *snapshot* do banco de dados este poderia ser o único valor correto. Em bancos de dados temporais este poderia ser o valor mutualmente consistente para aquele particular *time point*. A consistência mútua dentro de bancos de dados temporais implica que as réplicas para um *time point* devem eventualmente ter o mesmo valor. Quando a replicação espacial é adicionada a localização de dados também deve ser adicionada para isto. A consistência mútua com replicação espacial indica que as réplicas espaciais para uma localização (região de dados) convergem para o mesmo valor.

No ambiente do sistema de banco de dados móvel o conceito de

consistência é mais complexo porque também envolve a localização geográfica dos dados. Em sistemas de bancos de dados móveis uma visão consistente é obtida com referência à localização geográfica dos dados. Uma transação deve modificar um valor de dado que poderia ser válido somente para uma região de dados específica. Assim, uma transação de atualização idêntica (em uma diferente localização) deve ser sujeita a diferentes grupos de restrições de consistência, que dependerão do lugar da manipulação dos dados.

Para demonstrar o significado da replicação espacial e sua consistência, uma cadeia de hotéis pode ser tomada como exemplo. Cada filial desta cadeia oferece serviços idênticos, mas a política, custo ou facilidades podem depender do local de cada filial. Um mesmo tamanho de suíte pode ter preços diferentes para cada filial. Dessa forma as restrições de consistência de dados muda de uma filial para outra por conta das taxas e políticas locais. Mesmo que cada filial compartilhe o mesmo esquema suas instâncias (valores de dados) podem diferir. Qualquer mudança de diárias de suíte por uma transação em uma filial não afeta as diárias de outra localização. Isto indica que os convencionais estados de consistência de dados globais e mútuos não se aplicam em bancos de dados móveis. O conceito de consistência espacial é introduzido aqui para distinguí-lo da consistência convencional que é referido como uma consistência temporal.

Os exemplos ilustrados nas figuras a seguir podem ser tomados para explicar um típico caso de inconsistência de localização. Supondo que um motorista durante uma viagem planeja um intervalo de acordo com a condição do tempo. Se o tempo no local destino é ruim ele decide reservar um hotel a algumas milhas de sua localização atual e comer em um restaurante próximo. De outra forma ele decide continuar dirigindo em direção ao seu destino e comer em um

Figura 1 – Execução de transação durante mobilidade

Supondo que o destino é 20 milhas do ponto de partida ele lança uma transação *T* de sua unidade móvel que processa o seguinte grupo de fragmentos de transação:

- T_1 verifica o tempo a 20 milhas da localização corrente;
- T_2 verifica um hotel a 20 milhas da localização corrente;
- T_3 encontra uma lista de restaurantes a 20 milhas da localização corrente;
- T_4 verifica um hotel a X milhas da localização corrente;
- T_5 lista os restaurantes a X milhas da localização corrente;

Se o tempo no local destino for bom então a possibilidade de ordem de execução é T_1, T_2 e T_3; de outra forma a ordem de execução é T_1, T_4 e T_5. Um cenário de tempo ruim é usado para analisar uma possível ordem de execução ilustrando como uma inconsistência de localização aparece por causa da mobilidade da unidade móvel. O motorista parte para o destino *Y* e, após viajar algumas milhas ele lança *T1* (“Como está o tempo em *Y*?”) enquanto se move. Ele obtém a resposta de *T1* (“Mau tempo em *Y*”) no local *A*. Em um ambiente móvel, em *B* ele envia *T4* (“Reservar um hotel em 5 milhas”) e então envia *T5* (“Lista de restaurantes a 5 milhas de *B*. Surgem então algumas possibilidades:

- O viajante obtém a resposta de *T4* quando ele está a 5 milhas de *B*, isto é consistente com a requisição;

- O viajante obtém a resposta de *T4* 5 milhas além de *B* e, nesse caso, há inconsistência de localização;
- Se a segunda possibilidade ocorre então a resposta para *T5* será inconsistente (inconsistência de localização);

Na segunda possibilidade a região de dados associada ao objeto (restaurante) para a requisição é diferente daquele da localização corrente da unidade móvel.

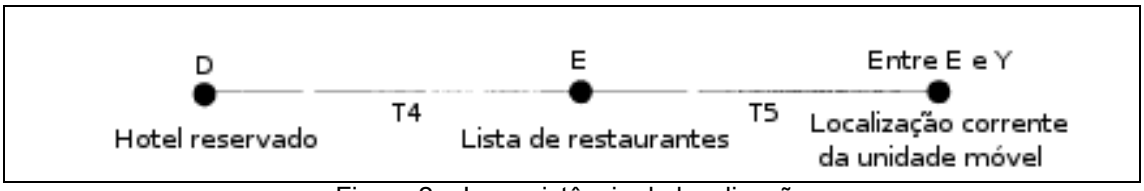

Figura 2 – Inconsistência de localização

A figura 2 mostra uma instância de inconsistência de localização. O resultado é que o hotel é reservado na localização *D* e os restaurantes .listados são da localização *D* também. A localização corrente está além de *D*, a unidade móvel teria de voltar para *D* onde o hotel é reservado e para a localização *E* para o restaurante.

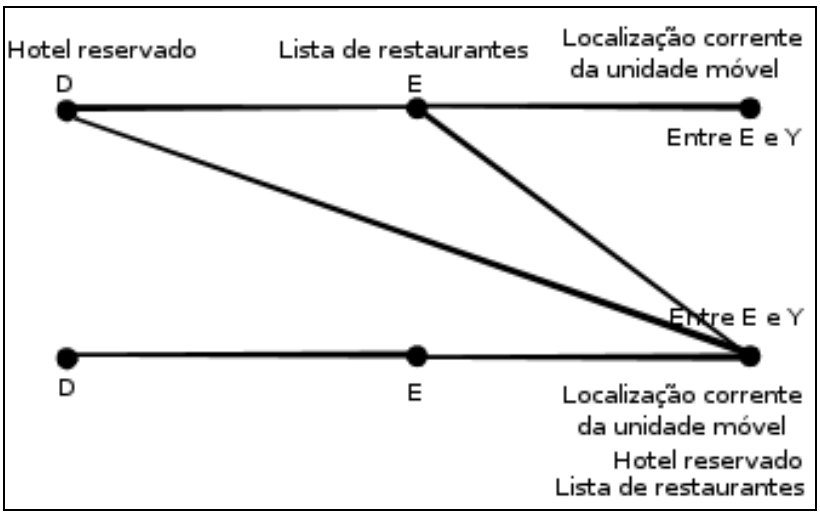

Figura 3 – Consistência de localização

Um correto modelo de execução de transação deve tratar esses casos. Uma quantidade de modelos de execução tem sido desenvolvida. Estes

modelos são baseados em modelos ACID básicos e avançados existentes tais como ACTA, SAGA e outros e suportam a execução de transação distribuída. A abordagem de execução de transação é similar, isto é, cada um suporta o fragmento de transação (subtransação) que satisfaz as restrições ACID para uma grande amplitude de casos. Entretanto eles se diferem em como estes fragmentos são executados e confirmados (*commit*), conduzindo para diferentes modelos de execução para o processamento de bancos de dados móveis. Os casos importantes que cada modelo de execução considera são as restrições de recurso (canal *wireless*, memória, espaço em disco, energia limitada e canal de banda) e a mobilidade das unidades de processamento.

3.2.3.1 Pro-motion

Este modelo de execução foi projetado para focar nos casos de problemas introduzidos pela desconexão e pelos limites de recursos do ambiente móvel. Seu principal bloco de formação é o *compact* que funciona como uma unidade básica de replicação de dados e é formado por: dados, métodos de acesso a esses dados, informações sobre o estado do *compact*, regras de manutenção de consistência global, obrigações tais como deadline, e uma interface para a unidade móvel gerenciar o *compact*.

O servidor chega a um acordo com a unidade móvel onde uma transação está para ser executada através do *compact*. Sob este acordo a unidade móvel é livre para executar a transação e fazer suas atualizações disponíveis para um servidor através do *compact*. Se há alguma necessidade para mudar algumas das condições de execução de transações, então esta mudança é renegociada através do *compact*.

O servidor e a unidade móvel, juntos, gerenciam o *compact*. A criação e remoção de um compact são tratadas pelo servidor de banco de dados. Quando uma unidade móvel começa a transação e precisa de dados ela envia um pedido para o servidor. Se os dados estão disponíveis então o servidor cria um *compact*, inicializa-o com dados e outras informações necessárias e envia-o para a unidade móvel. Se já existe um *compact* para a execução desta transação então, qualquer pedido mais adiante da unidade móvel é satisfeita pelo *compact* atualizado. A chegada ou modificação de um *compact* é registrada em estruturas de dados chamadas *compact registry*. O histórico do *compact* fornece a informação completa sobre todos os *compacts* abertos iniciados por uma unidade móvel.

O processamento de uma transação em uma unidade móvel é gerenciado por um agente *compact*. Uma transação, durante sua execução em uma unidade móvel, deve gerar uma quantidade de requerimentos. Um agente *compact*, em adição ao gerenciamento do grupo de *compacts* listados nos registros, é responsável por processar pedidos de transação. Suas funções devem ser incorporadas ao gerenciador de transação dos convencionais sistemas de gerenciamento de bancos de dados. Um agente *compact* é responsável por: forçar um mecanismo de controle de concorrência, *logging* e recuperação. Uma vez que um *compact* é um objeto independente uma quantidade de restrições de consistência e *cache* de dados podem ser eficientemente suportados.

Um agente *compact* tem os seguintes métodos disponíveis para gerenciar a execução de uma transação de forma eficaz: *inquire*, *notify*, *dispatch*, *commit* e *abort*.

Inquire pesquisa o estado de um *compact*. *Notify* avisa o *compact* de alguma mudança de estado da unidade móvel. *Dispatch* processa operações no

compact emitido pela execução da transação na unidade móvel. *Commit* confirma a execução da transação na unidade móvel. *Abort* rejeita ou remove as mudanças feitas no *compact* pela transação.

Um *compact* também é capaz de suportar métodos especializados que poderiam ser solicitados pelo controle de concorrência especial ou manipulação de dados específica.

Existe ainda um gerenciador de *compact* que armazena o estado de cada *compact* e, em intervalos predefinidos, o *compact* é atualizado toda vez que o estado de um *compact* muda. Deste modo ele age como um *front-end* para o servidor de banco de dados e o defende de unidades móveis que executam transações. O servidor de banco de dados interage com um gerenciador de *compact* como um cliente que executa a transação raiz. A transação raiz é fragmentada sob um modelo *open-nested* e cada unidade móvel executa um ou mais fragmentos. A execução destes fragmentos irmãos procede de forma independente sem violar as restrições de consistência definidas nos *compacts*. A transação raiz é confirmada pelo gerenciador de *compact*.

O processamento de transação sob o modelo *Pro-motion* tem quatro etapas que são executadas pelo agente *compact*. Estas etapas se preparam para execução desconectada, execução conectada e atualizações de servidor de banco de dados. As quatro etapas são: *hoarding*, execução conectada, execução desconectada e resincronização.

Em *hoarding*, o gerenciador de *compact* prepara e armazena as *compacts* requeridas para gerenciar a execução desconectada de transações. A unidade móvel pode ser desconectada da rede a qualquer momento. Na execução conectada, a unidade móvel é conectada à rede e o gerenciador de *compact*

processa as transações. Na execução desconectada a unidade móvel desconecta da rede e o gerenciador de *compact* processa as transações localmente. Na resincronização a unidade móvel, depois de completar a execução desconectada, é reconectada ao servidor e o agente *compact* reconcilia atualizações locais executadas durante a execução desconectada com o servidor de banco de dados.

A execução inteira é gerenciada pelo esforço coordenado do *compact*, gerenciador de *compact* e agente *compact*. A tarefa de execução conectada é simples e similar a qualquer processamento de transação em sistemas legados distribuídos. A unidade móvel pode processar a transação e as atualizações são instaladas no servidor de banco de dados. A operação desconectada se torna complexa na presença de mobilidade. As etapas responsáveis por gerenciar a execução desconectada são: *hoarding* e *caching*, e processamento de transação desconectada.

Em *hoarding* e *caching*, uma unidade móvel requer uma quantidade de recursos incluindo dados para execução de transação. Cada pedido de recurso é implícita ou explicitamente associado a um específico tipo de *compact*. O agente *compact* , a aplicação móvel e o usuário , juntos, constroem e mantém a lista de recurso para uma unidade móvel em benefício de uma transação. O agente *compact* monitora continuamente as necessidades de recursos adicionais e os adiciona para a lista como progressos de execução. Se uma aplicação tenta usar os dados que não estão na lista os agentes *compact* imediatamente os adicionam e tentam obtê-los. Existe uma quantidade de modos de etapas *hoarding* que podem ser executados, e o desenvolvedor da aplicação tem flexibilidade suficiente para implementar qualquer esquema adequado para a plataforma de banco de dados móvel.

Em *pro-motion* é possível ter múltiplos pedidos para o mesmo recurso com diferentes tipos de acesso. Por exemplo, uma aplicação *A* requer um item de dados *X* com permissões de leitura e escrita, e outra aplicação *B* somente quer ler *X*. Se a aplicação *A* é, mais tarde, removida da unidade móvel então *X* ainda será solicitada, mas o nível de acesso deve cair para “somente leitura”. Por outro lado, se a aplicação *B* é removida então o pedido associado a ela é removido e o *compact* é ajustado para cumprir com as solicitações de pedidos remanescentes.

Um conflito entre um proprietário de um recurso (unidade móvel) e um *compact* solicitante é resolvido com a ajuda do *compact* de servidor. Se o pedido pode ser satisfeito, um *compact* de unidade móvel com integridades apropriadas e obrigações é criado e enfileirado para a transmissão. Senão um *compact* nulo é criado e enfileirado para ser mandado para a unidade móvel. Se o recurso está disponível então o gerenciador de *compact* obtém direitos de modificações de dados e encontra o tipo de *compact* compatível. O gerenciador de *compact* cria uma instância do tipo de *compact* de servidor apropriada e transfere o pedido para o *compact* criado recentemente. O *compact* de servidor cria uma instância de *compact* da unidade móvel compatível com os dados e códigos necessários e enfileira o novo *compact* da unidade móvel para transmissão para a unidade móvel. Se o recurso está segurado por uma transação estacionaria então um *compact* nulo é criado e enfileirado para transmissão para a unidade móvel.

No processamento desconectado de transação, o agente *compact* numa unidade móvel começa o processamento local quando o link entre o servidor e unidade está indisponível. No processamento local o agente *compact* e os *compacts*, juntos, gerenciam todas as operações. O agente *compact* mantém um

log de eventos para gerenciar o processamento de transação, a recuperação e a sincronização. A execução completa envolve a iniciação da transação, a execução da transação, completar a transação e recuperação de falha.

