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

Fernando José Braz

ANÁLISE DE MECANISMOS PARA RECUPERAÇÃO DE FALHAS EM BANCOS DE DADOS MÓVEIS

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

Orientador: Murilo Silva de Camargo

Florianópolis, outubro de 2002

ANÁLISE DE MECANISMOS PARA RECUPERAÇÃO DE FALHAS EM BANCOS DE DADOS MÓVEIS

Fernando José Braz

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

_____

Prof. Dr. Fernando O Gauttier
(Coordenador do Curso)

Banca Examinadora

_____

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

_____

Prof. Dr. Roberto Willrich

_____

Prof. Dr. Rosvelter J. C. da Costa

"Dias inteiros de calmaria, noites de ardentia,

dedos no leme e olhos no horizonte,

descobri a alegria de transformar distâncias em tempo.

Um tempo em que aprendi a entender as coisas do mar,

a conversar com as grandes ondas

e não discutir com o mau tempo.

A transformar o medo em respeito, o respeito em confiança.

Descobri como é bom chegar quando se tem paciência.

E para se chegar onde quer que seja,

aprendi que não é preciso dominar a força, mas a razão.

É preciso antes de mais nada querer."

Amyr Klink

DEDICATÓRIA

A minha esposa, amiga e companheira Maricéia, que em todos os momentos desta jornada me prestou o seu irrestrito apoio, sua compreensão e o seu enorme amor, sem a qual esta tarefa não teria sido concluída.

A minha doce e meiga filha Laura, que sacrificou incontáveis horas de suas deliciosas brincadeiras para me dar a tranqüilidade necessária para este trabalho.

AGRADECIMENTOS

Agradeço a todos os amigos que direta ou indiretamente participaram da construção deste trabalho.

Agradeço o incentivo e a paciência do Professor Murilo que através da sua orientação tanto tem contribuído para a minha formação.

Agradeço a sábia teimosia e a dedicação do Professor Lineu, que insiste na árdua tarefa de me formar, ensinando-me com suas valiosas experiências e seu profundo conhecimento.

A todos os colegas da Divisão de Tecnologia da Informação da UNIVILLE, que sempre me prestaram o apoio necessário para a conclusão desta tarefa.

A minha boa mãe, que sempre esteve pronta a colaborar no meu dia a dia, e tanto contribuiu para a minha formação pessoal.

Mais uma vez, a minha esposa e a minha filha, sem as quais este trabalho não teria sequer começado.

SUMÁRIO

ÍNDICE DE FIGURAS

INDICE DE TABELAS

RESUMO

As ocorrências de desconexões nos ambientes de computação móvel podem traduzir-se em falhas nos processos dos bancos de dados, levando o ambiente de banco de dados a um estado inconsistente. Além do processo de movimentação, a possibilidade de esgotamento da energia disponível nas unidades móveis pode levar a uma falha na manutenção das transações de um banco de dados móvel. Mecanismos de recuperação de falhas em ambientes de bancos de dados móveis são propostos na literatura.

A análise de algumas destas propostas foi efetivada procurando identificar o comportamento do mecanismo em relação aos seguintes itens: *localização da área de armazenamento estável, local de armazenamento dos registros de checkpoint e log, possibilidade de previsão da ocorrência da falha, utilização de estruturas de dependência* e *construção de registros de checkpoint através da troca de mensagens.*

Também foram analisadas as propostas de recuperação de falhas em bancos de dados móveis, implementadas pelos principais SGBD comerciais. Esta análise tem por objetivo identificar a aplicação de alguns dos mecanismos estudados, nos produtos comerciais em análise. O estudo de cada produto aconteceu com base na verificação da documentação técnica das soluções de computação móvel de cada fabricante. Foram identificados os tratamentos dados por cada produto em relação aos seguintes itens: *suporte a transações, possibilidade de trabalho desconectado, utilização de arquivo de log, mecanismo de sincronização e detecção de conflitos, suporte a falhas em transações entre as unidades móveis e a rede estacionária.*

PALAVRAS-CHAVE: Bancos de dados móveis, redes sem fio, computação móvel, banco de dados distribuídos, recuperação de falhas.

ABSTRACT

The occurrences of disconnections in the environments of mobile computation can be translated in flaws in the processes of the databases, taking the database environments to an inconsistent state. Besides the movement process, the possibility of exhaustion of the available energy in the mobile units can take the a flaw in the maintenance of the transactions of a mobile database. Mechanisms of recovery of failures in environments of mobile databases are proposed in the literature.

The analysis of some of these proposals was executed trying to identify the behavior of the mechanism in relation to the following items: location of the area of stable storage, place of storage of the checkpoint registrations and log, possibility of forecast of the occurrence of the flaw, use of dependence structures and construction of checkpoint registrations through the change of messages.

Also the proposals of recovery of failures were analyzed in mobile databases, implemented by main commercial SGBD. This analysis has for objective to identify the application of some of the studied mechanisms, in the commercial products in analysis. The study of each product happened with base in the verification of the technical documentation of the solutions of each manufacturer's mobile computation. They were identified the treatments given by each product in relation to the following items: support to transactions, possibility of disconnected work, use of log file, synchronization mechanism and detection of conflicts, support the failures in transactions between the mobile units and the stationary net.

KEYWORDS: *Mobile databases, wireless, mobile computation, distributed databases , recovery of failures*

CAPÍTULO I

INTRODUÇÃO

A mobilidade do usuário apresenta características únicas para a utilização de bancos de dados. Um dos grandes problemas em ambientes de bancos de dados móveis é a possibilidade da ocorrência de desconexão sem qualquer antecipação do usuário. Um exemplo desta situação é o ingresso do usuário em área geográfica não coberta pelo sinal de conexão com a rede estacionária. Para que este processo de desconexão ocorra sem levar o banco de dados a um estado inconsistente, é necessária a implementação de mecanismos de recuperação de falhas.

Os mecanismos de recuperação de falhas em bancos de dados móveis garantem a preservação da consistência do banco de dados perante a ocorrência de falhas. Além da previsão da falha, a localização da sua ocorrência (unidades móveis ou estações de suporte) é também um fator especialmente importante. A ocorrência de falhas em estações de suporte pode levar todo o ambiente de banco de dados a um estado inconsistente, enquanto que as falhas nas unidades móveis podem contribuir para o prejuízo do trabalho do usuário móvel.

O trabalho aqui apresentado tem dois objetivos principais: (*i*) a apresentação, e análise comparativa, das principais propostas de algoritmos para a recuperação de falhas em bancos de dados móveis; (*ii*) a pesquisa dos mecanismos de recuperação de falhas implementados pelos principais sistemas gerenciadores de bancos de dados (*IBM, Microsoft, Oracle* e *Sybase*), em seus produtos para computação móvel e uma posterior análise destes mecanismos. A análise comparativa das propostas de algoritmos será efetuada sobre os seguintes parâmetros de comparação: localização da área de armazenamento estável, local de armazenamento dos registros de *checkpoint* e log, possibilidade de previsão da ocorrência da falha, utilização de estruturas de dependência e construção de registros de *checkpoint* através da troca de mensagens. A pesquisa dos mecanismos de recuperação de falhas, implementados nos sistemas gerenciadores

comerciais, tem por objetivo verificar a implementação dos algoritmos pesquisados nestes aplicativos.

Este trabalho foi organizado em seis capítulos, no capítulo II foram apresentados os principais aspectos relacionados à computação móvel. As propostas de arquiteturas, problemas de localização das unidades móveis, protocolos utilizados pela computação móvel, modelos de computação móvel e características de suporte do sistema para a mobilidade, fazem parte deste capítulo.

No capítulo III foram abordados os temas relativos à utilização de bancos de dados em ambientes móveis. As arquiteturas possíveis, a difusão e *caching* de dados, o gerenciamento de transações e processamento de consultas em ambientes de bancos de dados móveis, fazem parte deste capítulo. Ainda no capítulo III, foram apresentadas algumas aplicações práticas de bancos de dados móveis. Estas aplicações foram apresentadas com base na documentação técnica encontrada na pesquisa bibliográfica, a intenção desta abordagem foi de ressaltar a utilização de aplicações de bancos de dados móveis nas mais diversas situações.

No capítulo IV foram apresentadas as propostas de recuperação de falhas em ambientes de bancos de dados móveis encontradas no levantamento bibliográfico. A descrição e as principais características de cada protocolo foram apresentadas neste capítulo. Além da apresentação das propostas, foi também elaborada uma análise comparativa entre as propostas, esta análise foi baseada em cinco características escolhidas: (*i*) localização da área de armazenamento estável, (*ii*) local de armazenamento dos registros de *checkpoint* e *log,* (*iii*) classificação entre protocolos coordenados e não coordenados, (*iv*) localização da ocorrência da falha e (*v*) possibilidade de previsão da ocorrência da falha.

No capítulo V foram apresentados os principais sistemas gerenciadores de bancos de dados móveis e suas propostas e ferramentas para suporte à computação móvel. Os sistemas apresentados foram os seguintes: *Oracle9i Lite, SQL SERVER CE, Sybase SQL Anywhere Studio* e *IBM DB2 Everywhere.*

O capítulo VI compreende um resumo sobre o trabalho desenvolvido, as conclusões obtidas durante o trabalho e perspectivas futuras para os processos de recuperação de falhas em ambientes de bancos de dados móveis.