Uma aplicação inicia uma transação numa unidade móvel ao enviar um evento *begin*. Quando o agente *compact* recebe este evento ele assinala um *ID* de transação único e os escreve (evento e *ID*) no *log* de evento. O *Pro-motion* permite uma quantia de opções para o evento *begin* para mais adiante especificar o comportamento da transação. Duas das opções são o controle de transação de níveis de isolamento e confirmação local. Um evento *begin* com opção local confirma localmente e faz de seus resultados disponíveis para outras transações com a possibilidade de falha no servidor. Uma transação sem opção local não confirma (*commit*) localmente até as atualizações serem confirmadas no servidor. Um evento *begin* deve ter também um *ID* de grupo. Na ausência de tal *ID* o *compact* assinala um *ID* e o retorna para a nova transação. A aplicação pode passar esta informação para transações adicionais, permitindo-as unir transações ativas. Estas transações adicionais recebem o mesmo *ID* de grupo mas diferentes *IDs* para suas transações. Todas as transações com o mesmo *ID* de grupo são *commit-dependents*, isto é, elas terminam juntas.

Quando uma transação precisa acessar dados durante a execução um evento OP é enviado para o agente *compact* com o *compact ID*, a operação a ser executada, os parâmetros necessários para se completar a operação, o *ID* de transação e o *ID* de grupo. O *compact ID* e os eventos *OP* são usados pelo agente *compact* para invocar o método de despacho de *compact*. O *compact* determina a presença de um conflito entre as novas e quaisquer operações pendentes. Se houver algum conflito o evento *OP* é retornado para o agente *compact* com um

código *conflict* e uma lista de *IDs* de transações conflitantes. O agente então enfileira o evento *OP* para despachar para o *compact* quando o evento do *compact* muda. A informação salva na fila é suficiente para a detecção de *deadlock* e sua resolução. Na ausência de um conflito a operação é executada pelo *compact*.

A confirmação de transação é executada usando um protocolo de *commit* de duas fases. A confirmação é iniciada quando um *commit* é emitido pela transação. Diferente da maioria dos sistemas de processamento de transação o *Pro-motion* habilita um procedimento de contingência para ser anexado a cada evento *commit* e logado com o evento.Quando um evento *commit* é aceito por uma transação o agente *compact* coordena e confirma a transação.

3.2.3.2 Modelo de execução com transação reporting

O modelo de execução com transação *reporting* é considerado um modelo transacional aninhado aberto que relaxa as restrições impostas pela transação atômica. Estas transações convencionais trabalham com restrições que não permitem características como particionamento da computação e o compartilhamento de resultados parciais. Estas características são fundamentais em ambientes móveis por conta da escassez de recursos e baixa largura de banda.

Uma transação *reporting* compartilha suas atualizações com transações pai a toda hora durante sua execução. Este tipo de transação periodicamente se reporta a outras transações para delegar alguns de seus resultados correntes. Assim, as transações *reporting* são associadas à primitiva de transação *report* em adição às primitivas *begin*, *commit* e *abort*. *Begin* é usada para iniciar uma transação *reporting*, *commit* e *abort* são usadas para encerramento. O modo de *report* depende se ela é ou não associada a uma transação compensante.

Uma co-transação também compartilha seus resultados com a transação pai, mas, diferente de uma transação *reporting*, sua execução é suspensa quando ela se reporta à transação pai e retoma sua execução é retomada do ponto da suspensão. Similar a uma transação de reportagem uma transação delega seus resultados correntes para sua co-transação ao invocar a primitiva de transação *report*. Componentes de co-transação mantêm seu estado de execução e, por esta razão, eles não podem ser executados concorrentemente.

Os componentes *reporting* e co-transação são responsáveis principalmente por todas as modificações de dados. Componentes *reporting* sempre executam em estação base assim como co-transações executam em unidade móvel. Quando uma unidade móvel rodando uma co-transação se move e se registra em outra estação base, o componente *reporting* correspondente também se move para a estação base e a comunicação entre *reporting* e co-transação permanece ativa. Eles compartilham seus resultados parciais e então as co-transações têm os valores de itens de dados mais atualizados, e o componente *reporting* recebe atualizações para serem instaladas no banco de dados. Os componentes compensantes são invocados para reparar alguma inconsistência ocorrida durante a execução destes componentes.

Este modelo de execução que foi baseado no *framework open-nested* não visou os casos de processamento de dados dependentes de localização. Seu principal objetivo foi representar e suportar a execução de transação ACID usando componentes móveis.

Nesta abordagem as transações de *reporting* e as co-transações são tratadas como unidades independentes de ação. Estas transações suportam a delegação entre transações.

3.2.3.3 Modelo de execução de transação pré-escrita

A abordagem desse modelo de execução é feita para primeiro executar e confirmar uma transação em nível lógico e fazer seus itens de dados disponíveis para outras transações concorrentes. Este *commit* lógico é alcançado através do *pre-write*. O resultado desta transação não instalado no banco de dados final. A transação final é finalmente confirmada no nível físico somente quando ela satisfaz um grupo de restrições predefinidas.

Um *pre-write* anuncia qual valor um item de dados terá após o *commit* desta operação *write*. Em outras palavras, ele anuncia o *after image* (*AFIM*) de um item de dados antes da aplicação de um *write*. Assim, o resultado de um *pre-write* pode ser anunciado como {*nome de arquivo, número de registro, nome de campo, valor nome*} para uma simples aplicação de banco de dados. A noção de uma virtual execução de *pre-commit* de uma transação pode ser expressa em termos de *pre-write*. Por exemplo, uma transação pode ser chamada *pre-commited* se ela tem anunciado todos seus valores *pre-write*, mas a transação não foi confirmada. Da mesma forma, um *pre-read* retorna um valor *pre-write* do item de dados enquanto que um *read* convencional retorna o resultado de uma operação *write*. Um *pre-read* torna-se um *read* fraco se ele retorna o resultado de um *pre-write* de uma transação completamente confirmada. Este é um modo de usar *read* fraco. O modelo de execução de transação proposto é baseado nestes grupos de operações.

Cada operação *pre-write* de uma transação mostra o valor *AFIM* do item de dados. Os valores de *pre-write* dependem dos ambientes de execução. Em alguns casos seus valores devem ser idênticos a aqueles produzidos pelo *write*

convencional, mas em alguns casos podem diferir.

Uma transação pré-confirma depois que ela tiver lido ou pré-lido seus itens de dados requeridos e os valores pré-escritos são computados. Depois de uma transação pré-confirmar todos seus valores pré-escritos eles são feitos disponíveis para outras transações concorrentes. Uma vez que *pre-writes* não tocam a cópia final do banco de dados eles são tratados no nível de gerenciador de transação e *writes* convencionais são tratados pelo gerenciador de dados.

A ordem de *commit* de transações concorrentes é decidida no *pre-commit*. Isto ajuda a detectar conflitos bem cedo. Uma transação pode ser abortada ou retornada (*roll back*) antes de seus *pre-commits* e *roll-back* não pode ser em cascata.

Através do uso de *pre-read, pre-write* e *pre-commit* o modelo relaxa as restrições de isolamento e uma transação faz seus itens de dados disponíveis para outra transação imediatamente após seus *pre-commits*, mas antes de seu *commit* final.

Com relação à execução de *pre-write* em sistemas de bancos de dados móveis, o modelo de execução busca trabalhar nas restrições de recursos do sistema. Uma consideração importante é a velocidade de CPU (*Central Processing Unit*). Para CPUs lentas o modelo de execução não escala muito bem e as unidades móveis agem como simples clientes sem capacidade de processamento do banco de dados. Unidades móveis com CPUs velozes armazenam itens de dados consistentes em seus *caches* locais. Quando uma transação chega em uma unidade móvel, ela usa os dados em *cache* para processar *reads* e retornar os valores de *pre-write*. Aqueles *reads* para os quais o *cache* não tenha dados são enviados para o processamento do servidor. O servidor retorna os valores de *pre-*

write ou *write*. A transação pré-confirma quando todos os valores são retornados pelo servidor e a unidade móvel tem também completados seus processamentos. Todos os *locks* são liberados e o *schedule* de *pre-commit* é enviado para o servidor para o *commit* final da transação.

Em algumas situações de processamento uma tolerável diferença entre uma versão de *pre-write* e uma versão final de *write* pode aparecer.

3.2.3.4 Modelo de transação HiCoMo

High Commit Mobile (HiCoMo) é um modelo de transação para o processamento de dados agregados armazenados em *data warehouse* que reside em unidades móveis. Uma vez que a *data warehouse* reside em unidades móveis, as transações HiCoMo são sempre iniciadas em unidades móveis onde elas são processadas em modo desconectado. Assim, as confirmações de transação são mais rápidas. Os resultados destas transações são então instaladas no banco de dados após reconexão.

A base do banco de dados reside na rede fixa. Ela é manipulada por transações chamadas base ou *source*. Testas transações são iniciadas na rede fixa. As transações que são iniciadas e processadas em unidades móveis soa chamadas de HiCoMo. Este modelo impõe algumas restrições: os dados agregados a serem armazenados nas unidades móveis devem ter somente valores como média, soma, mínimo e máximo; operações como subtração, adição e multiplicação são permitidas com algumas restrições em suas ordens de aplicação; o modelo permite alguma margem de erros que podem ser definidas antes das operações permitidas serem iniciadas e seus valores podem ser variados entre o limite superior e inferior.

A estrutura da transação HiCoMo é baseada no modelo de

transação *nested*. A consistência de banco de dados é satisfeita através de critérios de convergência. Ela é satisfeita quando o estados do banco de dados base e do *data warehouse* em unidades móveis são idênticos. Este modelo de transação assegura que a convergência é sempre satisfeita.

Como o banco de dados base no servidor é atualizado por transações *source* é necessário instalar atualizações de transações HiCoMo. Para isso, elas devem ser convertidas para transações *source*. Esta conversão é feita por uma função de transformação de transação que trabalha com os seguintes passos: detecção de conflitos, geração de transação base, geração de transação base alternada.

Na detecção de conflito, um conflito é identificado entre outras transações HiCoMo e as transações base. Se há um conflito entre transações HiCoMo, então a transação que está sendo considerada para a transformação é abortada.

Na geração de transação base, na ausência de um conflito, as transações base iniciais são geradas e executadas como subtransações no banco de dados base do servidor. O tipo de transação base depende das transações HiCoMo.

Na geração de transação base alternada é possível que algumas dessas subtransações possam violar as restrições de integridade e, então, são abortadas. Estas atualizações são tentadas novamente pela redistribuição da margem de erro. No pior caso as transações *HiCoMo* original são abortadas. Se não há violação de integridade então as transações base são confirmadas (*commit*).

3.2.3.5 Modelo de transação Moflex

O modelo de transação *Moflex* é baseado em um modelo de transação flexível. É um modelo que visa o gerenciamento da mobilidade, heterogeneidade e flexibilidade na definição e execução de transações para ambientes móveis. Este modelo permite que transações possam ser submetidas enquanto o usuário muda de células. Uma transação *Moflex* é uma coleção de subtransações que podem ser relacionadas umas as outras através de um conjunto de dependências de execução. Três tipos de dependências entre transações podem ser definidos: dependências de sucesso, dependências de falhas e dependências externas. Uma transação que possui uma dependência de sucesso em relação a outra transação ela só pode ser executada se esta outra tiver sido efetivada com sucesso. Se uma transação possui uma dependência de falha em relação a outra transação ela só pode ser executada somente depois de esta outra falhar. Quando uma transação possui uma dependência externa ela só pode ser executada quando um predicado externo (tempo, custo, ou localização) for satisfeito. As seguintes regras de mudança de células podem ser definidas: *continue*, *restart*, *resume* e *split_restart*.

A regra continue determina que a transação definida continua sua execução na nova célula. A regra *restart* determina que a transação definida é abortada na célula anterior e reiniciada na nova célula. A regra resume determina que a transação definida é dividida em duas subtransações onde a primeira subtransação é efetivada na célula anterior e a segunda subtransação continua a execução na nova célula. A regra de mudança *split_restart* determina que a

transação é dividida em duas subtransações onde a primeira subtransação é efetivada na célula anterior e a segunda subtransação é reiniciada na nova célula.

As transações que foram divididas em subtransações pelas regras de mudança de célula devem também ter uma regra de junção, usada para validar se a execução concatenada das subtransações é equivalente à execução da transação original.

3.2.3.6 Modelo de transação Kangaroo

O modelo de transação *Kangaroo* é um modelo que assimila características de mobilidade. É um modelo que faz cumprir a maioria das propriedades ACID.

Uma transação *Kangaroo* pai ou global é composta de um número de subtransações. Cada subtransação é similar a uma transação ACID que é composta de um conjunto de *reads* e *writes*. Estas subtransações são chamadas de transação *Joey* e são locais para uma estação base. Após o início de uma transação *Kangaroo* uma estação base cria uma transação *Joey* para sua execução que pode ser feita em unidades móveis. Quando estas unidades móveis migram para outra célula a estação base desta célula obtém o controle da execução desta transação.

Transações *Kangaroo* suportam execução de transação em modos compensante ou *split*. Quando uma falha ocorre em um modo compensante toda execução de transação *Joey* (precedente e seguinte) é desfeita e previamente as transações *Joey* confirmadas (*commit*) são compensadas. É difícil para o sistema identificar um modo compensante, então os usuários fornecem a entrada útil para criar transações *Joey* compensantes. O modo de execução padrão é o modo *split*.

Quando uma falha ocorre (quando uma transação *Joey* falha) num modo padrão então nenhuma nova transação local ou global é criada da transação *Kangaroo* e as transações *Joey* previamente confirmadas (*commit*) não são compensadas. Como resultado, em modo compensante, as transações *Joey* são serializáveis, mas não em modo *split*.

Uma transação *Kangaroo*, quando iniciada por uma unidade móvel, é especificada com uma identidade única. A estação base inicial imediatamente cria uma transação *Joey* com uma identidade única e torna-se responsável por sua execução. Há uma transação *Joey* por estação base. Quando a unidade móvel encontra um *handoff* (move-se para uma célula diferente) a transação *Kangaroo* é dividida em duas transações *Joey*. Assim a mobilidade de uma unidade móvel é capturada pela divisão de uma transação *Kangaroo* em múltiplas transações *Joey*. Estas transações *Joey* são executadas seqüencialmente, isto é, todas as subtransações de uma primeira transação *Joey* são executadas e confirmadas (*commit*) antes de todas subtransações de uma segunda transação *Joey*.

3.2.4 Consistência de dados em conectividade intermitente

Para reduzir custos, consumo de energia e solucionar problemas de disponibilidade e latência os clientes móveis usam a rede alternando entre os modos de operação conectado e desconectado. Além da operação desconectada, uma operação que explora a conectividade fraca também é desejável. A conectividade fraca e intermitente também é aplicada em laptops. Neste paradigma os clientes operam desconectados na maioria do tempo e, ocasionalmente, conectam em linha fixa ou retornam para seus ambientes de trabalho.

No esquema proposto para a conectividade intermitente os dados

localizados em sites fortemente conectados são agrupados formando clusters. A consistência mútua é requerida para as cópias localizadas no mesmo cluster, enquanto que níveis de inconsistência são tolerados em diferentes clusters. A interface fornecida pelo sistema de gerenciamento de banco de dados é realçada com operações mais fracas oferecendo garantias a consistências. Tais operações fracas permitem o acesso a dados localmente disponíveis (em um cluster). Leituras fracas acessam cópias limitadas inconsistentes e escritas fracas fazem atualizações condicionais. As usuais operações, chamadas estritas, também são suportadas. Elas oferecem acessos a dados consistentes e executam atualizações permanentes.

O esquema suporta a operação desconectada desde que os usuários possam operar mesmo quando desconectados através do uso de operações fracas. Em casos de conectividade fraca o uso balanceado de ambas operações, fraca e estrita, fornece melhor utilização de largura de banda, latência e custo. Em casos de conectividade forte, o uso de somente operações estritas faz o esquema reduzir para a usual semântica de uma cópia. O suporte adicional para a adaptabilidade é possível pela regulagem do grau de inconsistência entre as cópias baseada nas condições da rede.

As operações fracas oferecem um modelo (*form*) de adaptação *application-aware*. A adaptação *application-aware* se caracteriza pelo esquema de espaço entre dois modos extremos para prover adaptabilidade. Em um extremo a adaptabilidade é inteiramente responsabilidade da aplicação, isto é, não há suporte de sistema ou qualquer modo padrão de prover adaptabilidade. No outro extremo, a adaptabilidade é subordinada ao sistema de gerenciamento de banco de dados. Uma vez que, em geral, o sistema não é informado das semânticas de aplicação ele

não pode fornecer um único modelo (*form*) de adaptação adequado. As operações fracas e estritas repousam em um ponto intermediário entre estes dois extremos, servindo de *middleware* entre um sistema de banco de dados e uma aplicação. São duas ferramentas oferecidas pelo sistema de banco de dados para aplicações. A aplicação pode, a seu critério, usar transações fracas ou estritas, baseada em suas semânticas. A implementação, controle de consistência e o suporte transacional básico são as tarefas do sistema de gerenciamento de banco de dados.

Uma análise é feita em árvore hierárquica ou uma taxonomia que verifica os problemas em níveis de abstração.

3.2.5 Modelo de consistência

Os sites de um sistema distribuído são agrupados em conjuntos chamados *clusters* físicos (*p-clusters*). Isto para que sites que são fortemente conectados um ao outro pertençam ao mesmo p-cluster, enquanto que sites que são fracamente conectados um ao outro pertençam a *p-clusters* diferentes. A conectividade forte recorre à conectividade alcançada através de comunicações de baixa latência e alta largura de banda. A conectividade fraca inclui a conectividade intermitente ou de baixa largura de banda. O objetivo é suportar operação autônoma em cada cluster físico (*p-cluster*), eliminando assim a necessidade de comunicação entre *p-clusters*, já que tal comunicação *inter-cluster* pode ser cara, restritivamente lenta e ocasionalmente indisponível.

3.2.5.1 Interface de operação de banco de dados estendido

Para o crescimento da disponibilidade e a redução da comunicação

inter-cluster o acesso direto à cópias disponíveis localmente é alcançado através de operações de escrita fraca e leitura fraca. Operações fracas são locais em um cluster físico, isto é, elas acessam cópias que residem em um único cluster físico. Uma cópia ou item é dito localmente disponível em um cluster físico se existir uma cópia daquele item no cluster físico. As operações padrão de leitura e gravação são chamadas de operações de leitura estrita e gravação estrita. Para implementar esta interface dupla de banco de dados as cópias de cada item de dados são distintas em cópias *core* e cópias *quasi*. Cópias *core* são cópias que possuem valores permanentes, enquanto cópias *quasi* são cópias que possuem somente valores condicionalmente confirmados (*commit*). Quando a conectividade é restaurada os valores de cópias *quasi* e *core* de cada item de dados são conciliados para se obter um valor consistente.

Para processar operações de transação o sistema de gerenciamento de banco de dados traduz operações de itens de dados em operações em cópias destes itens. Este procedimento é formalizado por uma função de tradução. A função mapeia cada operação de leitura de um item de dados em uma quantidade de operações de leitura de cópias deste item e retorna um valor como o valor lido pela operação. Isto significa que quando a função é aplicada em uma operação de leitura ela retorna um valor ao invés de um conjunto de valores. Em particular, a função mapeia cada operação de leitura estrita em uma quantidade de operações de leitura em cópias core do item e retorna um destes valores como o valor lido pela operação. Dependendo de como cada operação de leitura fraca é traduzida dois tipos de funções de tradução podem ser definidas: uma função *best-effort* que mapeia cada operação de leitura e gravação em uma quantidade de operações de leitura em cópias (disponíveis localmente) *core* e *quasi* do item e retorna o valor

mais atual; uma função de tradução conservativa que mapeia cada operação de leitura fraca em uma quantidade de operações de leitura somente em cópias *quasi* localmente disponíveis e retorna o valor mais atual.

Baseado no tempo de propagação das atualizações das cópias *quasi* e *core*, dois tipos de funções são definidas: a função de tradução eventual que mapeia uma gravação estrita do item em escritas de cópias core; e função de tradução imediata que atualiza as cópias *quasi* que residem no mesmo no mesmo cluster físico das cópias *core* escritas.

Cabe ressaltar também que são dois os tipos de transações suportadas: fraca e estrita. Uma transação estrita não inclui qualquer operação fraca.Transações estritas são atômicas, consistentes, isoladas e duráveis. Uma transação fraca não inclui qualquer operação estrita. As transações fracas acessam cópias de dados que residem no mesmo *cluster* físico e assim são executadas localmente neste *cluster*. As transações fracas são confirmadas (*commit*) localmente no *cluster* físico em que elas são executadas. Depois da confirmação (*commit*) local suas atualizações são visíveis somente para transações fracas no mesmo cluster físico. Outras transações não são informadas destas atualizações. A confirmação (*commit*) local de transações fracas é condicional no sentido de que suas atualizações poderiam se tornar permanentes somente após a reconciliação quando as transações locais podem se tornar globalmente confirmadas (*commit*). Assim, existem dois eventos de atualização associados a cada transação fraca, uma confirmação (*commit*) condicional local em seu cluster associado e uma confirmação global implícita na reconciliação.

Outros tipos de transações que combinem operações fracas e estritas podem ser visionados, porém suas semânticas as tornam difíceis de se

definir. Ao invés disso as transações fracas e estritas podem ser vistas como unidades de transação de algum modelo de transação avançado. Após sua submissão cada transação de usuário é decomposta em um conjunto de unidades de subtransações fracas e estritas de acordo com a semântica e o grau de consistência requerido pela aplicação.

3.2.5.2 Precisão de dados

Um banco de dados é um conjunto de itens de dados, e um estado de banco de dados é definido como um mapeamento de cada item dado a um valor de seu domínio. Os itens de dados são relacionados por um número de restrições, as chamadas restrições de integridade, que expressam as relações entre seus valores. O estado do banco de dados é consistente se as restrições de integridade são satisfeitas. A manutenção da consistência em tradicionais ambientes distribuídos confia na suposição de que normalmente todos os sites estão conectados. Esta hipótese, entretanto, não é sempre válida em computação móvel, uma vez que sites distribuídos são intermitentemente conectados.

Por conta da conectividade intermitente, ao invés de requerer a manutenção de todas as restrições de integridade de um banco de dados distribuído, são introduzidos clusters lógicos como unidades de consistência. Um *cluster* lógico (*l-cluster*) é um conjunto de todas as cópias *quasi* que residem no mesmo *cluster* físico. Além disso, todas as cópias *core* constituem um único *cluster* lógico independentemente do site em que elas residem fisicamente. A consistência é relaxada no sentido que as restrições de integridade são asseguradas somente para cópias de dados que pertencem ao mesmo *cluster* lógico. Uma restrição de integridade *intra-cluster* é uma restrição de integridade que pode ser completamente

avaliada usando cópias de dados de um único *cluster* lógico. Todas as outras restrições de integridade são chamadas de restrições de integridade *inter-cluster*.

Nas restrições de integridade *inter-cluster*, a inconsistência limitada significa que todas as cópias no mesmo *cluster* lógico tem o mesmo valor enquanto que entre as cópias em diferentes *clusters* lógicos há divergência limitada. A divergência limitada é quantificada por um valor inteiro positivo chamado grau de divergência. As definições possíveis desse valor são: o número máximo de transações que operam cópias quase; a variação de valores aceitáveis que um item de dados pode ter; o número máximo de cópias por item de dados que divergem de uma cópia de item primária pré-assinalada, isto é, cópias *core*; o número máximo de itens de dados que têm cópias divergentes; o número máximo de atualizações por itens de dados que não são refletidos em todas as cópias do item.

Uma restrição de replicação para um item de dados assim limitado é chamado *d-consistente*, onde *d* é grau de divergência. O grau de divergência entre cópias pode ser regulado com base na robustez de conexão entre *clusters* físicos, por manter a divergência pequena em instâncias de alta disponibilidade de largura de banda e permitindo maior desvio em instâncias de baixa disponibilidade de largura de banda.

3.2.6 Operação de conectividade fraca

As redes de conectividade fraca são representadas por uma freqüente perda de conexão em curtos espaços de tempo. A conectividade fraca impõe várias limitações que não se fazem presentes em uma conectividade normal, obrigando assim a revisão de vários sistemas de protocolo. Uma das características da conectividade fraca é sua variação em intensidade. A meta das propostas

fundamentais para conectividade fraca é o uso ponderado de largura de banda. As propostas visam negociar a fidelidade e a redução do custo de comunicação.

Uma estação móvel pode exercer diversos papéis em bancos de dados distribuídos. No caso de submeter operações executadas no servidor, ele pode definir que as transações serão executadas uma após a outra ou que um grupo de transações formem uma unidade atômica. O controle de concorrência passa a ser mais difícil em transações distribuídas e abrangendo estações móveis e fixas. As transações que buscam o acesso a dados em estações móveis e fixas são geradoras de *overhead*. No exemplo de um protocolo pessimista de controle de concorrência que solicita que as transações cumpram bloqueios em sites múltiplos as transações poderiam permanecer paradas por muito tempo se tivessem solicitado os bloqueios em sites desconectados. *Timestamps* poderiam ocasionar um grande número de transações abortadas visto que as operações podem se propagar com retardo de redes de baixa velocidade. Para evitar estes tipos de retardos os modelos de transações *open-nested* são mais adequados.

Transações móveis em estações fixas e móveis não são definidas como unidade atômica, mas sim como um conjunto de componentes de transação relativamente independentes, alguns rodando exclusivamente nas estações móveis. Os componentes das transações podem ser confirmados sem esperar pela resposta das outras transações que fazem parte do conjunto. Como no caso da desconexão, as transações executadas na estação móvel são visíveis somente para esta estação e seu resultado é visível para as transações locais subseqüentes. Estas transações são certificadas para correções posteriores nas estações fixas. As estações fixas podem distribuir para as estações móveis informações sobre as transações confirmadas antes do evento da certificação. Esta informação pode ser

usada na redução do número de transações abortadas.

A arquitetura *Bayou* é uma arquitetura cliente-servidor flexível e ponto a ponto com um conjunto de servidores fracamente replicados e conectados um ao outro. Neste esquema uma aplicação de usuário pode ler ou escrever qualquer cópia disponível. Os servidores propagam as escritas entre si através de um processo baseado na comunicação entre pares denominado sessão de anti-entropia (*anti-entropy*). Assim como na replicação *two-tier*, cada servidor sustenta duas visões do banco de dados: uma cópia que reflete apenas os dados confirmados (*commit*) e outra cópia que também reflete as tentativas de escrita conhecidas pelo servidor. Eventualmente cada escrita é confirmada através de um esquema de *commit* primário, onde um servidor é apontado como primário e obtém a responsabilidade de confirmar as atualizações. Os servidores podem precisar desfazer o efeito de tentativas anteriores de uma operação de escrita e reaplicá-las, isso por conta da capacidade dos servidores de receber as atualizações com origem de usuários e de outros servidores em diferentes ordens. O esquema de *Bayou* possibilita a checagem de dependência para a verificação automática de conflitos e procedimentos de fusão para resolução. O esquema trabalha com sessões e não transações. Uma sessão é uma abstração de uma série de operações de leitura e escrita realizadas durante a execução de uma aplicação.

O processamento de *commit* distribuído é feito através do protocolo *Two-Phase Commit* e suas variações. Cabendo às estações móveis somente a emissão de pedidos o protocolo *Two-Phase Commit* pode ser aplicado uma vez que o controle da transação está implicitamente definido a um site da rede fixa.

Em redes móveis algumas mudanças podem ser feitas. O fluxo do protocolo *Two-Phase Commit* deve ser usado para levar algumas informações de

suporte ao esquema de controle de concorrência. O processamento de *commit* é mantido. Em uma desconexão as transações de sites móveis são tentativas de *commit*, mas, na integração, as tentativas compõem uma transação distribuída que precisa ser confirmada (*commit*). O processamento de *commit* deve levar em consideração os novos problemas que são impostos pela fraca conectividade quando há participação da estação móvel como parceira e com seus dados próprios em uma transação móvel. Os protocolos de c*ommit* precisam ser verificados por conta das longas desconexões, de conectividade intermitente, de alta latência, de largura de banda baixa, de comunicação cara e da mobilidade.

As otimizações de *commit* para a computação móvel podem ser reguladas. É mais prudente transferir a coordenação de *commit* a um parceiro mais confiável enquanto o cliente móvel pode iniciar o processamento de *commit.* Isto pode ser feito através da otimização do último agente de *commit*.

3.2.7 Mecanismos de controle de concorrência

A transação convencional possui duas características: realiza um conjunto de operações sobre uma base de dados transformando-a de um estado consistente para um outro estado consistente; e é permitido que uma transação deixe, durante sua execução, o estado do banco de dados temporariamente inconsistente desde que o item anterior seja satisfeito.

Considerando tais características a forma mais simples de implementar o controle de concorrência para transações convencionais é através da execução serial. Pela execução serial uma transação iniciaria a sua execução quando nenhuma outra transação estivesse em execução. Levando em conta a primeira característica citada pode-se concluir que uma execução serial consegue

manter a consistência do banco de dados. A execução serial garante a consistência do banco de dados, mas anula as vantagens da multiprogramação ocasionando queda de desempenho. Estudos então visaram apresentar a possibilidade da execução paralela de um conjunto de transações sobre um mesmo banco de dados. Os estudos ainda procuraram demonstrar que essa execução pode ser equivalente à execução serial do mesmo conjunto de transações (serialização). Esse caso poderia garantir a consistência do banco de dados e também poderia acabar com a restrição da execução serial. Este é o desafio dos mecanismos de controle de concorrência de transações: permitir a execução de transações em paralelo e garantir a serialização. Por isso alguns esquemas de controle de concorrência tentam evitar que uma transação acesse os resultados intermediários de uma outra transação. Dentre eles, o esquema *two-phase locking*.