CAPÍTULO II

Computação Móvel

Um ambiente de computação móvel basicamente é composto de estações conectadas e participantes de uma rede fixa (dispositivos fixos), fornecendo a comunicação através de *interfaces*[1] para dispositivos desconectados fisicamente da rede. Os componentes desconectados da rede fixa são chamados de dispositivos móveis. A característica da mobilidade do dispositivo é a base de todo o conceito da computação móvel.

O ambiente de computação móvel é caracterizado segundo FORMAN & ZAHORJAN (1994) por sua particularidade de comunicação, mobilidade e portabilidade. A comunicação pode ser conduzida sobre redes *wireless*[2] que estão propensas a desconexões, erros e baixa largura de banda de rede. De acordo com OZU & VALDURIEZ (1999) a mobilidade de alguns componentes de uma rede estacionária abre a possibilidade da movimentação dos dados no espaço, trabalhando-se a partir deste momento com o conceito de dados dependentes da localização.

No decorrer deste capítulo serão abordados os conceitos relativos à computação móvel, seu ambiente e características específicas desta arquitetura. Aspectos relativos aos problemas resultantes da mobilidade do usuário serão também aqui contemplados.

Alguns conceitos da computação sem fio serão abordados neste capítulo, arquiteturas, tecnologias e protocolos da *wireless computing* [3]serão apresentados como forma de promover um melhor entendimento do uso deste meio para o suporte da computação móvel.

2.1 Arquitetura da computação móvel

[1] *Interfaces*: ponto no qual é feita uma conexão entre dois elementos, de forma que um possa trabalhar com o outro.
[2] *Wireless*: canal de comunicação sem fio.
[3] *Wireless computing*: computação sem fio, utilizando canais de rádio, infravermelho, microondas etc.

A arquitetura básica de um ambiente de computação móvel segundo MATEUS & LOUREIRO (1998), consiste da tradicional infra-estrutura de comunicação fixa com computadores estáticos ligada a uma parte móvel. Esta parte móvel é representada por uma área ou célula onde existe a comunicação sem fio dos dispositivos móveis. Dentro desta célula acontece a livre movimentação dos dispositivos, permitindo inclusive que aconteça a movimentação do ambiente de uma célula para outra.

Segundo DUNHAM & HELAL (1995) o modelo mais comum de um sistema que suporta a computação móvel consiste de componentes estacionários e componentes móveis. O único componente móvel é a unidade móvel *(mobile unit)* que consiste de um computador capaz de efetuar conexão com uma rede fixa através de um canal sem fio. Os componentes estacionários (fixos) são conectados através de uma rede de alta velocidade (Mbps – Gbps). Os componentes da rede fixa são classificados em *fixed hosts* e *base stations*. Um *fixed host* é um computador localizado na rede fixa sem a capacidade de efetuar conexões com as unidades móveis *(mobile unit)*. A estação base (*base station)* tem a capacidade de conectar com uma unidade móvel *(mobile unit)* tendo em vista ser equipada com uma interface *wireless*. As estações base *(base station)* são também referenciadas por *mobile support stations,* estas estações atuam como interfaces entre os dispositivos móveis e fixos (Figura 1).

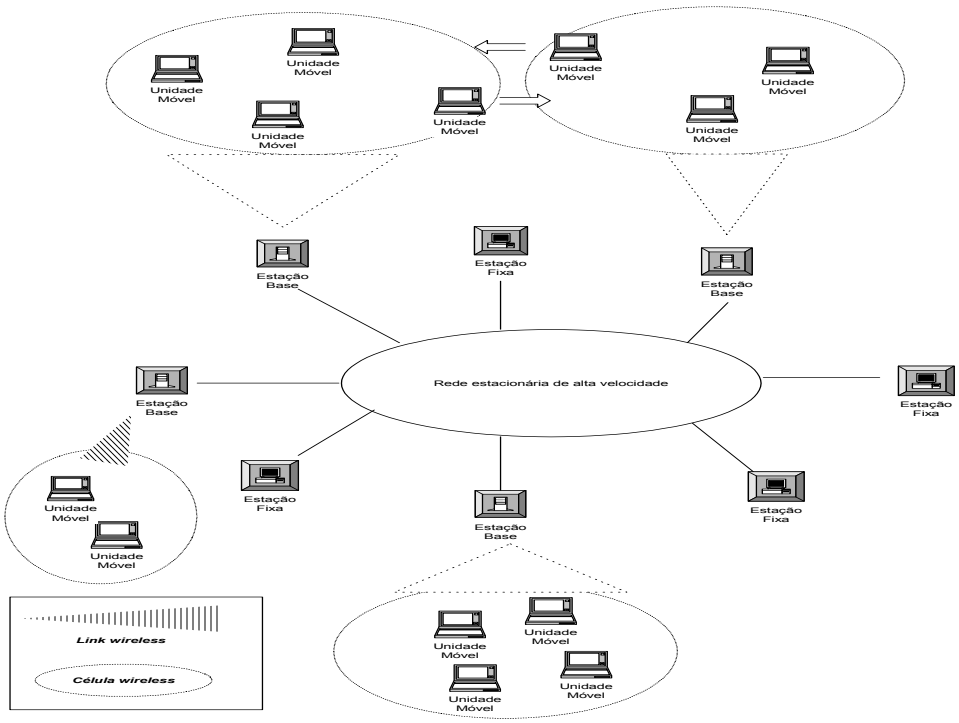

Figura 1 - Arquitetura Computação Móvel

2.1.1 Dispositivos Móveis

As unidades móveis, *hosts móveis* ou ainda *mobile stations* são computadores portáteis interligados em rede através de conexões sem fio, ambiente que permite ao usuário um alto grau de mobilidade no uso de tais estações. OZU & VALDURIEZ (1999) utiliza a capacidade das unidades móveis para formalizar uma divisão entre elas. Em um grupo estariam classificadas as estações com recursos limitados, neste caso os dados estariam localizados nos dispositivos fixos, com as unidades móveis buscando os dados conforme a necessidade imediata de cada estação. Em outro grupo estariam classificadas as estações dotadas de uma grande capacidade de armazenamento tanto de dados quanto de energia. Estas estações fazem uso desta capacidade para manter dados nativos e desta forma permitir o acesso de outras estações na busca por informações ali mantidas.

2.1.2 Dispositivos fixos

Ao contrário dos dispositivos móveis, estes dispositivos fazem parte da rede estacionária cuja localização não muda. A comunicação entre a rede fixa e os dispositivos móveis é provida por estações de suporte móveis (*mobile support station*).

As estações de suporte móveis transmitem e recebem sinais dos dispositivos localizados em uma área específica, a conexão destes dispositivos com a rede fixa é implementada fisicamente através de linhas de comunicação. Com a intenção de facilitar o gerenciamento da mobilidade dos dispositivos móveis, as regiões de cobertura foram divididas em áreas menores denominadas células. A comunicação destas células com a rede fixa é realizada por meio de *interfaces* sem fio, sendo que o gerenciamento de cada célula é tarefa das estações base (dispositivos fixos). Na eventualidade da proximidade entre os dispositivos móveis, a comunicação poderá acontecer sem a interferência de uma estação de suporte móvel.

2.2 Histórico da computação móvel

A evolução da computação móvel tem o seu início com Hans Christian Orested que em 1820 fez a descoberta do campo elétrico produzido pela corrente magnética. O conceito da radiotelegrafia pode ser considerado como a primeira forma de comunicação sem fio, grandes saltos foram dados no desenvolvimento e aprimoramento da tecnologia de comunicação a partir daquele momento.

A chegada do telefone e da tecnologia digital permitiram a utilização do computador na comunicação de dados trazendo um grande impulso nesta tecnologia, a seguir apresentaremos a cronologia desta evolução conforme MATEUS & LOUREIRO (1998).

1820 Hans Christian Oersted descobre que a corrente elétrica produz um campo magnético. André Marie Ampére quantifica esta observação na Lei de Ampére.

1830 Joseph Henry descobre que a variação do campo magnético induz uma corrente elétrica mas não publica o resultado. Em 1831, Michael Faraday descobre independentemente esse efeito que passaria a ser conhecido como a Lei de Faraday.

1864 James Clark Maxwell modifica a Lei de Ampére e desenvolve as quatro equações de Maxwell sobre campos magnéticos.

1876 Alexander Graham Bell inventa o telefone.

1887 Heinrich Rudolph Hertz detecta as ondas eletromagnéticas previstas pelas

equações de Maxwell.

1896 Guglielmo Marconi inventa o primeiro receptor sem fio: telégrafo sem fio.

1907 Início do serviço de radiodifusão comercial transatlântico (antenas de 30 x 100 m).

1914 Início da Primeira Guerra Mundial. Rápido desenvolvimento das comunicações e sua interceptação.

1921 Entrada em operação da radiodifusão comercial nos Estados Unidos.

1928 A polícia de Detroit introduz um sistema de acionamento de carros baseado em radiodifusão na faixa de 2 MHz.

1933 A FCC autoriza o uso de quatro canais na faixa de 30-40 MHz.

1935 FM (*modulação em freqüência*) surge como alternativa para a AM (*modulação em amplitude*), reduzindo os problemas de ruídos na transmissão.

1939 Pesquisa e uso da comunicação via rádio tem grande desenvolvimento durante a Segunda Guerra Mundial.

1945 AT&T Bell labs inicia experimentos no uso de freqüências mais altas com o objetivo de melhorar os serviços móveis.

1947 AT&T lança o IMTS (*Improved Mobile Telephone Service*), um sistema de transmissão onde apenas uma torre de alta potência atendia uma grande área ou cidade. Em seguida, AT&T propõe o conceito de celular.