Em geral, os mecanismos de controle de concorrência podem ser classificados como controles pessimistas e controles otimistas. Os controles de concorrência pessimistas evitam antecipadamente que um item de dado compartilhado se torne inconsistente graças ao acesso de transações concorrentes. Para isso, esses tipos de controle geralmente bloqueiam o acesso ao recurso quando a operação é conflitante com as operações dos demais processos concorrentes. O esquema *two-phase locking* é um controle de concorrência pessimista. Os controles de concorrência otimistas são permissivos em relação ao bloqueio de um recurso compartilhado quando acessado por um recurso compartilhado. Estes controles são ditos otimistas porque confiam na possibilidade de que o acesso concorrente não deixará inconsistente o recurso compartilhado. Como nem sempre essa hipótese é verdadeira, os mecanismos otimistas possuem meios de devolver a consistência ao recurso que se tornou inconsistente.

O controle de concorrência otimista possui três fases: fase de leitura, fase de validação e fase de escrita.

Na fase de leitura cada transação recebe uma cópia do item de dado que deseja acessar (versão de tentativa). Sobre a sua versão de tentativa uma transação possui autorização para realizar operações de leitura e de escrita. Quando há várias transações em concorrência ao mesmo item de dado, várias versões de tentativa desse mesmo item de dado podem existir. Durante essa fase cada transação só acessa as suas próprias versões de tentativa. Assim as mudanças feitas por uma transação não são visíveis a outras transações. As versões de tentativa de uma transação são ainda divididas em dois subconjuntos: um conjunto de leitura que congrega os itens de dados que são utilizados pela transação somente para operações de leitura e um conjunto de escrita que congrega aqueles itens de dados que receberão operações de escrita por parte da transação.

Depois que uma transação realizou todas as suas operações sobre suas versões de tentativa, os itens de dados originais (dos quais as versões de tentativa forma copiadas) têm de ser atualizados. Entretanto, como podem existir várias versões de um mesmo item de dado mantidas por diferentes transações faz-se necessário um processo de validação que detecte e resolva os conflitos existentes. Se uma transação for validada com sucesso ela pode efetivar suas operações em um processo de commit. Caso contrário o mecanismo de resolução de conflitos é usado ou a transação conflitante deve ser abortada.

Se a transação foi validada com sucesso os itens de dados correspondentes às versões de tentativa de transação devem ser atualizados e se tornam permanentes (armazenados em disco). Isso garante que uma transação é

finalizada com sucesso e seus dados não serão mais perdidos em decorrência de falhas.

O controle de concorrência pessimista evita de antemão a ocorrência de inconsistências geradas pelo acesso concorrente. O princípio de um controle desse tipo é bloquear o acesso ao item de dado a transações que desejam realizar operações que são conflitantes com as operações de uma transação que está usando o mesmo item de dado. Um mecanismo de controle de concorrência baseado em *lock* é o two-*phase locking*. Os dois tipos de *lock* básicos são *lock* de gravação e *lock* de leitura. Antes de uma transação realizar uma operação de leitura ou de gravação em um determinado item de dado ela tem de obter um *lock* de leitura ou gravação. Observando a relação de conflito entre *locks* é possível afirma que o *lock* de gravação é exclusivo, isto é, uma transação consegue obter um *lock* de gravação quando nenhuma outra transação possui *lock* de leitura ou *lock* de gravação sobre determinado item de dado. É possível afirmar também que duas ou mais transações podem obter simultaneamente *locks* de leitura sobre o mesmo item de dado contanto que nenhuma outra transação possua um *lock* de gravação sobre o mesmo item de dado.

No esquema *two-phase locking*, uma transação que deseja realizar uma operação em um item de dado verifica se a operação desejada é conflitante com algum *lock* que já tenha sido obtido por uma outra transação no mesmo item. Se a operação for conflitante a transação deve esperar a liberação do item. Quando não há mais conflitos com *locks* de outras transações ela pode obter seu próprio *lock* sobre o item e acessá-lo. Durante sua execução uma transação pode obter *locks* sobre vários itens de dados, mas depois de liberar um desses *locks* a transação não pode mais obter *locks*.

Esse esquema é chamado de *two-phase locking* porque possui duas fases: a fase de crescimento e a fase de decrescimento. Na fase de crescimento a transação obtém seus *locks* e na fase de decrescimento os *locks* são liberados. O fato de uma transação liberar um *lock* e não mais poder obter outros *locks* garantem a serialização no esquema *two-phase locking*.

3.3 Recuperação em bancos de dados móveis

Os protocolos de recuperação de banco de dados recuperam um banco de falhas de transação ou sistema, isto é, eles restauram o banco de dados para um estado consistente de onde o processamento da transação pode reiniciar. Estas falhas podem ocorrer devido a algumas razões tais como erro de endereçamento, entrada incorreta, falha de memória *RAM*, etc. Em um ambiente de execuções concorrentes quando uma falha ocorre então uma transação pode ser ativada, bloqueada, retornada (*roll back*) ou confirmada ao meio. A tarefa de um protocolo de recuperação é identificar a operação correta para a recuperação de cada transação. Estas operações são *roll forward* ou *redo* e *roll back* ou *undo*. Dependendo do estado da execução de uma transação uma destas operações é selecionada. Assim, em um processo de recuperação algumas transações são desfeitas e algumas são refeitas. Para implementar essas operações, um l*og* de transação é gerado e mantido pelo sistema é solicitado. O *log* contém valores confirmados de itens de dados (*Before Image ou* BFIM) e valores de itens de dados modificados (*After Image*). O *log* é um documento crucial para a recuperação, então ele é gerado e mantido por um protocolo chamado *Write Ahead Logging* (WAL*)*. O protocolo garante que os conteúdos de um log são confiáveis e podem ser usados para operações *Undo* e *Redo*. Depois de uma falha o sistema de banco de dados

reinicia e, pelo uso de *log*, aplica as operações *Redo* e *Undo* em transações que estavam no sistema quando ele falhou. Um *Redo* completa a operação de *commit* para uma transação e um *Undo* volta uma transação para manter a atomicidade. Estas operações nos concedem quatro diferentes protocolos de recuperação: *Undo-Redo*, *Undo-NoRedo*, *NoUndo-Redo* e *NoUndo-NoRedo*.

O protocolo *Undo-Redo* aplica o *Redo* e o Undo para recuperar os sistemas de banco de dados. Isso significa que durante a execução de transação ele pode escrever no banco de dados valores intermediários nos seu item de dados. Se a transação estava ativa quando o sistema falhou, então a transação é desfeita (*Undo*) e ela é refeita (*Redo*) se a transação estava pronta para o *commit*.

O protocolo *Undo-NoRedo* não suporta *Redo* e recupera o banco de dados aplicando somente a operação *Undo*. Isso significa que o sistema força atualizações intermediárias de transações para o banco de dados imediatamente.

O protocolo *NoUndo* - *Redo* certifica que nenhum dos resultados intermediários da transação são instalados no banco de dados. Assim, se uma transação não pode ser refeita (*redo*) no momento da recuperação então ela é removida do sistema.

O protocolo *NoUndo* - *NoRedo* não aplica *Redo* e *Undo* e recupera o banco de dados usando a cópia *shadow* de itens de dados. Assim, durante a execução uma transação cria uma cópia dos itens de dados que ela modifica. Durante a recuperação ela usa cópias *shadow* e atuais de um item de dados para selecionar a versão correta a ser instalada no banco de dados.

A recuperação é uma operação consumidora de tempo e de recursos e seus protocolos requerem muito de ambos. A operação mais cara é o gerenciamento de *log*. Esta operação é essencial para a recuperação, então para

um Sistema de Banco de Dados Móveis é necessário um esquema econômico e eficiente para seu gerenciamento.

Um sistema de banco de dados móvel é um sistema distribuído baseado no paradigma cliente-servidor, mas com funções diferentes dos sistemas centralizados ou distribuídos convencionais. Ele alcança tais funcionalidades variadas por impor comparativamente mais restrições e demandas na infra-estrutura do Sistema de banco de dados móvel (SBDM). Para gerenciar as funções de nível de sistema, o SBDM pode solicitar diferentes esquemas de gerenciamento de transação (controle de concorrência, recuperação de bancos de dados e aplicação, processamento de consultas, etc), diferentes esquemas de *logging* e *caching*, e assim por diante.

Em qualquer SGBD, distribuído ou centralizado, o banco de dados é recuperado de uma maneira similar e o módulo de recuperação e uma parte essencial do sistema de banco de dados. Os protocolos de recuperação de banco de dados, então, não são ocupados com aplicações em nível de usuário. Um sistema que executa aplicações, além do protocolo de recuperação do banco de dados, requer um eficiente esquema para recuperação de aplicações. A recuperação de aplicação, diferente da recuperação do banco de dados, eleva a disponibilidade da aplicação por recuperar o estado de execução das aplicações. Por exemplo, em SBDM ou em qualquer sistema distribuído um número de atividades relacionadas à execução de transações, tais como a chegada da transação em um cliente ou em um servidor, a fragmentação da transação e a distribuição destes fragmentos para nós relevantes para execução, a remessa de atualizações feitas em clientes para o servidor, a migração de uma unidade móvel para outra célula (*handoff*), etc., têm de ser logada para recuperar o ultimo estado

da execução. Com a ajuda do *log* o módulo de recuperação da aplicação recria o último estado da aplicação de onde a execução normal reinicia.

A recuperação da aplicação é relativamente mais complexa que a recuperação do banco de dados por causa do grande número de aplicações solicitadas para gerenciar o processamento de dados, pela presença de múltiplos estados da aplicação e por causa da ausência da noção do “último estado consistente”. Isto se torna mais complexo em SBDM por causa das demandas de processamento único de unidades móveis, pela existência de *handoffs* aleatórios e por causa da presença de operações em modos conectado, desconectado e de conexão intermitente, pelo *logging* dependente de localização e por causa da presença de diferentes tipos de falhas. Estas falhas podem ser categorizadas como falhas de hardware ou de software. Falhas de hardware incluem perda de unidade móvel, que não podem ser facilmente reparadas. Falhas de software incluem falha de sistema (falha do programa, erros de endereçamento, bateria fora de operação, unidade de processamento desligada, etc.) e são recuperáveis.

Uma aplicação pode estar em qualquer estado de execução (bloqueada, executando, recebendo dados lentamente, e assim por diante). Além disso, a aplicação pode estar sob execução em unidades estacionárias (estação base ou servidor de banco de dados), em unidades móveis ou em ambos. Estas unidades de processamento, especialmente a unidade móvel, podem estar passando por um *handoff*, desconectada, em modo cochilo (*doze mode*) ou completamente desligada. A aplicação pode estar processando uma *mobilaction*, lendo alguns dados ou confirmando (*commiting*) um fragmento, e assim por diante. Se uma falha ocorre durante qualquer uma destas tarefas o sistema de recuperação deve trazer a execução da aplicação de volta ao ponto de reinício.

Em recuperação de aplicações, diferente da consistência de dados, a questão da consistência da aplicação não aparece porque a aplicação não executa corretamente na presença de qualquer erro. Assim, a tarefa mais importante para facilitar a recuperação da aplicação é o gerenciamento de *log*. Os protocolos de recuperação de banco de dados concedem um esquema de *logging* altamente eficiente e confiável, infelizmente, mesmo com modificações, o esquema de *logging* convencional imporia cargas intratáveis em SBDM restritos de recursos. O que é necessário é um esquema de *logging* eficiente, que armazene, recupere e unifique os fragmentos *log* de aplicação para a recuperação em restrições de sistemas de banco de dados móveis.

3.3.1 Gerenciamento de log

O *Log* é um arquivo seqüencial onde informações necessárias para recuperação são registradas. Cada registro de *log* representa uma unidade de informação. A posição de um registro no *log* identifica a ordem relativa da ocorrência do evento que o registro representa. Em sistemas legados (centralizados ou distribuídos) o *log* reside em localizações fixas que sobrevivem a *crashes* de sistema. Ele é resgatado e processado para facilitar a recuperação do sistema de qualquer tipo de falha. Essa propriedade de persistência do *log* é alcançada através do protocolo WAL. Esta propriedade estática do *log* assegura que nenhuma outra operação além do seu acesso é solicitada para processá-lo para recuperação. A situação muda completamente nos sistemas que suportam mobilidade terminal e pessoal por permitir as unidades de processamento de mover-se ao redor. Como resultado eles conectam e desconectam algumas vezes durante a atividade de execução inteira das transações que eles processam. O *logging* se torna complexo

porque o sistema deve seguir o protocolo WAL enquanto o *logging* registra em vários servidores.

Um esquema de recuperação eficiente para SBDM exige que o gerenciamento de *log* deve consumir um mínimo de recursos de sistema e deve recriar o ambiente de execução assim que possível depois da unidade móvel reiniciar (*reboot*). As unidades móveis e os servidores devem construir um *log* de eventos que muda os estados de execução da *mobilaction*. As mensagens que mudam os conteúdos de *log* são chamadas de eventos de escrita/gravação (*write events*). Os exatos eventos de gravação dependem do tipo de aplicação. Em geral, uma unidade móvel registra eventos como a chegada de uma *mobilaction*, a fragmentação da *mobilaction*, o histórico da mobilidade da unidade móvel (*handoffs*, estado corrente do *log*, localização de armazenamento, etc) e a remessa de atualizações da *mobilaction* para os sistemas de bancos de dados. Os sistemas de banco de dados devem registrar eventos similares além dos eventos ao *commit* da *mobilaction*.

Os esquemas que provem a recuperação nos PCSs salvam o *log* na estação base onde a unidade móvel atualmente reside. É importante notar que o gerenciamento de *log* para falha de PCS é relativamente fácil porque não suporta processamento de transação. De qualquer forma, o conceito pode ser usado para desenvolver esquemas eficientes de *logging* para SBDM.

Existem três lugares onde o *log* pode ser salvo: no MSC, na Estação Base ou na Unidade Móvel. A confiabilidade e disponibilidade das unidades móveis, entretanto, fazem deste local o menos desejável para se salvar o *log*. O MSC e a Estação Base são locais adequados, mas do ponto de vista de custo e gerenciamento, o MSC não é uma localização conveniente. O MSC pode controlar

um grande número de Estações Base e, no caso de uma falha, acessar e processar o *log* para transação específica pode consumir tempo. Um MSC não é diretamente conectado aos servidores de banco de dados, que fornecem aplicações de gerenciamento de *log* necessárias. As Estações Base, por outro lado, são diretamente conectadas ao sistema de banco de dados e também às Unidades Móveis. Então, dos aspectos de conectividade e disponibilidade, as Estações Base são comparativamente melhores candidatas para salvar um log de aplicação. Sob essa configuração uma unidade móvel pode salvar o *log* na Estação Base corrente e a Estação Base pode arquivá-lo nos Sistemas de Banco de Dados.

Em sistemas de banco de dados convencionais, a geração de log e sua manipulação são predefinidas e fixas. Em um ambiente móvel isto nem sempre ocorre por causa dos freqüentes movimentos e desconexões das unidades móveis. Uma *mobilaction* pode ser executada por uma combinação de unidades móveis, estações bases e *hosts* fixos. Além disso, se ocorrer de um fragmento da *mobilaction* visitar mais de uma unidade móvel então seu *log* pode ser disseminado para mais de uma estação base. Isto implica que o processo de recuperação pode precisar de um mecanismo para a unificação de *log* (ligação lógica de todas a porções de *log*). Os possíveis esquemas de *logging* podem ser categorizados como: *Logging-saving* centralizado de *log* em um site designado e *Home Logging*.

Sob o primeiro esquema uma estação base é designada como site *logging* onde todas as unidades móveis de todas as regiões de dados salvam seus *logs*. Uma vez que a localização de *logging* é fixada e antecipadamente conhecida, e o *log* inteiro é armazenado em um lugar, seu gerenciamento (acesso, deleção, etc) torna-se mais fácil. Sob este esquema cada unidade móvel gera o *log* localmente e, em intervalos adequados ou quando uma condição predefinida existe,

copia seu *log* local para a estação base de *logging*. Se um fragmento ou *mobilaction* falha, então o gerenciador de recuperação local adquire o *log* da estação base e recupera a *mobilaction*. Este esquema funciona, mas ele tem limitações. Ele tem confiabilidade muito baixa. Se a estação base de *logging* falha, então ela vai parar o processo de *logging* inteiro; conseqüentemente, o processamento de transação vai parar até a estação base recuperar. Adicionar outra estação base backup não aumentará o custo de recursos mas aumentará o custo de gerenciamento de *log* também. Além dessas limitações, o *logging* pode se tornar um gargalo. O tráfego de *logging* na estação base de *logging* pode se tornar intratavelmente pesado, causando significantes *delays* de *logging*. Para um sistema levemente carregado com pequenos movimentos de unidades móveis, entretanto, este esquema concede um modo simples e eficiente de gerenciamento de *log*.

No esquema *Home Logging* cada unidade móvel armazena seu *log* em uma estação base que ela inicialmente registra. Contudo uma unidade móvel rodará em volta do domínio geográfico livremente e continua a acessar dados de qualquer sites, todo *logging* ainda estará em sua estação base. Este esquema também possui limitações. Sob esse esquema o *log* inteiro da *mobilaction* pode ser disseminado por um número de estações base se seus fragmentos forem processados por unidades móveis com estações bases diferentes. Para recuperar a *mobilaction*, todas as partes do *log* exigirão ligação (lógica). Considerando uma requisição dependente de localização que chega a uma unidade móvel para processar, mas de quem a estação base não é a única que armazena os dados dependentes de localização. Isto pode ocorrer se um viajante do Kansas emite uma requisição de dados em sua unidade móvel para *Dallas Holiday Inn*. Este esquema pode causar um tráfego de mensagem excessivo. Uma vez que a localização de

logging não é distribuída, ele tem fraca disponibilidade e excessivo tráfego de mensagem durante a execução de transação.

Em uma estação base designada, nesse esquema uma unidade móvel constitui localmente o *log* e, em alguns intervalos predefinidos, salva-o na estação base designada. No momento de salvar o *log*, uma unidade móvel pode estar na célula da estação base designada ou em uma estação base remota. Neste último caso, o *log* deve viajar através de uma cadeia de estações base, terminando na estação base designada. Isso funcionará enquanto não houver falha de comunicação em qualquer lugar da cadeia de estações base.

Em todas as estações base visitadas, nesse esquema uma unidade móvel salva o *log* na estação base da célula que ele está visitando atualmente. O *log* de aplicação inteira é armazenado em estações base múltiplas, e no momento de recuperação de todas as porções de *log* são unificadas para criar o *log* completo. É possível que duas ou mais porções do *log* inteiro podem ser armazenadas em uma estação base se a unidade móvel volta a visitar a estação.

Alguns esquemas e *logging* foram desenvolvidos sob estas duas abordagens anteriores, dentre eles o esquema preguiçoso (*lazy scheme*) e o esquema pessimista (*pessimistic scheme*).

No esquema preguiçoso os *logs* são armazenados na estação base corrente e se a unidade móvel se more para uma nova estação base, um ponteiro para a antiga estação base é armazenada na nova estação base. Esses ponteiros são usados para unificar o *log* distribuído por diversas estações base. Este esquema tem a vantagem de ele incorrer relativamente menos a overhead de rede durante *handoff* como nenhuma informação de *log* precisa ser transferida. Infelizmente, este esquema tem um grande tempo de recuperação porque ele

requer a unificação de porções de *log*. A unificação de *log* pode ser executada em dois modos: esquema baseado em distância e esquema baseado em freqüência. Em um esquema baseado em distância a unificação de *log* é iniciada assim que unidade móvel cobre a distância predefinida. Esta distância pode ser medida em medida de estação base visitada ou em medida de sites de células visitados. No esquema baseado em freqüência, a unificação de *log* é executada quando o número de *handoffs* suportados pela unidade móvel cresce além de um valor predefinido. Depois de unificar o *log* a distância ou contador de *handoff* é resetada.

No esquema pessimista, o *log* inteiro é transferido para cada *handoff* da antiga para a nova estação base. Este esquema, então, combina o *logging* e a unificação do *log*. Conseqüentemente, a recuperação é rápida, mas cada *handoff* requer um grande volume de transferência de dados.

O *framework* de rede móvel existente não é eficiente para transações de banco de dados *full-fledge* rodando em servidores de banco de dados e unidades móveis. Nos esquemas acima a mudança de localização da unidade móvel tem de ser atualizada pelos servidores de bancos de dados, que poderia ser uma grande desvantagem. Para superar isso, o IP móvel foi introduzido. Na recuperação baseada na arquitetura de IP móvel as estações base armazenam o *log* atual, a informação de *checkpoint* e a estação base ou o agente *home* que mantém a informação de recuperação como a unidade móvel transversa. Este esquema tem a vantagem de o gerenciamento de *log* ser fácil e os servidores de banco de dados não precisam ser referenciados com a atualização da localização da unidade móvel, mas ele sofre quando a unidade móvel está longe de casa. Conseqüentemente, a recuperação é provavelmente lenta se o agente *home* está longe da unidade móvel. O outro problema de se usar IP móvel é o roteamento

triangular onde todas as mensagens do servidor de banco de dados para a unidade móvel têm de ser roteadas através do agente *home*. Isto invariavelmente impede a execução da aplicação. Os esquemas discutidos até agora não consideram o caso onde uma unidade móvel recupera um uma estação base diferente de uma em que ela colidiu. Em tal cenário, a nova estação base não tem a informação da estação base prévia no seu *Visitor Location Register* ou Registro de Localização de Visitante (VLR), e ela tem que acessar o *Home Location Register* ou Registro de Localização de *Home* (HLR) para ter essa informação, que é necessária para ter a recuperação do *log*. O acesso do HLR aumenta o tempo de recuperação significantemente se ele está armazenado longe da unidade móvel. Uma desvantagem similar pode ser observada no esquema de IP móvel, onde a unidade móvel precisa contatar o agente *home* toda vez que precisar de recuperação.

3.3.2 Esquemas de recuperação

Os esquemas de recuperação em banco de dados móveis têm abordagens diferentes; entretanto eles fazem seus esquemas na mesma plataforma de banco de dados móveis. A plataforma contém um grupo de unidades móveis e estações base. Estas unidades salvam *logs* e *checkpoint* e têm certeza que a informação necessária está disponível para recuperação de falhas eficientemente e economicamente.

3.3.2.1 Esquema de recuperação híbrida de três fases

Um esquema de *checkpoint* e recuperação de três fases combina esquemas de *checkpoint* e comunicação coordenada e induzida. Todas estações

base usam *checkpoint* coordenado e o *checkpoint* de comunicação baseada é usado entre unidades móveis e estações base. Os passos seguintes descrevem brevemente o trabalho do algoritmo. O algoritmo usa as unidades móveis MU_1, MU_2, MU_3 e MU_4, assim como as estações base MSS_1, MSS_2 e MSS_3 para descrever o tráfego da mensagem.

Inicialmente, um coordenador (estação base) MSS_1 envia a todos um pedido de mensagem com um índice de *checkpooint* para MSS_2 e MSS_3.Cada *MSS* (estação base) atualiza um timer T_{lazy}. Elas usam um esquema de coordenação preguiçosa para reduzir o número de mensagens, assim, ele é especialmente adequado para sistemas de banco de dados móveis. Nessa abordagem raros *snapshots* são tomados os quais só ocasionalmente impõem alto *overheads* de *checkpoint* de *snapshots* coordenados na rede de banda baixa conectando todas as unidades móveis. Esta abordagem também previne snapshot global de ficar desatualizado, como resultado, o total de computação para recuperação de falha é minimizado.As unidades móveis MU_2 e MU_3 não importa qual está ativa, pega um *checkpoint* antes da mensagem *m2* ou *m3* chegar de MSS_2 e MSS_3 durante T_{lazy}. MU_1 ou MU_4 pega um *checkpoint* quando T_{lazy} estiver expirada, e ela recebe um pedido de *checkpoint* de MSS_1 ou MSS_3. MSS_2 e MSS_3 respondem (mandam uma mensagem de resposta) para MSS_1. MSS_1 envia uma mensagem de *commit* para todas as estações base (*MSSs*) depois de receber mensagens de reposta das outras estações base. MU_3 migra de MSS_3 para MSS_2 e envia uma mensagem para acordar MU_4 se ela estiver em modo cochilo. MU_2 pega um *checkpoint* antes de ela desconectar-se da rede. Se a MU_2 já está em modo desconectado, então ela na pega qualquer *checkpoint*. No caso de MU_1 falhar, ela para de executar e envia uma mensagem de recuperação para MSS_1. MSS_1 envia para todas as estações base

(*MSSs*) mensagens de recuperação. Cada estação base envia mensagem de recuperação para todas suas unidades móveis. Estas unidades móveis retornam (*rollback*) para seus últimos estados consistentes.

3.3.2.2 Recuperação de falhas e checkpointing de baixo custo

Um esquema de coleção de snapshot síncrono de baixo custo é aqui apresentado onde se permite interferência mínima para a computação básica. Os nós de processamento são representados por P_0, P_1,P_2, P_3 e m_1, m_2, m_3, m_4, m_5 e m_6 representam as mensagens.

O nó P_1 primeiro coleta *snapshots* locais no ponto X tempo de ponto. Assuma que os nós P_1, P_2, P_3 são dependentes, então uma mensagem de pedido de snapshot é enviada para P_1 e P_3 por P_2. O nó P_3 envia a mensagem m_4 para o nó P_1 para depois de pegar seu próprio snapshot. Há duas possibilidades quando a mensagem m_4 alcança P_1: P_1 não tem processado qualquer mensagem a desde seu último *shapshot* ou P_1 já tem processando uma mensagem de qualquer nó desde seu último snapshot. Neste exemplo, já que P_1 não tem processado qualquer mensagem, como resultado ele tira sua tentativa de *snapshot* e registra esse evento antes de processar a mensagem m_4. Ele então propaga o *snapshot*. O nó P_0 tira um snapshot local já que ele não tem recebido qualquer mensagem de qualquer nó e envia uma mensagem m_5 para P_1. Quando m_5 alcança P_1, ele descobre que m_5 não é uma nova mensagem para forçar um snapshot então P_1 não tira um snapshot. Quando um nó P_i falha, então ele volta (*roll back*) para seu último *checkpoint* envia pedidos de *rollback* para um subgrupo de nós. Quando um nó P_j recebe sua primeira mensagem de *rollback*, então ele retorna para seu último *checkpoint* e envia um pedido de *rollback* para um seletivo grupo de nós. O nó P_j pode receber

mensagens subseqüentes de *rollback* como um resultado de falha de P_i, mas ele ignora todas elas. No caso de unidades móveis, todos seus pedidos de *rollback* são roteados através de suas estações base.

3.3.2.3 Gerenciamento de log baseado em agentes móveis

Agentes móveis têm sido usados com sucesso para gerenciar uma quantidade de aplicações e atividades de sistema. Eles também estão sendo usados para desenvolver um esquema para gerenciar *log* de aplicação em sistemas de banco de dados móveis. Um agente móvel é um programa autônomo que pode se mover de uma máquina para outra em redes heterogêneas sob seu próprio controle. Eles podem suspender sua execução em qualquer ponto, transportar-se para uma nova máquina e retomar a execução do ponto que parou. Um agente leva ambos o código e o estado da aplicação. Atualmente um paradigma de agente móvel é uma extensão da arquitetura cliente/servidor com mobilidade de código. Algumas das vantagens dos agentes móveis são: encapsulamento de protocolo, robustez e tolerância a falhas, assincronia e execução autônoma.