Anos 50 Os sistemas requerem uma elevada banda para transmissão, uma faixa de 120 kHz para transmitir um circuito de voz de apenas 3 kHz. Esta faixa é reduzida pela metade. Com os transistores os equipamentos reduzem de tamanho e são transportáveis. Os primeiros sistemas de *paging* começam a surgir.

Anos 60 Um novo receptor de FM permite reduzir a banda para 30 kHz, abrindo espaço para um maior número de canais de comunicação com o mesmo espectro. Bell Labs já testa as técnicas de comunicação celular e surgem os primeiros aparelhos portáteis.

Anos 70 A FCC aloca um espectro de freqüências para os sistemas celulares. Nesse período AT&T lança o sistema celular conhecido por AMPS (*Advanced Mobile Phone System*). Inicialmente era um serviço de luxo. Destinado para

uso em automóveis e de aplicação limitada tendo em vista a baixa durabilidade das baterias.

1983 O sistema AMPS evolui com a primeira rede celular americana lançada em 1983. Outros sistemas similares entram em operação no mundo: TACS (*Total Access Communications System*) no Reino Unido, NMT (*Nordic Mobile Telephone Service*) na Escandinávia, NAMTS (*Nippon Advanced Mobile Telephone System*) no Japão. O AMPS ainda em uso nos EUA, Brasil e grande parte do mundo, é considerado um sistema de primeira geração. Surgem os sistemas de transmissão digital, pelas técnicas de processamento digital de sinais foi possível reduzir a banda necessária, viabilizando os sistemas móveis digitais.

1991 Validação inicial dos padrões TDMA e CDMA nos EUA. Introdução da tecnologia microcelular.

1992 Introdução do sistema celular Pan-Europeu GSM (*Groupe Spéciale Mobile*).

1994 Introdução do sistema CDPD (*Cellular Digital Packet Data*). Início dos serviços PCS (*Personal Communication Services*) CDMA e TDMA.

1995 Início dos projetos para cobertura terrestre de satélites de baixa órbita, como o projeto Iridium.

2000 Estima-se um número de 100 milhões de assinantes do sistema móvel celular no mundo inteiro

2002 Lançamento no Japão de aparelhos celulares com a tecnologia 3G, permitindo a comunicação de dados com velocidade em até 40 vezes superior aos aparelhos da segunda geração, estes aparelhos abrem a possibilidade do envio de informações em vídeo e de forma interativa, iniciando a videoconferência móvel.

2.3 Características da computação móvel

A computação móvel habilita seus usuários a acessar e trocar informações enquanto se deslocam ou trabalham em locais diferentes do seu ambiente normal. Segundo FUCHS et al. (1999) as unidades móveis têm diferentes níveis de capacidade

computacional e de rede, *pagers*[4], por exemplo, podem receber e enviar pequenas mensagens, *personal digital assistants*[5] *(PDA's)* podem executar algumas tarefas mais complexas como agendas eletrônicas. Computadores portáteis já possuem recursos comparáveis às estações fixas *(fixed hosts)* permitindo a conexão tanto para redes estacionárias quanto para *wireless.*

Redes *wireless* são usadas em ambientes onde o uso das redes estacionárias não é possível ou economicamente inviável. Redes temporárias podem ser mais baratas e de configuração mais simples fazendo uso da tecnologia *wireless* em contra partida à rede estacionária padrão. Esta característica é particularmente interessante e diretamente aplicável em ambientes de recuperação de desastres como incêndios, inundações e terremotos.

FUCHS et al. (1999) fazem referência sobre o fato de unidades móveis possuírem características que as diferenciam das estações fixas, algumas destas características serão descritas nos sub itens a seguir.

2.3.1 Localização dinâmica

Como o usuário move-se durante a utilização do dispositivo, a localização da unidade móvel na rede está em contínua alteração. Esta característica poderá causar um grande impacto no processamento das consultas, tendo em vista a necessidade de determinação do local ótimo para que se efetue o processamento.

2.3.2 Mobilidade

As unidades móveis comunicam-se com diferentes estações base, na medida em que se movem para dentro e fora de novas células. A mobilidade também inclui o fato de que os clientes movem-se segundo padrões de atividade de leitura e escrita. Por questões de desempenho pode ser necessário que alguns itens de dados ou tarefas precisem mover-se para as proximidades da unidade móvel correspondente.

---

[4] *Pager*: dispositivo que recebe e envia mensagens a partir de uma central de controle.
[5] *PDA*: computador dedicado com programas como agenda, calendário, bloco de notas, calculadora etc.

2.3.3 Desconexão

Uma estação móvel *(mobile host)* entra em estado desconectado quando ultrapassa os limites de cobertura dos emitentes. Neste estado a estação móvel não poderá enviar ou receber mensagens, devendo existir a possibilidade de continuidade do trabalho com as informações já mantidas na unidade.

2.3.4 Capacidade limitada das baterias

Segundo FORMAN & ZAHORJAN (1994), operações de rede e acesso a disco são os maiores consumidores de recursos de energia, fato que deverá ser observado na etapa do desenvolvimento de um ambiente de computação móvel.

2.3.5 Características de rede distintas

As várias tecnologias *wireless* possuem qualidade de serviço completamente diferente NEMZOW (1995) tais como largura da banda de rede, custo, taxas de perda de pacotes e latência. Esta diversidade deve ser atendida e resolvida na tentativa de reduzir o impacto no ambiente da computação móvel.

2.3.6 Tipos de falhas

A primeira categoria de falha inclui todas as falhas que não podem ser reparadas; por exemplo, a queda e a quebra da estação ou ainda a perda ou roubo da mesma, este tipo de falha é classificada como *hard failures.* A segunda categoria engloba as falhas que não danificam completamente a estação, por exemplo, a descarga da bateria, erros do sistema operacional ou ainda problemas específicos com a utilização da memória do equipamento, esta segunda categoria de falha é referida como *soft failures*.

2.3.7 Handoff

Em PITOURA & SAMARAS (1999) encontramos uma definição bastante precisa deste fenômeno exclusivo do ambiente de computação móvel. Durante o processo de movimentação das unidades móveis é possível que se ultrapassem os limites das células em que se encontra uma determinada estação. A partir deste momento, e na medida em que se ultrapassam os limites da próxima célula acontece o processo de *handoff*. Este processo envolve a atualização dos bancos de dados que mantêm a localização das unidades móveis (Figura 2).

No caso de redes para o tráfego da voz este fenômeno não chega a ser preocupante, tendo em vista o nível de tolerância de ruído que uma rede deste tipo pode suportar. O mesmo não acontece no caso de redes para o tráfego de dados, o gerenciamento desta ocorrência torna-se mais sofisticada devido ao baixo nível de falha permitido neste ambiente.

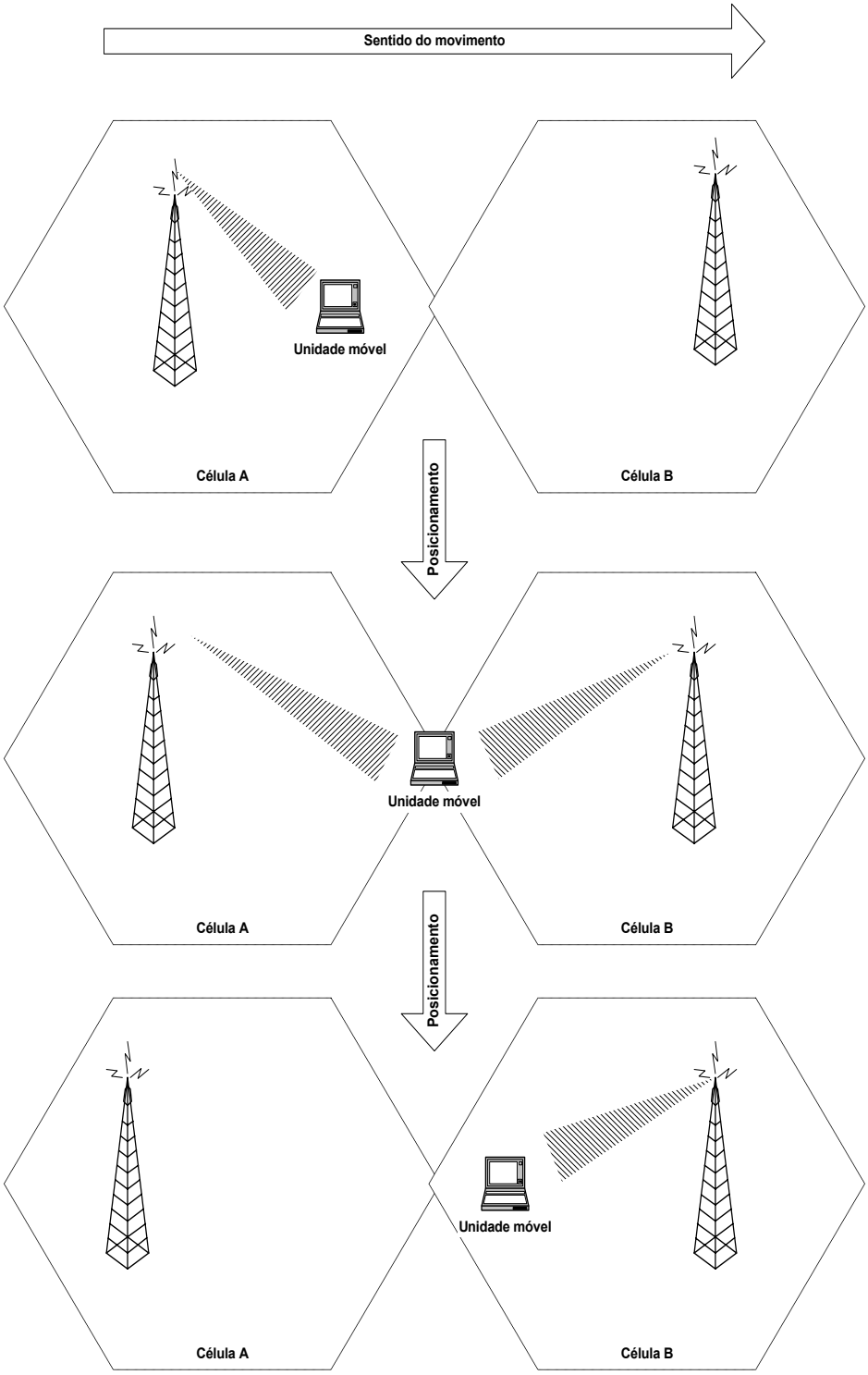

Figura 2 - Processo de *handoff*

2.4 Computação sem fio (*wireless computing*)

A utilização de redes *wireless* no suporte para a computação móvel vem sofrendo um grande incremento devido à sua característica de mobilidade. A comunicação neste tipo de rede pode ser feita com o uso de ondas de rádio, infravermelho, satélite, microondas, *laser* etc.