No encapsulamento de protocolo agentes móveis podem incorporar seus próprios protocolos em seus códigos ao invés de depender do código legado fornecido pelos *hosts*. Sobre a robustez e a tolerância a falhas cabe ressaltar que quando as falhas são detectadas os sistemas de *host* podem facilmente despachar agentes para outros *hosts*. Esta habilidade faz os agentes tolerantes a falhas. No caso da assincronia e execução autônoma, uma vez que agentes são despachados de um *host* eles podem tomar decisões independentemente. Isso é particularmente útil para ambientes *wireless* onde manter uma conexão do começo ao fim e executar *mobilaction* pode não ser econômica ou necessária. Em tais casos os

agentes podem visitar o destino, executar qualquer processamento requerido e trazer os dados finais para a origem desse modo removendo a necessidade de uma conexão *wireless* contínua. Um agente pode, por exemplo, pegar uma *mobilaction* de uma unidade móvel, executá-la no mais adequando nó (poderia ser remoto) e trazer o resultado de volta à unidade móvel. Agentes tem desvantagens, que poderiam ser afetar o esquema de *logging*, sua alta migração e overhead de máquina. Seu overhead deve ser minimizado para melhorar sua performance. O esquema usa serviços de agentes com a abordagem *only when needed*. Não é possível desenvolver um esquema que otimize a performance em todos os níveis e em todas as diferentes situações. Por esta razão alguns esquemas de recuperação melhoram a performance pelo objetivo de minimizar o overhead de comunicação, alguns podem concentrar no tempo de recuperação total, alguns podem otimizar o espaço de armazenamento, e assim por diante. Assim, cada esquema envolve certos *tradeoffs*. Quando estas questões são levadas em consideração torna-se necessário construir um *framework* que suporta a implementação dos esquemas existentes e poderia também ser capaz de suportar qualquer novo esquema. O *framework* poderia suportar a ativação/desativação de um esquema, dependendo do ambiente particular em que ele oferece melhor performance. Assim um *framework* poderia abstrair o software núcleo da estação base (que controla o registro, *handoff*, atividades, etc.) pelo controle de procedimentos de recuperação, assim permitindo para a melhor recuperação protocolos para serem implementados sem a necessidade de mudar o software núcleo. O *framework* pode também suportar um rápido desenvolvimento do código de recuperação sem muita intervenção humana. Em sistemas de banco de dados móveis o módulo coordenador reside na estação base. Ele divide a *mobilaction* em fragmentos se

necessário, e ele envia alguns para um grupo de servidores de bancos de dados. Este requerimento pede por inteligência específica para ser embutida no código da estação base. Uma *mobilaction* iniciada pela unidade móvel pode usar diferentes tipos de protocolos de *commit* como o *commit* de duas fases, *commit* de três fases ou *Transation Commit on Timeout* (TCOT). O módulo coordenador precisa suportar todos estes. Se um certo módulo em uma estação base não suporta um protocolo particular, então ele poderia ser um modo fácil de acessar um certo código. Uma ampliação disto é que quando um novo protocolo eficiente é introduzido todas as estações base poderiam ser aptas para atualizar isso facilmente e assim que possível com pequena ou nenhuma intervenção humana. Da perspectiva da recuperação do *log* da unidade móvel, uma arquitetura que suporta *logging* inteligente é exigida e é capaz de incorporar quaisquer desenvolvimentos futuros sem dificuldades. Alguns esquemas de recuperação especificam que os *logs* se movem entre as unidades móveis através de uma multidão de estações base. As novas estações base poderiam ser aptas de controlar os *logs* do mesmo modo que uma prévia fez ou poderia resultar em inconsistência de *log*. É demonstrado que a flexibilidade e restrições mencionadas acima poderiam ser incorporadas com sucesso em uma arquitetura baseada em agentes móveis sob a qual o código necessário para recuperação e coordenação pode ser incorporado nos agentes móveis. O coordenador pode ser modelado como um agente móvel e pode ser iniciado pela própria unidade móvel se necessário. Se durante um *handoff* a nova estação base não suporta um esquema específico de *logging*, então o agente na estação base prévia que suportar isso pode clonar-se e a nova réplica pode migrar para a estação base corrente sem qualquer intervenção manual. A mesma técnica pode ser usada rapidamente povoando as estações base com alguns novos

protocolos. O agente móvel com o novo protocolo embutido nele pode ser introduzido em qualquer estação base e ele pode replicar e migrar para outra estação base.

3.3.2.4 Arquitetura de logging baseado em agentes

Esta arquitetura suporta os mecanismos de *logging* independentes. É assumido que cada estação base suporta a funcionalidade dos agentes móveis. Os componentes principais da arquitetura são: agente bootstrap, agente base, agente home, agente coordenador, agente evento e agente driver.

O agente *bootstrap* controla a falha da estação base. Qualquer agente que deseja recuperar deve registrar com o agente *bootstrap*. A estação base inicia com o agente *bootstrap*. Uma vez carregado este agente inicia todos os agentes que têm registrado com ele. Estes agentes têm a capacidade de ler a informação do *log* que eles tem criado e agir adequadamente. A necessidade de tal agente pode ser neutralizada se o agente móvel fornece uma renovação automática dos agentes com seus estados intactos.

O agente base decide qual o esquema de *logging* para usar no ambiente corrente. Tal funcionalidade pode ser decidida pelo pela própria inteligência ou pode ser dada como uma entrada. Para cada unidade móvel ela cria uma instância de um agente que controla a recuperação de *mobilactions* baseada no esquema de *logging* relevante.

O agente *home* controla *mobilactions* para cada unidade móvel. Ele é responsável por sustentar o *log* e a recuperação da informação em nome da unidade móvel. A unidade móvel envia os eventos de *log* para este agente que é responsável por armazená-los na estável estação base. O agente *home* é uma

interface de estação base para a unidade móvel para *mobilactions*.

O agente coordenador reside na estação base e age como coordenador para todas as *mobilactions*.

Além do *framework*, a estação base fornece agentes móveis com uma interface para os vários eventos tomando lugar como registro de uma unidade móvel, falha de uma unidade móvel, *handoff* de uma unidade móvel, etc. Esta abordagem abstrai de longe as funções de estação base núcleo do suporte de recuperação da aplicação. Quando uma unidade móvel sofre *handoff* seu agente *home* deve saber sobre ele e então ela pode executar operações requeridas. O agente evento é a interface da estação base para o *framework* do agente para disseminação de tais informações.

Estes agentes colaboram um com o outro para facilitar o gerenciamento de *log*. A análise da interação de agente coordenador e agente *home* apresenta algumas observações. Uma unidade móvel envia *mobilaction* para seu agente móvel que a remete para agente coordenador correspondente. Se o agente coordenador precisa contatar a unidade móvel ele o faz então através do agente *home* correspondente à unidade móvel. Quando o agente coordenador envia um evento *write* para o agente *home* ele o armazena em seu local adequado antes de enviá-lo para a unidade móvel. Similarmente se quaisquer eventos chegarem na unidade móvel através da entrada de usuário a unidade móvel envia as mensagens de log correspondentes para o agente *home*

Vale ressaltar também a ação dos agentes quando ocorre *handoff*. O agente *home* se move entre as unidades móveis para a nova estação base em um *handoff*. Baseado em esquemas como Preguiçoso e Baseado em Freqüência o agente pode ou não levar os *logs* armazenados com ele para a nova estação base.

Quando ocorre um *handoff*, um agente *driver* é enviado para frente com a informação de *log* necessária para a nova estação base ao invés do agente *home* completo com toda sua inteligência para a unificação de *log*. O agente *driver* tem um código muito leve cuja função principal é para ver se o código para o agente home está presente na nova estação base. Se estiver então ele solicita o agente base residente na nova estação base para criar uma instância do agente *home* para a unidade móvel. Se qualquer código compatível não está presente então o agente *driver* envia um pedido para o agente base da estação base prévia que clona o *home* agente necessário e envia a cópia para a nova estação base. Quando uma unidade móvel se move para fora da estação base sua informação de *log* não é deletada automaticamente, ela é armazenada exceto quando notificada pelo agente da unidade móvel. Isto facilita a unificação de *logs* quando *logs* são distribuídos para um grupo de estações base.

3.3.2.5 Estratégia forward

Todos os esquemas vistos anteriormente têm assumido a recuperação imediata da unidade móvel depois de uma falha. O que se reconhece é a possibilidade da unidade móvel poder quebrar em uma estação base e recuperar em outra estação base. Um intervalo de tempo é definido entre a falha da unidade móvel e seu *reboot* subseqüente. Esse intervalo é o *Expected Failure Time* (EFT). Este esquema se concentra em tais cenários onde o EFT não é tão insignificante. A estação base detecta a falha de uma unidade móvel e agentes não trabalham em parte alguma em tal detecção. Por exemplo, se a comunicação entre duas unidades móveis quebra porque causa da falha de uma das unidades então a estação base correspondente saberá imediatamente sobre esse evento. Similarmente, a estação

base também sabe que a unidade móvel tem executado o registro de *power-down*, passado por um *handoff*, e assim por diante.Uma estação base também pagina continuamente suas unidades móveis. Se a unidade móvel sofre um *handoff* então a comunicação com a última estação base não é quebrada até a conexão com a nova estação base estar estabilizada (*soft handoff*). Estas características permitem ao SBDMs detectar uma falha da unidade móvel. Assim, enquanto uma unidade móvel está executando seu fragmento seu estado é continuamente monitorado pela estação base e qualquer mudança na situação da unidade móvel é imediatamente capturada pela interface do agente evento. Uma vez que essa detecção é dependente de sistema o EFT tende a ser um valor aproximado. A detecção pode ser transmitida para o agente *home* de vários modos. O SBDMs pode fornecer uma interface que permitiria aos agentes esperar por um evento. Outra abordagem deveria ser para fornecer uma variável de sistema de agente legível que poderia ser colocada em qualquer evento. O agente periodicamente apura a variável para checar se ela está fixada. Ambas abordagens são possíveis e fáceis de implementar em linguagens tais como Java em que alguns sistemas agentes como *IBM´*s *Aglets* e *General Magic´*s *Odissey* tem sido desenvolvidos. Já que o *handoff* não ocorre nos casos acima a nova estação base não sabe a localização da antiga estação base. Esta situação conduz a nova estação base a contatar o HLR para a estação base prévia. Isso poderia ser um impedimento para a recuperação rápida se o HLR estiver longe da estação base requisitante. Na realidade o VLR é primeiro requisitado para a informação de estação base prévia que é armazenada em VLR se ambas estações base caírem sob o controle do mesmo VLR. Se as estações base estão sob diferentes VLRs então a HLR da unidade móvel tem de ser requerida. Tal informação é armazenada no HLR quando uma unidade móvel

primeiro registra a estação base.

No esquema preguiçoso a estação base inicia estabelecendo o *log* imediatamente sobre a falha da unidade móvel. Nos esquemas apresentados a unidade móvel explicitamente emite um chamado de recuperação para a estação base e a estação base começa a unificação de *log*. Isso provoca certas questões no evento de *crash* e recuperação da unidade móvel em uma estação base diferente. Se o *log* está para ser unificado imediatamente após a falha, então ele poderia ser necessário para a nova estação base para esperar a antiga estação base para finalizar sua unificação e então apresentar seu *log*. Se o tempo de falha é grande ou o tamanho total de *log* é pequeno então a unificação será além do tempo da nova estação base requisitar a estação base prévia. Em tal caso, a recuperação pode ser rápida. No caso de uma EFT relativamente pequena ou um tamanho grande de *log* (para ser unificado) a nova estação base deve primeiro esperar pela unificação e então pela transferência de *log* atual. Isto resulta em crescimento do tempo de recuperação e custo de rede. Em tais casos poderia ser preferível a unificação do *log* ser feita na nova estação base se a lista de estações base onde o *log* é distribuído é conhecida. Na abordagem onde o *log* é unificado depois de uma chamada de recuperação, o tempo de recuperação poderia não ser pequeno suficiente se o tamanho de *log* a ser unificado é pequeno. Nesse caso a unificação tem de começar depois de obter a lista de estações bases envolvidas à estação base prévia. Também, se a unidade móvel não tem migrado para uma nova estação base antes da recuperação então o *log* tem de ser unificado, o que possivelmente aumenta o tempo de recuperação.

3.3.2.6 Reduzindo o tempo de recuperação

O esquema de unificação do *log* é baseado no número de *handoffs* ocorridos desde a última unificação de *log* ou o início da transação seja qual for mais tarde. O *log* é unificado periodicamente quando o número de *handoffs* ocorridos passa um *handoff-threshold* predefinido. Quando um *handoff* ocorre, a informação Trace é transferida da estação base antiga para a nova estação base. Esta informação trace é uma lista ordenada de elementos concedendo informações sobre as estações base envolvidas no armazenamento do *log* da unidade móvel. Cada elemento *array* consiste de dois valores: a identificação desta estação base (BS-ID) e o tamanho do *log* armazenado em *Log-Size*. Quando um *handoff* ocorre então o BS-ID na nova estação base e um valor zero de *Log-Size* são adicionados ao fim do trace. O valor de *Log-Size* é atualizado sempre que a unidade móvel presenteia a estação base com alguma informação de *log*. Os parâmetros opcionais podem também ser apresentados na informação de trace. Já que o trace não contém os atuais conteúdos de *log* e é, sobretudo, um *array* de identidades de estações bases e tamanhos de *log*, ele não apresenta um overhead significante durante o *handoff*. O esquema também assume a presença do valor de EFT que pode ser armazenado como um acessível atributo de ambiente para o agente *home* da unidade móvel na estação base. Se tal suporte não pode ser oferecido pelo sistema então o agente *home* pode também estimar o EFT das atividades da unidade móvel. Se o agente estima o EFT então este valor também é armazenado na informação de trace. Quando o sistema detecta a falha da unidade móvel ele informa o *framework* de agente através da interface de agente evento. Este agente

notifica o *home* agente apropriado que inicia o relógio de EFT. Este relógio é parado para obter o valor registrado de EFT, quando o agente *home* recebe um chamado de recuperação da unidade móvel, que pode chegar da unidade móvel na mesma estação base ou de uma estação base diferente em que a unidade móvel foi recuperada. Em outro caso, o agente reside na estação base onde o relógio EFT é iniciado. Ele estima o novo EFT como: *(K1 X EFT Registrado)* + *(K2 X EFT*), onde *K1 +K2 = 1*.

O novo EFT é uma soma balanceada do EFT prévio e do EFT Registrado. *K1* indica a confiança no EFT registrado, enquanto *K2* indica a confiança no EFT calculado previamente. Os valores de *K1* e *K2* são funções do ambiente. Em uma rede onde o tempo de falha é relativamente estável, *K2* tem mais peso; em uma rede onde o tempo de falha varia freqüentemente, *K1* tem mais peso. Para aperfeiçoar a utilização de armazenagem os registros desnecessários do *log* são apagados. Esta coleção de sobras é opcional e é feita em conseqüência da unificação do *log*. Quando um *log* de uma unidade móvel é unificado em uma estação base, uma mensagem de coleção de sobras é enviada para todas as estações base hospedando *logs* da unidade móvel como especificado na lista de trace BS-ID. As estações base prévias removem estes *logs* ao receberem esta mensagem. As listas de *BS-ID* e os tamanhos de *log* são apagados da informação de trace na estação base corrente para espelhar unificação, e uma única entrada é criada no trace com a identidade da estação base corrente e o tamanho de log unificado.

3.3.2.7 Esquema forward de unificação de log

Uma vez que a informação de trace contém o tamanho do *log*

armazenado em diferentes estações base o *home* agente pode estimar o tempo de unificação de *log* baseado na velocidade de link da rede e no tamanho total de log. Este tempo é chamado de Tempo Estimado de Unificação de *Log* ou *Estimated Log Unification Time* (ELUT), que pode ser medido como: *Max(BSi-TamanhoDeLog / Velocidade de link de rede + Delay de propagação)*, para todas as estações base no trace. A caracterização exata do valor de ELUT depende de outros fatores tais como se as estações base estão localizadas na mesma área VLR ou diferentes áreas, *delay* de fila, etc. O agente *home* poderia levar em consideração alguns parâmetros disponíveis do sistema assim que possível para estimar o ELUT precisamente. A unificação de *log* é iniciada se *(δ X ELUT) ≤ EFT* ou senão ela é deferida até uma chamada de recuperação ser ouvida pela unidade móvel.