Segundo TANENBAUM (1997), com a devida manipulação das faixas do espectro eletromagnético a transmissão de informações torna-se possível mediante o uso das ondas eletromagnéticas. A seguir abordaremos algumas características específicas de utilização do meio.

2.4.1 Transmissão de rádio

As ondas de rádio conseguem atingir grandes distâncias e transpor os obstáculos com relativa facilidade, além do fato de sua transmissão ser onidirecional. O fato de poder transmitir as informações em todas as direções é além de uma grande vantagem (transmissor e receptor não precisam estar alinhados), fonte de um grande problema se pensarmos no caráter da privacidade dos dados.

Nas freqüências baixas, as ondas de rádio possuem maior capacidade de transpor os obstáculos, porém a potência de transmissão diminui na mesma medida em que cresce a distância da origem. Nas freqüências altas as ondas tendem a viajar em linhas retas e a refletir nos obstáculos. Em todas as freqüências existe a possibilidade de interferência de equipamentos elétricos TANENBAUM (1997).

2.4.2 Transmissão de microondas

A transmissão de microondas consiste basicamente na concentração da energia em um pequeno feixe através do uso de uma antena parabólica. Este tipo de transmissão proporciona um sinal com maior qualidade, trazendo como característica um baixo valor no índice de ruído da transmissão. Porém existe a necessidade de um alinhamento perfeito entre as torres de transmissão e recepção dos sinais. Devido ao fato de as ondas viajarem em linha reta, pode existir a necessidade da utilização de torres para a

repetição e amplificação dos sinais. Outra característica marcante deste meio de transmissão é a impossibilidade das microondas de transpor os obstáculos.

2.4.3 Ondas infravermelhas

As ondas infravermelhas são relativamente direcionais, baratas e fáceis de construir, porém apresentam um grande limitador que é a incapacidade total de atravessar objetos sólidos TANENBAUM (1997). Este fator limitador pode ser visto como uma grande vantagem com relação ao aspecto da privacidade da informação.

2.4.4 Ondas de luz

A transmissão por ondas de luz é unidirecional, necessitando de que cada agente no processo de comunicação possua seu próprio transmissor e foto-receptor. Pelo fato do feixe de transmissão ser bastante estreito, existe a necessidade de uma grande exatidão no direcionamento dos transmissores e receptores, e por vezes a utilização de lentes no intuito de aprimorar a precisão do raio de transmissão.

Este esquema de transmissão oferece uma largura de banda bastante alta a um custo relativamente baixo e, ao contrário das microondas, não precisa de licença de uso do órgão competente.

2.5 Arquiteturas de redes sem fio

A seguir apresentaremos as principais arquiteturas de rede sem fio em uso atualmente: *ad hoc*, arquitetura celular, *wireless lan*[6]*, wireless wan*[7] e *wireless man*[8] .

2.5.1 Arquitetura *Ad Hoc*

Neste tipo de arquitetura todas as unidades móveis podem se comunicar entre si, sem a necessidade de uma infra-estrutura fixa, ou qualquer ponto de acesso (Figura 2). Havendo a necessidade de comunicação entre duas unidades móveis esta comunicação

[6] *Wireless LAN* : rede local sem fio
[7] *Wireless WAN*:: rede geográfica sem fio
[8] *Wireless MAN*: rede metropolitana sem fio

se concretizará diretamente entre as duas unidades, ou através da utilização de outras unidades para o envio dos pacotes de comunicação.

Na arquitetura *Ad Hoc* a topologia da rede sofre mudanças constantes. A baixa largura de banda de rede, altas taxas de erro nos enlaces de rede e o processo de adaptação e reconfiguração das rotas para o tráfego dos pacotes, são alguns dos problemas inerentes a esta arquitetura.

A rápida instalação e configuração da rede, alta conectividade, alta taxa de mobilidade das unidades e a tolerância à falhas são citadas na literatura como algumas das vantagens da utilização da arquitetura *Ad hoc*.

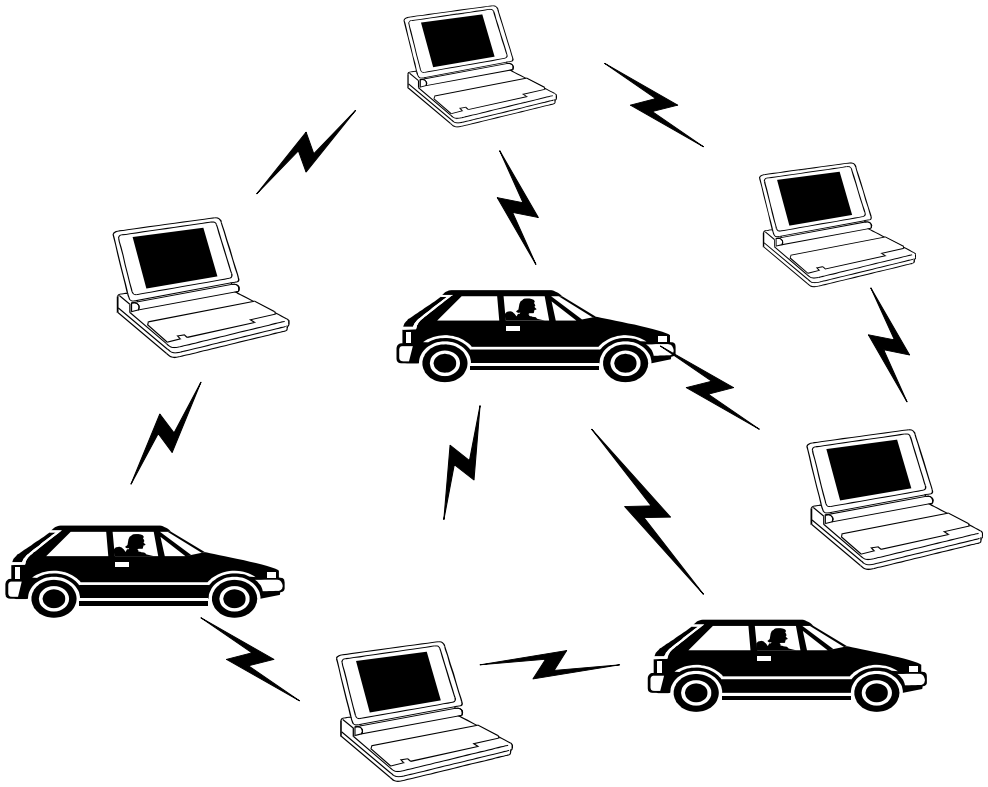

Figura 3 - Rede *Ad Hoc*

2.5.2 Arquitetura celular

Segundo HELAL et al (1999), a arquitetura de rede celular é composta basicamente da divisão de uma área em áreas menores denominadas células. As células possuem a comunicação com a rede fixa através de canais sem fios (Figura 4). Ainda

segundo HELAL et al (1999), um modelo ideal de arquitetura celular seria aquele onde as células seriam divididas em hexágonos e gerenciadas por uma estação base que efetua a comunicação com a unidade móvel.

Um conjunto de estações base será controlado por uma estação base controladora denominada BSC (*Base Station Controller*), a administração dos recursos disponíveis e o gerenciamento de *handoffs*[9] são de responsabilidade das BSC. Na rede fixa encontra-se o MSC *(Mobile Switching Center*) onde é feita a conexão da BSC. As unidades móveis poderão ser registradas em sua HLR *(Home Location Register)* ou na condição de visitante em uma VLR *(Visitor Location Register)*.

Cada unidade móvel tem uma área registrada que é o local onde ela poderá ser encontrada. Os servidores de localização detêm a responsabilidade pela manutenção destes registros de localização, permitindo que se trabalhe com uma trilha de movimentação da unidade.

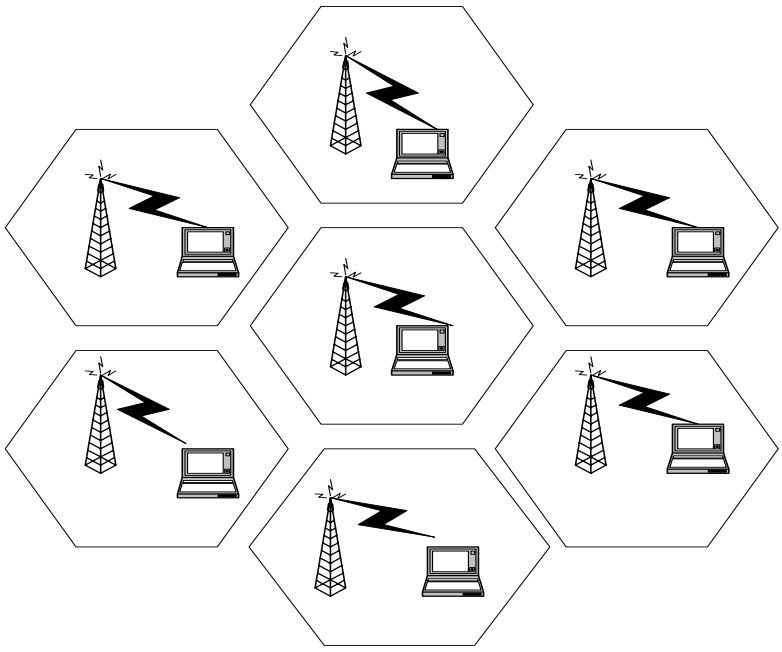

Figura 4 - Arquitetura celular

[9] *Handoff*: processo de movimentação do usuário de uma área de cobertura para outra, localizado especificamente na fronteira de abrangência das duas células.

2.5.3 Wireless LAN

Wireless LAN é uma rede local que permite a comunicação entre as unidades sem a utilização da conexão com fios, a flexibilidade deste tipo de rede torna-se aparente tendo em vista a facilidade da movimentação das unidades.

A comunicação da rede fixa com as unidades móveis é concretizada pela utilização de placas de redes com interface *wireless* em conjunto com os *hub*[10]*s*. Os *hubs* são encarregados de receber e transmitir os pacotes que trafegam a partir das antenas das placas de rede.

2.5.4 Wireless WAN

Segundo MATEUS & LOUREIRO (1998), a arquitetura de rede *Wireless LAN* diferencia-se da arquitetura *wireless LAN* pela área de cobertura do sinal transmitido. O uso de satélites para permitir uma total cobertura geográfica é uma possibilidade neste tipo de arquitetura. Poderá existir a necessidade de interligação entre os satélites utilizados na construção deste modelo de rede, neste caso a ligação entre os satélites será efetuada através da utilização de sinais de microondas. A conexão desta rede espacial será efetivada com a terra através do uso de *gateways*[11], que serão então conectados com a rede pública de telefonia.

2.6 Protocolos Móveis

Apresentaremos a seguir alguns dos principais protocolos para a comunicação sem fio: IP Móvel e *Wireless Aplication Protocol (WAP)*, *IEEE 802.11* e o *Bluetooth.*

2.6.1 IP Móvel

[10] *Hubs* : dispositivo que reúne as linhas de comunicação em um local central, fornecendo uma comunicação comum a todos os dispositivos da rede.
[11] *Gateways*: dispositivo que permite a comunicação de redes de computadores operando em protocolos distintos.

No ambiente da computação móvel a correta entrega dos pacotes de rede apresenta-se com uma característica singular: o endereço da unidade móvel para a entrega de pacotes (endereço IP) pode sofrer alterações conforme o ponto de conexão da própria unidade. Para que aconteça um perfeito roteamento das entregas dos pacotes foi sugerido o uso do protocolo IP Móvel, este protocolo resolve a característica da mobilidade do endereço da unidade móvel.

O IP móvel permite que a unidade móvel trabalhe com dois números IPs: o *Home addres* associado a sua estação origem que é fixo e o *care-of address* que é associado a cada ponto de acesso à rede.

O *home address* é utilizado para que as aplicações e usuários se comuniquem com o usuário móvel. Segundo MATEUS & LOUREIRO (1998), quando a unidade não estiver conectada a seu endereço fixo, deverá existir um mecanismo que permita o envio e o recebimento de todos os pacotes endereçados à unidade móvel no seu ponto de acesso à rede dado pelo endereço *care-of address.* A responsabilidade pela execução desta tarefa é creditada ao agente *home agent* .

Na movimentação da unidade móvel acontece uma alteração no ponto de acesso à rede fixa, neste caso a unidade móvel registra o novo *care-of address* com o seu *home agent* que será o encarregado da entrega dos pacotes para o novo endereço. Para que isto seja possível é necessário que se altere o campo de destinatário no pacote IP de *home address* para *care-of address*. Quando o nó de endereço *care-of address* recebe um pacote que deve ser enviado para a unidade móvel, ele aplica a transformação reversa. Esta operação consiste em colocar como endereço de destinatário o *home address* do cliente, esta tarefa é executada pelo agente *foreign agent*.

2.6.2 Protocolo de aplicação sem fio

O protocolo WAP (*Wireless Aplication Protocol*) provê um padrão universal para o acesso ao conteúdo da Internet e serviços relacionados com a computação sem fio ou com a computação móvel. Esta tecnologia utiliza transmissão digital para garantir a compressão de dados. Através da utilização do *WAP* é possível fazer uso de aplicações como acesso a *e-mails,* navegação na Internet, transações móveis, comércio eletrônico etc. A linguagem utilizada pelo *WAP* é a *WML* que é uma linguagem do padrão

XML(Extensible Markup Language)[12] , lida e interpretada por um *microbrowser* instalado no dispositivo móvel que contém o *WAP*. ITO (2001)

2.6.3 Padrão IEEE 802.11

Em 1999, o *IEEE (Institute of Electrical and Eletronics Engeneers)* definiu uma norma para redes locais sem fio chamada *"Wireless LAN Médium Access Control and Physical Layer Specifications"* , este padrão especifica as camadas física e de controle de acesso ao meio. Segundo RUBINSTEIN & RESENDE (2002), os componentes do padrão *IEEE 802.11* interagem no sentido prover uma rede local sem fio oferecendo suporte à mobilidade das estações. O *BSS – Basic Service Set* (Conjunto básico de servicos) é o fundamento do padrão *IEEE 802.11*. Um *BSS* pode ser definido como um conjunto de estações que estão sobre o controle direto de uma única função de coordenação, a qual determina quando uma estação pode transmitir e receber dados. Dentro do padrão existem dois tipos de redes sem fio: *ad-hoc* e infra-estruturada. Na rede *ad-hoc* é possível que aconteça a comunicação entre as estações localizadas dentro de uma mesma *BSS* sem a interferência de um ponto de acesso centralizado. Já na rede infra-estruturada é utilizado um ponto de acesso responsável pela funcionalidade da rede. Com o objetivo de aumentar a cobertura deste tipo de rede é possível a utilização de vários pontos de acesso interligados através de um sistema de distribuição.

2.6.4 Bluetooth

O padrão *Bluetooth* surgiu em 1998 de um consórcio formado por cinco empresas (Ericsson, Nokia, IBM, Intel e Toshiba) interessadas em desenvolver um padrão de comunicação entre dispositivos sem a utilização de cabos físicos. A grande limitação desta tecnologia encontra-se na área de cobertura da comunicação, atualmente em torno de 10 metros.

Os Dispositivos Bluetooth operam na faixa ISM (*Industrial, Scientific, Medical*) centrada em 2,45 GHz. Nos Estados Unidos, a faixa ISM varia de 2400 a 2483,5 MHz. No Japão, de 2400 a 2500 MHz. Para a operação de dispositivos Bluetooth em países

[12] *XML*: linguagem de programação de hipertexto que fornece um formato que permite descrever dados

como Espanha e França são necessários alguns ajustes pois a largura de banda e a localização da faixa ISM diferem.

A comunicação entre os dispositivos Bluetooth é feita através do estabelecimento de um canal FH-CDMA (*Frequency Hopping - Code-Division Multiple Access*). O transmissor envia um sinal sobre uma série de freqüências de rádio, um receptor, alternando entre as freqüências, capta o sinal. A mensagem é totalmente recebida apenas se o receptor conhecer a série de freqüências na qual o transmissor alternará para enviar o sinal.

2.7 Modelos de Comunicação na Computação Móvel

Levando-se em conta as características específicas e ainda as restrições da computação móvel, PITOURA & SAMARAS (1998) apresentaram alguns modelos de comunicação na computação móvel: *client/server, peer-to-peer* e *mobile agents*. Um resumo desta proposta será apresentado nos próximos itens.

2.7.1 Modelo client/server

O modelo client/server na forma tradicional pode ser descrita como uma aplicação executando em um sistema computacional, chamado de cliente, que requisita um serviço de outra aplicação executando em outro sistema computacional, chamado de servidor (Figura 5).

Em um ambiente móvel, a unidade móvel atua como o cliente requisitando serviços de servidores localizados na rede fixa. Em alguns casos, a funcionalidade e os dados são distribuídos através de vários servidores localizados na rede fixa, o que permite a comunicação entre os mesmos para o atendimento de possíveis requisições de clientes.

Em outros casos um servidor pode estar replicado por vários *sites*[13] da rede fixa, na tentativa de promover um aumento da disponibilidade do sistema em caso de falhas de rede ou mesmo falhas de *sites*. SATYANARAYANAN (1996) fala sobre uma

---

estruturados.

extensão do modelo *client/server* onde os papéis de cliente e servidor são provisoriamente burlados. Um exemplo desta situação pode ser encontrado durante os períodos de desconexão das unidades móveis, neste momento o cliente que se encontra desconectado necessita emular a funcionalidade de um servidor para continuar com a operação em curso. Duas extensões do modelo *client/server* serão apresentadas nas próximas seções.