O fator de unificação "*δ*" descreve que fração da unificação de *log* será feita pelo tempo da falha da unidade móvel. O valor padrão pode ser tomado como 1 que indica que a unificação do *log* se inicia somente se ela pode ser totalmente completada no tempo que a unidade móvel é esperada para completar seu *reboot*. Se a unidade móvel reinicia em uma estação base diferente, enquanto o *log* está sendo unificado na estação base prévia, ela tem de esperar pela unificação completa. As variações deste esquema são possíveis se o agente *home* pode estimar o tempo efetivo de *handoff*. Baseado nesse valor se ainda há um longo tempo para o próximo *handoff*, então a unificação de *log* pode iniciar imediatamente após uma falha, assim é mais provável que a unidade móvel em falha irá recuperar na estação base onde ela falhou ao invés de em outra estação base. No evento a unificação de *log* não é executada porque (δ X ELUT) ≤ EFT, o agente *home* espera pela unidade móvel para recuperar. Se a recuperação acontece na mesma estação base, então a unificação de *log* se inicia, mas se a unidade móvel reinicia em uma

estação base diferente então o agente *home* transfere a informação de trace e o *log* armazenado para esta estação quando solicitados. Neste caso, a nova estação base tem de executar a unificação de *log* depois de obter a informação de trace da estação base prévia. Este trace contem o valor EFT calculado recentemente.

4 SEGURANÇA EM BANCOS DE DADOS MÓVEIS

Este capítulo apresenta alguns pontos importantes sobre segurança de bancos de dados móveis presentes em Lubinski (1998). São discutidos também quais objetos que devem ser resguardados e em quais situações.

Clientes buscando seus objetivos são móveis em relação a seus ambientes, às suas localizações e a outras pessoas e terminais. Com a computação móvel e a comunicação *wireless*, por um lado, existem reais possibilidades inspiradoras; mas, por outro lado, a segurança e a privacidade tornam-se vulneráveis. O dinamismo do ambiente móvel é confrontado com os estáticos serviços de segurança. Além disso, freqüentemente a escassez de recursos impede a correta aplicação de mecanismos de segurança e, assim, a informação gerenciada precisa de proteção particular. Os metadados no ambiente móvel em particular são dados adicionais acompanhando a comunicação (telemetadados ou contexto de comunicação). São dados pessoais e devem ser resguardados. A garantia devia efetuar o gerenciamento, acesso e transferência de dados. Nós distinguimos entre ações no site móvel e fixo. As ameaças de segurança para dados e metadados na transferência de dados (comunicação) através de um link *wireless*. A maioria destes são problemas do sistema operacional básico e da camada de rede.

As principais ameaças são o vazamento de informações, a violação de integridade, a recusa de serviços (*denial of services*), usuário ilegítimo e a irresponsabilidade. Tal classificação parece ser muito genérica porque grande parte dos problemas de segurança é relacionada a confidencialidade. Há ainda uma

possível análise que busca o foco em problemas de comunicação e propõe um agrupamento em privacidade de conteúdo, indisponibilidade de link de remetente e recipiente e privacidade de localização.

4.1 Segurança de dados em transferência de dados móveis

As desconexões ocorrem freqüentemente em comunicação *wireless*. Elas podem ser forçadas pelo usuário por causa da economia em custo de comunicação ou ser induzido por falhas. Esta situação pode pôr em perigo a consistência de dados, mesmo sem considerar as réplicas. As desconexões são primariamente um problema das camadas básicas de um banco de dados, mas o sistema de banco de dados é também responsável por evitar perda de dados em caso de tais desconexões inesperadas com a ajuda da recuperação de transação. A maior freqüência de particionamento de rede requer uma recuperação de erro mais poderosa que em redes fixas. Além da recuperação de erros, esta situação oferece aos *atackers* a possibilidade de mascarar de modo idêntico a unidade móvel ou a estação base. Graças a mascara de identidade, os dados estão em risco para serem lançados impropriamente. Além disso, o uso de um link *wireless* facilita escutas, porque informações emitidas no ar são acessíveis para qualquer um com um *receiver* sem qualquer esforço qualquer requerido. Este tipo de violação de segurança é difícil de detectar. Em ambos os casos, a segurança confia na criptografia para conseguir a autenticação de usuário e privacidade de dados. Os usuários móveis são registrados com suas reais identidades ou com um pseudônimo com aquele servidor de autenticação de domínio. O serviço de autenticação poderia prover às partes de comunicação a convicção de que eles estão de fato se comunicando um com o outro. A comunicação subseqüente

4.2 Segurança de metadados em transferência de dados móveis

O metadado consiste de um perfil de usuário, informação sobre a situação do recurso corrente, características da informação, localização e tempo. O atual paradeiro e especialmente o movimento dos usuários são um assunto de privacidade, e idealmente somente o próprio usuário poderia ter conhecimento sobre esses dados. Sua proteção é considerada como o principal objetivo de segurança móvel. A ameaça de possuir o paradeiro do usuário aparece em diferentes camadas. Na camada de rede, a localização do usuário ou a presença em uma célula de rádio particular, respectivamente, é gerenciada a fim de alcançar um usuário móvel para se comunicar com ele. Toda informação de identificação de usuário incluindo a origem e o destino da mensagem tem de ser protegida com a ajuda de criptografia para ocultar uma comunicação com outros usuários da rede. A fim de alcançar uma comunicação anônima, *alias* e pseudônimos são usados. Além disso, a identidade do usuário poderia ser mantida em segredo para o provedor de serviço, ainda que eles consumam serviços que eles tenham que pagar. Não é necessário saber a identidade de usuários para ter informação de solvência de seus nós base de *home*. O site *home* é informado sobre as *aliases* e a real identidade. A garantia de anonimato adicionalmente contra o nó base de *home* requer uma confiável terceira parte para gerenciar os pseudônimos.

Um método implícito para descobrir a localização engana na

possibilidade de defender uma análise de tráfego. A prevenção contra a disponibilidade de trace das conexões de rede em ambientes móveis pode ser oferecida através ou de *MIXes* ou do método de não-divulgação. Ambos métodos usam criptografia. *MIXes* atrasam e coletam diferentes mensagens e as enviam em uma seqüência embaralhada para os receptores ou outro *MIX*. O uso de *MIXes* requer somente uma modificação das redes disponíveis, mas uma suficiente quantidade de mensagens com tamanhos iguais é necessária, caso contrário tráfego falso será criado.

No método de não-divulgação, a introdução de agentes de segurança é proposta. O caminho de comunicação é disfarçado através de uma rota selecionada pelo remetente (com desvios) sobre uma série de diferentes agentes de segurança. Cada agente de segurança sabe somente de seu predecessor e sucessor na cadeia de roteamento. A segurança cresce no caso de agentes de segurança largamente dispersos, possivelmente entre diferentes provedores. Mas os desvios aceitam uma rede com fio e intacta. No caso de requisições de banco de dados, *MIXes* e desvios ampliam o tempo de resposta em uma modo dinâmico e impedem uma otimização eficiente. Outro aspecto da segurança de comunicação *wireless* é a permanente capacidade de alcance de pessoas que põe em perigo o direito de usuários de comunicação autodeterminada. Assim, uma abordagem implementada para gerenciamento da capacidade de alcance de pessoas é proposta. A idéia principal é avaliar e negociar um pedido de comunicação e decidir automaticamente por uma sistema de gerenciamento da capacidade de alcance se um contato pessoal é feito ou não, para permitir chamadas escolhidas ou evitar perturbações. A conexão com o chamado assinante será somente estabelecida em certas situações, isto é, se o contexto da

comunicação negociada tem realizado certas condições, que podem conter informação, por exemplo, sobre parceiros de comunicação e a urgência de pedidos. Este aspecto aumenta o problema de manter a consistência de dados freqüentemente em redes móveis distribuídas.

Enquanto um usuário atravessa as fronteiras das células, suas informações (telemetadados) como localização e perfil de usuário será transferida e replicada para a estação base adjacente. Deste modo, os riscos para os dados pessoais muito suscetíveis são aumentados devido à multiplicação de pontos de ataque e os possivelmente diferentes níveis de confiança concedidos para cada nó. As dificuldades serão mais intensas em relação a diferentes modelos de segurança.

4.3 Segurança de dados

Os efeitos de desconexões como condição especial de recursos foram descritos anteriormente. Um aspecto freqüentemente negligenciado em comunicação móvel é a perda de unidades móveis. Elas são mais suscetíveis a se perder do que *hosts* fixos e as conseqüências são dados e confidencialidade perdidos. A única forma de prevenir a perda de confidencialidade é o uso de encriptação e identificação eficaz, autenticação e mecanismos de controle de acesso. Estes são desafios não específicos para a computação móvel. Justamente os dispositivos móveis são providos apenas de uma proteção muito simples.

Em situações particulares, a computação isolada sem comunicação e sua coleção de ameaças à segurança são necessárias. Mas recursos escassos como pequena capacidade de armazenamento ou energia poderiam prevenir tais situações. Em adição, recursos escassos podem causar situações de falha. O sistema pode não ser apto ou o usuário pode renunciar do cumprimento de métodos

Outro problema pode consistir em uma desproporção entre a quantidade de dados pedidos e os recursos disponíveis, que pode conduzir à violação de integridade ou disponibilidade.

4.4 Segurança de metadados

Como acima mencionado, há uma ameaça à segurança por causa dos diferentes níveis de confiança das estações base. Em ambientes de banco de dados nós temos de ampliar nossa atenção de um lado das estações base para todos os *hosts* móveis e fixos interessados e do outro lado para os modelos de controle de acessos. Nós temos que levar em conta a heterogeneidade dos modelos de controle de acesso (multiníveis, discreto, baseado em função) e a integração heterogênea de dados em modelos homogêneos (além de objetivos heterogêneos de segurança e estratégias). A mesma informação pode ser classificada diferente em sistemas distintos. Nós chamaríamos este efeito de incompatibilidade de modelo de segurança. Nós indicamos o percalço dos movimentos do usuário como o ponto central de segurança móvel. Os paradeiros e movimentos podem ser apanhados do overhead de comunicação ou deduzidos da análise de tráfego. Mas há também outro modo indireto de detectá-los. É óbvio que usuários móveis trabalham em acesso a dados necessários em banco de dados nos seus ambientes de computação correntes, isto é, em suas localizações correntes como a cidade ou a construção onde eles estão, dependentes de parceiros de comunicação e assim por diante. A informação cujos dados o usuário tem acessado

(criado, lido ou alterado) no tempo faz uma dedução dos seus possíveis movimentos por causa das dependências de localização dos dados. Isto é uma ameaça totalmente nova que estamos confrontando no acesso a banco de dados móvel.

4.5 Abordagens de segurança para ambientes de bancos de dados móveis

Agora temos uma grande quantidade de problemas e desafios de segurança. Enquanto há um amplo esforço de pesquisa na área de segurança de redes para ambientes móveis, bancos de dados em conexão com mobilidade são mais negligenciados. Mesmo que nós assumíssemos uma transferência de dados segura e confidencial existem vários problemas de segurança em banco de dados. O capítulo procura demonstrar a resolução de alguns problemas de gerenciamento de dados, segurança de transferência e acesso. Primeiro, nós investigamos as diferenças entre sistemas de banco de dados e transparências relacionadas a segurança. Então nós explicamos a segurança de movimento e localização do usuário e depois disso apresentamos uma abordagem para responder o dinâmico e restrito de recursos ambiente móvel.

4.5.1 Transparências

Existem desafios de segurança básicos graças à mobilidade. Incluída nesses desafios está a contradição das transparências, a transparência no sentido de banco de dados contra a transparência no sentido de privacidade. No primeiro modo o usuário será liberado do conhecimento do sistema interno. Por exemplo, ele vê sua requisição de banco de dados e o resultado relacionado,

enquanto as operações no sistema como análise e otimização são escondidas do cliente. É possível uma comparação com uma visão através de uma janela, onde o vidro da janela é transparente, mas não é visível. Ao contrário a transparência influenciada por privacidade requer expor as operações para as visões dos usuários. Os usuários podem ver a estrutura e as operações nos sistema para controlá-las. Eles querem saber a natureza da janela para descobrir se há um vidro transparente, não um espelho, ou se a janela distorce o mundo real atrás do vidro impropriamente. Os sistemas móveis usados são projetados para suportar o usuário, para reduzir o processamento de requisição remota e para evitar discordâncias entre a quantidade de dados e recursos disponíveis através do pré-processamento inteligente e influência durante a fase de processamento. O gerenciamento e avaliação do contexto de dados é uma precondição necessária. Esses metadados são, de um lado, bastante suscetíveis e, por outro lado, os usuários têm garantido o direito de ler e influenciar o contexto de dados. Isto é importante já que uma adaptação transparente do processo de requisição não é compreensível e pode conduzir a interpretações erradas dos resultados da requisição. Além disso, o usuário deve ter a possibilidade de influenciar o processo de adaptação cuidadosamente. A transparência do banco de dados precisa ser restrita para o beneficio da transparência relacionada à privacidade.

4.5.2 Localizações seguras e movimentos

A localização é uma informação suscetível somente em conexões com as identidades de usuários. A melhor proteção dos paradeiros do usuário consiste na anulação do gerenciamento de informação de localização ou informação de usuário. A informação de movimento pode ser alcançada pela

informação de localização em relação à informação de tempo, já que o movimento é definido como mudança de localização no tempo.