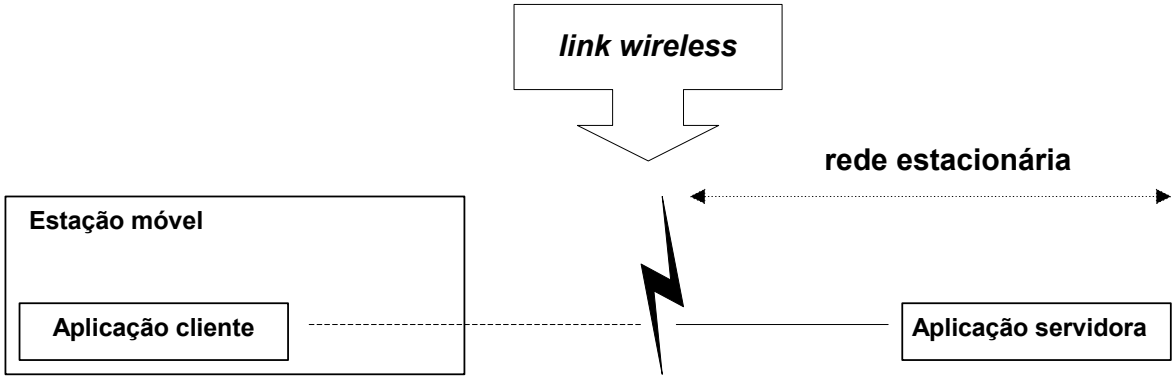

Figura 5 - Modelo *Client-server*

2.7.1.1 Modelo client/agent/server

O modelo client/agent/server utiliza uma infra-estrutura de comunicação baseada no enfileiramento de mensagens do cliente móvel para o agente, e do agente para o servidor. O agente é apenas um representante do cliente na rede fixa. Os agentes dividem a interação entre os clientes móveis e servidores fixos em duas partes, uma parte entre o cliente e o agente e outra parte entre o agente e o servidor (Figura 6).

Para trabalhar com desconexões, um cliente móvel pode submeter suas requisições para o agente e aguardar os resultados após o restabelecimento da conexão. Durante o período de desconexão, todas as requisições para o cliente serão enfileiradas no agente para a conseqüente transferência após a re-conexão. Da mesma forma o agente pode ser utilizado para preservar o tempo de vida da bateria. Após submeter a

[13] *Sites*: servidores localizados em determinados nós de uma rede de computadores.

requisição para o agente, o cliente entra em estado de repouso e aguarda o envio da resposta do agente.

A localização do agente na rede fixa depende do papel assumido no ambiente. Posicionar o agente na estação base *(base station)* pode trazer algumas vantagens quando o agente atua como representante das unidades móveis sob a sua cobertura, por outro lado poderá surgir a necessidade do agente se movimentar juntamente com a unidade móvel. No caso de agentes para serviços específicos pode fazer mais sentido colocá-los próximo da maioria dos clientes ou do servidor.

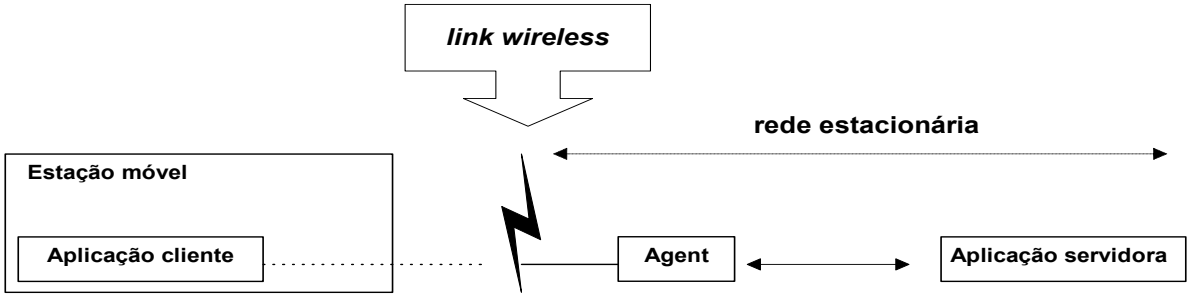

Figura 6 - Modelo *client/agent/server*

2.7.1.2 Modelo client/intercept/server

Para resolver as limitações do modelo *client/agent/server*, SAMARAS & PITSILLIDES (1997) e HOUSEL et al. (1997) propuseram o desenvolvimento de um agente *client-side*. Este agente irá rodar no dispositivo móvel paralelamente ao agente do modelo *client/agent/server* que roda dentro da rede fixa (Figura 7). O agente *client-side* intercepta as requisições dos clientes e juntamente com o agente *server-side* constrói otimizações para reduzir a transmissão de dados sobre o enlace *wireless.*

Este modelo oferece uma maior flexibilidade no manuseio de desconexões, um *cache* local pode ser mantido junto do agente *client-side.* Este *cache* pode ser utilizado para satisfazer as requisições de clientes durante os períodos de desconexão. Erros de cache podem ser enfileirados pelo agente *client-side* com o propósito de serem resolvidos após a re-conexão. Da mesma forma, as requisições para o cliente podem ser

serializadas juntamente com o agente *server-side* e transferidas para o cliente após a re-conexão.

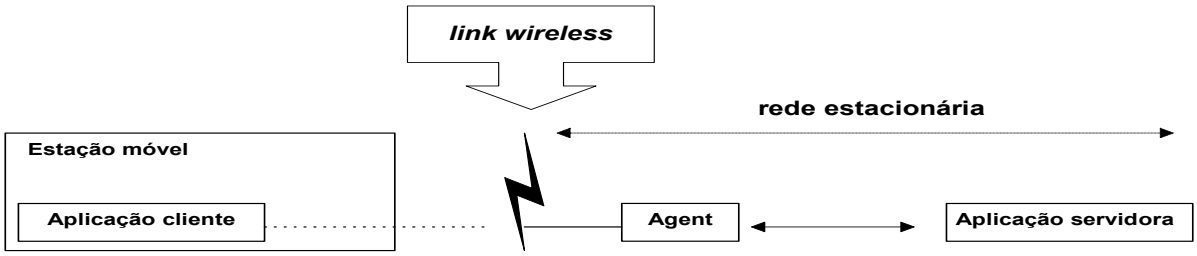

Figura 7 - Modelo *client/intercept/server*

2.7.2 Modelo par-par

No modelo par-par não existe a distinção entre clientes e servidores, cada estação tem a funcionalidade total tanto de um servidor quando de uma estação cliente. Este modelo aplica-se com mais sucesso nos casos em que existe uma forte conexão entre as unidades móveis e freqüentes desconexões com o servidor. No entanto os custos de estabelecimento de comunicação podem ser elevados, necessitando então de unidades móveis com considerável capacidade de processamento.

2.7.3 Modelo de agentes móveis

Agentes móveis são processos disparados de um computador de origem para executar uma determinada tarefa. CHESS et al. (1995). Cada agente móvel possui seus próprios dados, instruções e estados de execução, desta forma o paradigma de agente móvel estende o mecanismo de comunicação RPC para o qual uma mensagem enviada por um cliente é apenas uma chamada de *procedure*.

Após a sua submissão, o agente móvel processa de forma independente e autônoma do cliente que o disparou, o agente móvel pode transportar-se para outros servidores, criar novos agentes, ou ainda interagir com outros agentes. Depois de completa a execução total, o agente móvel entrega os resultados para o cliente ou servidor que o emitiu.

O modelo de agente computacional suporta a operação desconectada. Durante uma breve conexão o cliente móvel submete um agente e então desconecta, o agente processa independentemente para concluir a tarefa solicitada. Quando a tarefa estiver completa o agente aguarda uma nova conexão para então repassar os resultados da tarefa que lhe foi solicitada. Da mesma forma, um agente móvel pode ser carregado da rede fixa para dentro de uma unidade móvel antes da desconexão, o agente age como um representante para a aplicação permitindo interação com o usuário mesmo durante o período de desconexão.

2.8 Considerações

Durante este capítulo foram apresentadas as principais características de um ambiente de computação móvel. Suas particularidades e necessidades foram abordadas no discorrer do texto. Arquitetura, histórico, protocolos, modelos e componentes da computação móvel foram estudados com a intenção de suprir o conhecimento necessário para que se possa entrar no conceito da utilização de bancos de dados em ambientes de computação móvel.

CAPÍTULO III

Bancos de Dados em Ambientes de Computação Móvel

A utilização de bancos de dados em ambientes de computação móvel torna-se particularmente interessante tendo em vista a possibilidade de manipulação de dados tanto sob a forma conectada quando sob a forma desconectada. A mobilidade trouxe um novo impulso na utilização dos bancos de dados. A inserção de novas informações nas unidades móveis nos locais mais remotos, sem a necessidade do estabelecimento de uma conexão com a rede estacionária, abre o caminho para a definição de novas aplicações. A manutenção de informações localmente permite um alto grau de independência da unidade móvel em relação à rede fixa, traduzindo-se também em um incremento na mobilidade da unidade.

Neste capítulo será feita uma descrição de uma arquitetura genérica de um ambiente de bancos de dados móveis. Também serão abordadas algumas características específicas de banco de dados em ambiente de computação móvel: formas de acesso aos dados, gerência de transações, processamento de consultas, replicação de dados e recuperação de falhas. Após a apresentação destas características faremos uma abordagem nos exemplos de aplicações de banco de dados móveis.

3.1 Arquiteturas de bancos de dados móveis

Neste item serão apresentadas algumas arquiteturas de um ambiente de banco de dados móvel. As arquiteturas apresentadas a seguir foram identificadas na literatura levando-se em conta o estudo de PITOURA & SAMARAS (1998), que apresenta algumas opções de modelos da computação móvel.

3.1.1 Arquitetura cliente - servidor

Na tradicional arquitetura cliente-servidor encontramos uma aplicação executando em uma unidade cliente, esta unidade emite solicitações de dados para o servidor central. O servidor central, no caso de um ambiente de banco de dados móvel estará localizado na rede estacionária. Com a intenção de aumentar a disponibilidade dos dados a partir do servidor, poderá acontecer a replicação do banco de dados por vários outros servidores da rede fixa. As unidades móveis irão trabalhar no papel de clientes deste ambiente, disparando suas requisições diretamente para os servidores da rede estacionária. Para que o ambiente de bancos de dados móveis possa ser atendido na totalidade (mobilidade e desconexão), é necessário que se façam adaptações na tradicional arquitetura cliente-servidor.

3.1.1.1 Arquitetura cliente-agente-servidor

Esta arquitetura surge de otimizações na arquitetura cliente-servidor tradicional, a inclusão de um agente que representa o cliente na rede fixa, proporciona a migração de carga de processamento da unidade móvel para o servidor da rede estacionária. A unidade pode emitir, através do agente, as requisições que necessitar e entrar em modo de espera (doze). A responsabilidade de alcançar as respostas para as requisições da unidade móvel fica à cargo do agente. No momento da reativação da unidade móvel o agente será encarregado de efetuar a entrega das informações solicitadas através da requisição anterior. Esta arquitetura ainda não oferece suporte para o trabalho desconectado. Quando a unidade móvel estiver desconectada, as requisições direcionadas para a estação cliente serão enfileiradas pelo agente localizado na rede fixa e, após a reconexão serão enviadas para a unidade móvel.

3.1.1.2 Arquitetura cliente-interceptador-servidor

A arquitetura cliente-interceptador-servidor resolve a impossibilidade do trabalho desconectado da arquitetura cliente-agente-servidor. A figura do agente foi “replicada” para o lado cliente (agente-lado-cliente) além do lado servidor (agente-lado-servidor).

Os agentes interceptam as requisições clientes e servidoras trabalhando de forma cooperativa no atendimento de cada requisição.

A possibilidade de trabalhar com desconexões vem do fato de se poder fazer uso de um *cache* no agente-lado-cliente. Durante o período de desconexão, as unidades móveis terão suas requisições atendidas através da utilização do *cache* mantido pelo agente-lado-cliente. Se a informação não estiver disponível no cache da unidade móvel, esta requisição será enfileirada pelo agente-lado-cliente e no momento da reconexão será entregue aos cuidados do agente-lado-servidor.

3.1.2 Arquitetura par-par

Na arquitetura par-par não existe a distinção entre as figuras de servidores e clientes, cada estação tem a funcionalidade total tanto de servidor quanto de estação cliente. Neste caso a desconexão pode se revelar em um grande problema, pois a indisponibilidade de um *site* pode comprometer uma transação por completo. Para amenizar o problema da desconexão seria necessário introduzir a figura de agentes, agindo como representantes das unidades no caso de desconexões.

3.1.3 Agentes móveis

A arquitetura baseada em agentes móveis trabalha com processos disparados a partir de uma unidade para executar determinada tarefa. Cada agente móvel possui instruções, dados e um estado de execução. O agente trabalha de forma autônoma e independente da aplicação que o criou. Possui a capacidade de cooperação com outros agentes no intuito de resolver as tarefas para as quais foi incumbido, pode se movimentar entre as estações para a satisfação das requisições. Esta arquitetura suporta a desconexão, durante um breve período de desconexão a unidade pode disparar o agente e então seguir desconectada, o agente terá a responsabilidade de buscar as informações e posteriormente entregar para a estação cliente.

3.2 Principais paradigmas de acesso aos dados

O acesso aos dados em plataformas móveis apresenta uma série de limitações. A restrição na capacidade de armazenamento de dados e a possibilidade de desconexão destas plataformas necessitam de uma forma de acesso que diminua o tráfego no canal de comunicação. As principais proposições para trabalhar com estas limitações são a difusão de dados (*broadcast*) e o *caching* de dados. Estas duas formas de acesso aos dados serão consideradas a seguir.

3.2.1 Difusão de dados *(broadcast)*

Segundo BARBARÁ (1999), difusão (disseminação) de dados é a entrega de informações de um conjunto de provedores para um grande número de clientes. Este processo de entrega de informações é caracterizado pela assimetria nas comunicações, a largura da banda da rede é maior no sentido servidor – cliente que no sentido cliente – servidor. O modelo de envio de informações para um conjunto de clientes é chamado de modelo *push-based*. Neste modelo, um dos grandes problemas a serem solucionados é a decisão sobre quais os itens de dados que serão disseminados. Uma possível solução para este problema pode ser prover o cliente com a possibilidade de construir perfis (*profiles*) abrangendo apenas os dados que lhe são de interesse. Estudos neste sentido podem ser encontrados em TERRY et al. (1994).

O uso de uma arquitetura com disseminação periódica de informações foi proposta em ACHARYA et al. (1995), surgindo então o conceito de *broadcast disks*. O uso de *broadcast disks* permite a construção de uma hierarquia de memória onde o nível mais alto contém alguns poucos itens de dados que são disseminados em uma alta freqüência. Nos níveis inferiores encontram-se mais itens que são disseminados em uma freqüência menor. Desta forma pode ser estabelecida uma relação entre tempo de acesso para dados de alta prioridade e de baixa prioridade. A figura 8 mostra três diferentes organizações de difusão de itens de igual tamanho. A seqüência de letra (a) da figura 8 somente considera os dados a serem transmitidos, sem se preocupar com a probabilidade de acesso às informações. Esta forma de transmissão é conhecida como transmissão plana de dados. Nas letras (b) e (c) o item de dados A tem o dobro da freqüência de transmissão dos itens B e C. Na letra (b) as transmissões de A estão

agrupadas porém de forma aleatória, enquanto que na letra (c) já existe um padrão regular no intervalo de transmissão de qualquer item de dados. A seqüência em (c) sugere uma difusão de multi-discos sendo que A está armazenado em um disco com velocidade duas vezes maior que as dos discos onde se encontram os itens B e C.

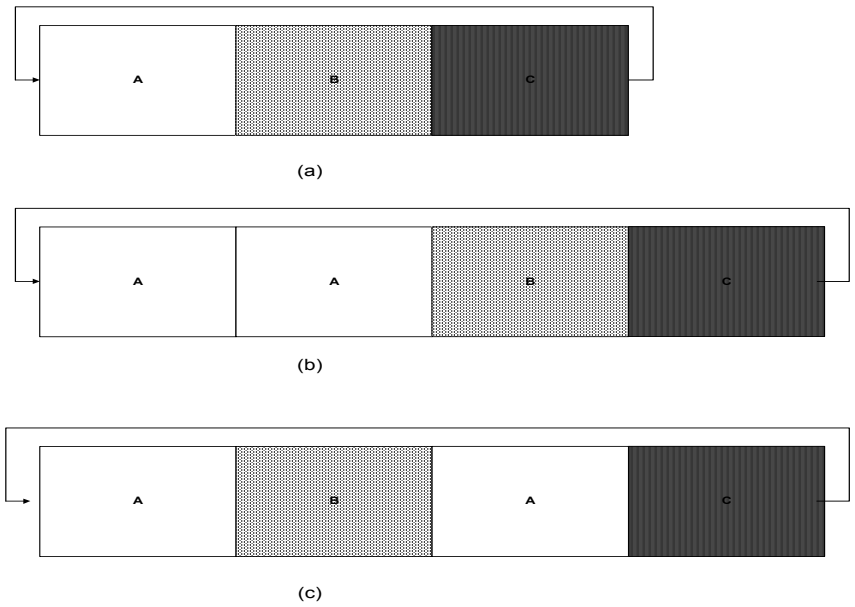

Figura 8 - Organizações de difusão de dados

As necessidades de cada cliente são dinâmicas, portanto é necessário que se estabeleça algum mecanismo para a compensação dos desvios na informação disseminada. Em ACHARYA et al. (1997) podemos encontrar a integração das arquiteturas *pull-based* e *push-based*. Em uma operação do tipo *pull-based* os clientes podem requisitar os itens de dados através do envio de mensagens para o servidor, que então retorna a informação para os clientes. A proposta trabalha com a possibilidade de utilização de dois canais de comunicação independentes: *frontchannel* e *backchannel*. O canal *frontchannel* é utilizado para as operações do tipo *push-based*, enquanto que o canal *back-channel* é utilizado somente para as operações tipo *pull-based*. A disponibilidade de largura de banda da rede é compartilhada entre os dois canais conforme a necessidade de comunicação

Em BARBARÁ & IMIELINSKI (1994), os autores apresentam uma nova forma de envio de mensagens de invalidação sobre uma largura de banda de rede limitada.

Utilizando *Invalidation Reports* (*IRs*), o servidor pode avisar os clientes sobre mudanças nos itens de dados que fazem parte do seu *cache* local. A transmissão desta informação pode acontecer sob a forma de uma lista de identificadores dos itens que foram alterados desde que o último *invalidation report* (relatório de invalidação) foi enviado.