A economia de dados é um conceito na área de privacidade, visa um gerenciamento econômico e o uso de dados pessoais. Dados pessoais podem significar qualquer informação relativa a uma pessoa natural identificada ou identificável. O uso de pseudônimos representa um tipo fraco de economia de dados. Uma vez que sistemas de banco de dados não suportam computação anônima ou pseudônima, os pseudônimos devem ser criados também fora do sistema de banco de dados, ou os usuários têm de agir em funções. Nós recomendamos um projeto de economia de dados durante a fase de projeto dos sistemas de banco de dados. Em uma matriz conectando diferentes níveis de economia de dados com dados e metadados (contexto), uma detalhada visão global da necessária e possível economia de dados é definível. Presumindo que a economia de dados não é aplicável. É possível assumir a localização do usuário diretamente do seu gerenciamento na camada de rede e no contexto móvel indiretamente através da análise de tráfego e do acesso comprometedor. A informação de localização diretamente disponível tem de ser protegida com a ajuda da criptografia e técnicas adequadas de controle de acesso. No caso de um sistema trabalhando corretamente somente os sistemas de adaptação usam a informação de localização. Assim, um contexto móvel poderia ser acessível somente por aquele sistema com exceção dos acessos de usuário para assegurar transparência projetada de privacidade. As pesquisas de localização indireta podem ser evitadas pelo modo de disfarce do fluxo real de informação. Foram descritas anteriormente técnicas de disfarce em transferência de dados. Em sistemas de banco de dados o fluxo de informação entre remetente e receptor é assíncrono por causa do

armazenamento dos dados no banco de dados entre lê-los e escrevê-los. Isto porque há interesse em informações sobre acessos de dados. Para evitar deduzir os paradeiros pelos dados acessados nós consideraremos as dependências de localização dos dados do banco de dados. A proteção das localizações do usuário contra um certo ataque comprometedor requer o conhecimento dos atributos de localização nos bancos de dados. Os bancos de dados podem incluir atributos de localização (cidade, endereço, distrito, país) e atributos relativos a localizações identificáveis como paisagens especiais (Cristo Redentor, Pirâmides, etc). As localizações de dependências são baseadas no conhecimento de contexto do usuário. Os atributos de localização, na sua maior parte, constroem uma hierarquia de localização. Assumem, além disso, que as localizações e atributos referidos a localizações são bem conhecidos. Nesse ponto são definidas a separação de agregação, separação vertical e separação horizontal para alcançar a garantia de paradeiros diretamente gerenciados no contexto móvel assim como localizações indiretamente gerenciadas.

4.5.2.1 Separação de agregação

Conforme descrito anteriormente, as informações de localização e tempo põem em risco a privacidade somente se estão juntas com as identidades do usuário ou a informação relacionada a usuários identificáveis. É definido {*p*} o contexto gerenciado ou as propriedades de balanço. A separação divide as propriedades em {*u, l, t, r*}, onde *u* são as propriedades ou contextos de identificação do usuário, *l* a localização e as propriedades de localização identificáveis, *t* a informação de tempo e *r* as propriedades remanescentes ou informação de contexto. A projeção é obtida por uma separação projetiva. A

agregação destes dados poderia ser acessível somente por usuários autorizados. Os usuários autorizados são administradores, mas com a restrição da separação vertical, e cada usuário influenciado. A proteção é alcançada por uma separação das identidades do usuário ou localização e tempo. Ela tem de ser realizada com a ajuda do controle de acesso, também graças à sua separação física. Enquanto em identidades de usuários gerais são simplesmente determinadas, o estabelecimento dos atributos de localização é um processo muito complexo e precisa de métodos de descoberta de conhecimento.A mobilidade geral é um conceito para suportar essa separação, graças à separação modelada de usuário e localização em contextos. O movimento de um site não revela efetivamente o movimento do usuário.

4.5.2.2 Separação vertical

Uma separação dos dados pessoais é aconselhável para prevenir uma geração de uma visão de usuário extensiva. A separação pode ser vertical ou seletiva. Isto significa que as informações de balanço ou contextos móveis devem ser guardadas com visões baseadas em seleções do banco de dados. As seleções dividem os dados de localização, por exemplo, conforme suas funções ativadas nos acessos aos bancos de dados, fragmentos, domínios, dispositivos, sistemas e assim por diante. Conseqüentemente, somente as seções pequenas dos paradeiros do usuário são visíveis.

4.5.2.3 Separação horizontal

O requerimento da separação horizontal representa um desafio

clássico de banco de dados. Os dados são controlados através do sistema operacional básico e da rede para armazená-los e transferi-los. Não deve existir uma oportunidade para enfraquecer a segurança de banco de dados através dos serviços das camadas básicas. Em outras palavras, a informação do usuário dependente de localização não poderia ultrapassar os limites do banco de dados. O contexto móvel é comum para várias aplicações e camadas, mas deve ser acessível somente em visões particulares.

4.5.3 Ambiente móvel dinâmico e de recurso restrito

Os métodos de segurança e privacidade são freqüentemente muito estáticos enquanto que o ambiente de comunicação móvel é dinâmico e necessita de ajustes de requisições e resultados. O ajuste automático é um conceito básico em pesquisa de computação móvel dentro do projeto *MoVi*. A adaptação é aplicada dinamicamente. Baseado no contexto móvel o componente adequado será selecionado. Os ambientes dinâmicos de bancos de dados móveis surgem da característica de mudança do contexto móvel, isto é, a mudança de localização, os recursos escassos e dinâmicos, o usuário variante e o contexto de aplicação. Resumindo, os problemas de segurança e privacidade causados pelo dinamismo são: os modelos heterogêneos de controle de acesso no site fixo e móvel, a heterogênea integração da informação no mesmo modelo de controle de acesso, a computação isolada, e nenhuma ou a reduzida aplicação do grau de segurança por causa da escassez de recursos.Conforme a adaptação da funcionalidade do banco de dados nós tentaremos usar o conceito de adaptação para responder aos problemas de segurança no ambiente móvel. Um acesso de um sistema de banco de dados no site móvel para um banco de dados fixo pode motivar o problema de

modelos heterogêneos de controle de acesso. A informação é gerenciada, por exemplo, em um modelo de matriz enquanto o modelo de controle de acesso do *site* móvel é realizado em multiníveis. Um problema similar concerne diversas introduções de dados no mesmo modelo. Por exemplo, os endereços de funcionários são acessíveis em um sistema de banco de dados para o departamento pessoal. Em outro banco de dados uma lista de funcionários especiais é definida pra ter acesso. Estas incompatibilidades de modelos não são específicas para a computação móvel. Elas são características de bancos de dados distribuídos e especialmente federados. Mas as precondições mais heterogêneas de hardware e software aumentam os problemas. Um processo de adaptação é preciso para selecionar o modelo necessário e executar um modelo de adaptação de dados. A precondição para um processo de adaptação é a existência de informação de segurança adicional. Nós recomendamos uma transferência de informação sobre o modelo do controle de acesso original e a integração nele em conexão com os próprios dados. Se uma adaptação não é possível (as diferenças de níveis de segurança são intransponíveis) o acesso falha. O processo de adaptação pode garantir que nenhum dado será acessado ou transferido para um domínio inseguro. O outro efeito favorável do processo de adaptação é que a obrigação dos controles de segurança não é somente do usuário. Além do mais, um componente de adaptação poderia controlar a computação *standalone*. Isto implica em proibir o acesso a um banco de dados remoto. Um monitoramento das aplicações correntes é necessário para uma decisão. O componente de adaptação tem de cooperar com as camadas básicas de sistema.Outra tarefa para um processo de adaptação é concebível, a adaptação relacionada a recurso. Ela ajusta os acessos ao banco de dados para os recursos disponíveis. Os dispositivos

móveis pequenos têm freqüente falta de recursos. As técnicas correntes não são aptas para a recuperação destes erros. O outro efeito é que o usuário decide em tal caso executar a operação pretendida por dispensar os graus de segurança. O processo de adaptação pode reduzir os métodos de segurança conforme a funcionalidade reduzida e ainda manter uma segurança mínima e obrigatória.

5 CONCLUSÃO

Este trabalho de monografia procurou apresentar os principais conceitos de computação móvel e de bancos de dados convencionais apresentando a relação de novos desafios e pesquisas referentes a tais conceitos em bancos de dados móveis.

Os pesquisadores e as empresas com suas metas comerciais possuem um objetivo em comum: a construção de sistemas gerenciadores de informações em ambientes móveis capazes de tratar transações e os principais requisitos para o funcionamento adequado de um banco de dados. Os usuários visam unidades móveis que possam gerenciar transações e dessa necessidade surgiu a definição dos sistemas de gerenciamento de informação ubíqua e os sistemas de bancos de dados móveis. Os sistemas bancos de dados móveis são, em essência, sistemas de bancos de dados cliente/servidor distribuídos onde todo o ambiente de processamento é móvel.

As aplicações mais comuns em banco de dados móveis baseiam-se em cliente móvel e host fixo. São usuários que acessam dados pessoais a partir de unidades móveis ou bancos de dados corporativos acessados por funcionários em viagem.

Uma análise sobre os sistemas de bancos de dados móveis pode evidenciar um conjunto de desafios e as pesquisas procuram indicar os possíveis caminhos para a solução de cada um dos problemas expostos.

A mobilidade do sistema implica em mudanças de tratamentos convencionais e a categorização de dados. Os dados passam a ser dependentes de localização, assim como a distribuição e o processamento dos mesmos. É também

por conta da mobilidade que as questões de consistência de dados devem ser revistas, graças à conexão intermitente ou desconexão e à alternância de modos do usuário móvel (desconectado, cochilo e ativo).

Os estudos feitos sobre o assunto apontam para a criação de novos modelos transacionais. O modelo ACID convencional é incapaz de gerenciar o processamento de dados móveis visto que fatores como *handoff*, a alternância de modos de conexão da unidade móvel, a distribuição e o processamento de dados dependentes de localização afetam diretamente no tratamento das exigências do modelo ACID. O desafio aqui então é criar um modelo mais forte de transação ou um modelo de execução ACID que possa manter a mobilidade durante o processamento de dados. Como foram vistas neste trabalho de monografia cada abordagem de modelo de transação que se apresenta propõe tratar em um determinado foco os problemas gerados pela mobilidade do sistema.

Em se tratando de consistência de dados a conectividade fraca e a desconexão de clientes móveis constituiu a necessidade de uma solução baseada na formação de *clusters* de dados de sites fortemente conectados. Para diferentes *clusters* são permitidos níveis de inconsistência e em cópias de mesmo *cluster* é necessária a consistência mútua.

Alguns mecanismos de controle de concorrência também foram propostos em virtude da mobilidade. A base do controle de concorrência é a execução serial, mas esta anula os benefícios da multiprogramação. O desafio neste caso é a execução de transações em paralelo garantindo a serialização. A proposta para o controle de concorrência observada neste trabalho é o esquema *two-phase locking* baseado em *locks* ou bloqueios de itens de dados.

A mobilidade torna mais complexa a tarefa de recuperação dos

sistemas de bancos de dados móveis, isto por conta de fatores como *logging* dependente de localização e a ocorrência aleatória de handoffs, por exemplo. O mais importante no esquema de recuperação é o gerenciamento de *log*. Em bancos de dados móveis com restrição de recursos o esquema de *logging* convencional é inviável, sendo necessário um outro esquema eficiente nesse caso. Alguns esquemas desenvolvidos que podem ser destacados: esquema de recuperação híbrida de três fases, recuperação de falha e checkpointing de baixo custo e o gerenciamento de log baseado em agentes móveis.

Avaliadas questões de segurança é possível notar que a privacidade é colocada em risco quando informações de localização e tempo se juntam aos dados de identidades do usuário. Em termos gerais o ambiente móvel é dinâmico e os métodos de privacidade e segurança são estáticos. São necessários ajustes de requisições e resultados.

Alguns bancos de dados disponíveis no mercado foram estudados para a conclusão deste trabalho. São eles: Oracle Lite, ThinkDB, DB2 Everyplace, Microsoft SQL Server Mobile Edition. Além destas, ferramentas de sincronização de dados também foram avaliadas; MobiLink e Active Sync.

O Active Sync é o aplicativo usado por dispositivos móveis com Windows CE para sincronização de dados com computadores domésticos e de escritório.

O MobiLink é uma solução da Sybase para replicação de dados entre bases de unidades móveis e o banco de dados original. Através de um servidor, o MobiLink Server, a aplicação atende requisições de emissão e recepção de dados. Estas requisições são passadas para o banco de dados original que executa scripts (comandos SQL) de sincronização que indicam os dados que são

trocados entre as bases. Essa ferramenta oferece controle de transações e integridade referencial.

As ferramentas de banco de dados avaliadas observam principalmente a tarefa de sincronização dos dados e pouco apresentaram sobre a implementação de novos modelos de transação e consistência, mecanismos de controle de concorrência, esquemas de recuperação e segurança em ambientes móveis.

Uma avaliação geral destas ferramentas de bancos de dados móveis aponta que poucos dos recursos e tratamentos vistos neste trabalho estão de fato implementados em soluções comerciais. É possível concluir que muitas das propostas apontadas como soluções aos desafios de gerenciamento de bancos de dados móveis ainda estão sendo desenvolvidas em pesquisas e estudos.

REFERÊNCIAS

KUMAR, Vijay. *Mobile Database Systems.* New Jersey: Wiley Interscience, 2006.

HARRISON, Thomas H. *Intranet Data Warehouse*. São Paulo: Berkeley Brasil, 1998.

KORTH, Henry F.; SILBERSCHATZ, Abraham. *Sistemas de Banco de Dados.* 3 ed. São Paulo. Makron Books, 1999.

ELMASRI, Ramez; NAVATHE, Shamkant. *Sistemas de Bancos de Dados – Fundamentos e Aplicações.* 3 ed. Rio de Janeiro. LTC, 2002.

AMADO, Paulo Gustavo Fell; *Banco de Dados Móveis: visão geral, desafio e soluções atuais.* Recife: Universidade Federal de Pernambuco, 2002. Disponível em: <http:// www.cin.ufpe.br/~tg/2002-1/pgfa-proposta.doc>. Acesso em: 16 de abril de 2007.

LUBINSKI, Astrid. *Security Issues in Mobile Datbase Access.* University of Rostock, Computer Science Department, Rostock, 1998.

PIRES, Rafael P.; REDIN, Ricardo M.; *BELUSSO*, Rubens C.; LIMA, João C.D.; AUGUSTIN, Iara. *Comunicação entre Componentes da Aplicação em Ambiente Pervasivo.* In: XV Seminário Regional de Informática, 2005, Santo Ângelo, RS. Anais do XV Seminário Regional de Informática, 2005.

YAMIN, A. C. ; AUGUSTIN, Iara ; BARBOSA, J. L. V. ; da SILVA, L. C. ; GEYER, C. F. R. . *Explorando o Escalonamento no Desempenho de Aplicações Móveis Distribuídas* . In: WSCAD 2001 - Workshop em Sistemas Computacionais de Alto Desempenho, 2001, Pirenópolis. WSCAD 2001 - Workshop em Sistemas Computacionais de Alto Desempenho. Porto Alegre: Sociedade Brasileira de Computação, 2001. p. 1-8.

CÔRTES, S.C. ; LIFSCHITZ, S. . *Bancos de Dados para um Ambiente de Computação Móvel*. Livro Texto das Jornadas de Atualização em Informática (JAI). Campinas: SBC, 2003, p. 95-144.

ENDLER, Markus ; SILVA, F. J. S. . *Requisitos e Arquiteturas de Software para Computação Móvel.* In: Workshop sobre Métodos e Serviços para Computação Móvel, 2000, São Paulo, SP.

BOROS, Rogério. *CDMA – Code Division Multiple Access*. Disponível em: <http://www.wirelessbrasil.org>. Acesso em: 21 de maio de 2007.

AMODEI JUNIOR, Aurélio; DUARTE, Otto C.M.B.. *Segurança no Roteamento em Redes Móveis Ad Hoc*. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2003.