Uma extensão desta proposta foi apresentada pelos mesmos autores em 1995, os valores em *cache* nos clientes seriam compostos de *quasicopies* dos dados no servidor. Um *quasicopie* é um valor mantido em cache com um desvio controlado de valor com relação ao mantido no servidor. Através do uso de *quasicopies* os relatórios de invalidação podem ser mais curtos, economizando tempo e largura de banda na transmissão dos mesmos.

BESTRAVOS & CUNHA (1996) propõem uma série de organizações de *broadcast disks* adequados para o uso em ambientes do tipo tempo-real. Na proposta foram apresentadas três organizações: *flat, rate monotonic* e *slotted rate monotonic.* Na organização do tipo *flat,* o servidor envia a união de itens de dados necessários para os seus clientes de uma maneira periódica. Esta organização consome a maior largura de banda em comparação com as outras duas organizações. Na tentativa de solucionar este problema, a organização do tipo *rate monotonic* dissemina cada item em uma taxa inversamente proporcional à sua restrição de tempo. Desta forma, cada item representa um disco "girando" a uma taxa recíproca à restrição de tempo associada ao item de dados específico. Uma mesclagem entre as duas organizações anteriores é o que podemos encontrar na organização do tipo *slotted rate monotonic.* Nesta organização os itens de dados são agrupados de forma que possam ser transmitidos dentro do mesmo *broadcast disk*. Este agrupamento é possível pelo particionamento do conjunto de dados dentro de "caixas", itens de dados da mesma "caixa" compartilham o mesmo *broadcast disk* e possuem atrasos idênticos.

Em IMIELINSKI et al. (1994), encontramos a situação na qual o usuário está interessado em alguns itens de dados identificados por alguma chave. Uma forma de índice será transmitida juntamente com os itens de dados, este índice conterá a informação de quando o item será transmitindo. Desta forma, o cliente necessita sintonizar apenas no canal que irá conter a informação necessária. Na maior parte do tempo o cliente poderá permanecer no modo adormecido (*doze*), proporcionando economia de energia da estação móvel.

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

3.2.2 Caching

A utilização de memória *cache* para a execução de consultas e operações de atualizações de itens de dados contribui para a diminuição da utilização dos recursos de processamento da unidade móvel. O cliente móvel pode efetuar o armazenamento das informações necessárias para sua operação e trabalhar livremente sob a forma desconectada. A consistência das informações deveria permitir que as transações realizadas no ambiente pudessem utilizar o conceito de serialização[14], neste ponto é necessário um relaxamento deste conceito, no intuito de diminuir a utilização dos recursos de rede.

A maior dificuldade no trabalho de montagem do *cache* de dados da unidade móvel está em selecionar quais os itens que farão parte do *cache* local. Uma tentativa de resolver este problema foi proposta por ARAUJO & FERREIRA (2000), que propõem a classificação dos dados quanto ao uso, tornando possível a definição de duas regiões de dados: região quente e região fria.

Na região quente estaria localizado o conjunto de dados com utilização mais freqüente, enquanto que na região fria os dados com utilização menos freqüente estariam localizados. A natureza destas regiões é bastante dinâmica, ou seja: os dados

[14] Serialização: a execução concorrente de transações mantém o banco de dados em um estado concorrente da mesma maneira que se as transações fossem executadas em uma determinada ordenação.

podem mudar a sua classificação em conformidade com o uso dos mesmos. A natureza das operações sobre os dados mantidos em *cache* serviu como parâmetro para que DESPANDE et. Al (1998) os dividisse em três categorias:

a) Dados somente para consulta: não são efetuadas atualizações nos dados mantidos em *cache*, não havendo a necessidade de reintegração das informações no servidor.

b) Dados não concorrentes: atualizações são permitidas, mas são mantidos bloqueios nos dados do servidor, o que impede a ocorrência de conflitos.

c) Dados concorrentes: os dados podem ser alterados tanto no servidor quanto nas unidades móveis, no intuito de resolver conflitos de atualização, informações sobre as operações de atualização de dados também devem ser mantidas em *cache*.

3.3 Gerenciamento de transações móveis

No intuito de manter a consistência do banco de dados, a atualização e recuperação de informações são efetuadas através de transações. Em um ambiente de banco de dados móvel o conceito de transação sofre algumas alterações com relação à sua definição mais formal. A característica da mobilidade abre a possibilidade da execução de uma transação móvel acontecer de forma totalmente distribuída sobre várias estações de suporte.

3.3.1 Modelos de transações móveis

Neste item apresentaremos os principais modelos de transações móveis, com base no estudo de ADIBA et. al (2001).

Clustering: neste modelo o banco de dados é dinamicamente dividido em *clusters,* cada um deles agrupando dados semanticamente relativos ou de localização aproximada. Um *cluster* pode ser distribuído em diversos *hosts* com forte conexão. No momento de desconexão uma estação móvel inicia a construção de um novo *cluster*. Duas cópias de cada objeto são mantidas no banco de dados: a versão *strict* e a versão

weak, a versão *strict* precisa ser globalmente consistente. Já a versão *weak* poderá trabalhar com algum nível de inconsistência desde que seja localmente consistente. As transações neste modelo são classificadas em *strict* e *weak.* As transações *strict* só irão acessar as cópias do tipo *strict*, e serão executadas quando os *hosts* estiverem fortemente conectados. As transações *weak* só terão acesso aos dados do tipo *weak* e acontecerão nas unidades móveis enquanto estiverem desconectadas.

Two-tier replication: neste modelo existirão uma versão master para cada item de dados e várias cópias replicadas, além disto dois tipos de transações estarão presentes neste modelo: *base* e *tentative*. As transações *base* serão executadas sobre cópias *master* de dados, enquanto que as transações *tentative* farão seu trabalho sobre as cópias replicadas (cópias locais).

Pro-motion: este modelo permite o suporte para o modo desconectado. O modelo apresenta o conceito de *compacts,* que são a unidade básica de controle e *cache.* Encapsulado no próprio *compact* encontram-se as informações necessárias para o seu gerenciamento.

Reporting: neste modelo uma transação móvel é estruturada como um conjunto de transações, algumas delas executadas nas unidades móveis. A utilização de uma estação localizada na rede estacionária é obrigatória neste modelo, pois poderia comportar uma parte do processamento ou uma parte do estado do banco de dados. Uma transação do tipo *reporting* pode compartilhar seu resultado parcial com a transação de origem em qualquer ponto durante a sua execução. Uma transação do tipo *co-transaction* pode ter a sua execução suspensa de forma a permitir a execução de outra transação.

Semantics-based: este modelo trabalha com a fragmentação de objetos como uma solução para operações concorrentes e limitação da capacidade de armazenamento nas unidades móveis. Esta proposta sugere a divisão de objetos complexos em fragmentos menores e similares semanticamentes. Cada fragmento poderá ser alocado em *cache* separadamente e gerenciado independentemente. As transações podem acontecer totalmente nas unidades móveis, através do uso dos fragmentos, no momento da reconexão um processo de reconciliação irá acontecer na tentativa de "remontar" os objetos divididos anteriormente.

Prewrite: neste modelo é proposta a introdução de uma nova operação além das operações tradicionais de escrita, a operação de *prewrite* possibilita que os dados fiquem visíveis para uma operação de *precommit* antes do *commit* das transações móveis. As atualizações permanentes no banco de dados são concretizadas mais tarde pela operação *write* na confirmação.

Kangaroo transactions: propõe um modelo de transação móvel que prioriza a mobilidade da unidade móvel durante a execução da transação. A execução da transação móvel é coordenada pela estação base para a qual a unidade móvel estiver alocada. A captura da mobilidade da unidade móvel é possível pela divisão da transação original em duas novas transações.

MDSTPM (Multidatabase transaction Processing Manager): esta proposta trabalha com um ambiente para suportar submissões de transações de unidades móveis em um ambiente de banco de dados múltiplo. Um *MDSTPM* é assumido em cada *host* no topo de um DBMS local. O processamento local é responsabilidade do DBMS local. O MDSTPM coordena a execução de transações globais, trabalhando com os agendamentos e coordenando os processos de confirmação de transações. Desta forma, depois que uma unidade móvel submete uma transação global, ela pode desconectar e executar outras tarefas sem que haja a necessidade de esperar para que aquela transação móvel seja confirmada. O *host* coordenador irá executar este processo em benefício da unidade móvel.

3.3.2 Requisitos das transações móveis

Segundo WALBORN & CHRYSANTHIS (1998), para melhor descrever o processamento local de uma transação em uma unidade móvel, é melhor generalizar as propriedades ACID[15] e falar sobre **visibilidade**, **consistência**, **permanência** e **recuperação**.

[15] ***ACID***: conjunto de propriedades que garantem a integridade do banco de dados. ***A****tomicidade*: cada transação é tratada como uma unidade de operação, ou todas as ações da transação são executadas ou nenhuma delas será. ***C****onsistência*: uma transação é um programa que mapeia um banco de dados de um estado consistente para outro igualmente consistente. ***I****solamento*: uma transação que está sendo executada não pode revelar os seus resultados para outra transação concorrente até que seja confirmada na totalidade. ***D****urabilidade*: uma vez que a transação tenha sido efetivada, seus resultados devem ser permanentes e não podem ser apagados do banco de dados.

O conceito de visibilidade diz respeito à possibilidade de uma transação ver os efeitos causados a um conjunto de itens de dados por uma outra transação. Permitir que novas transações vejam mudanças não confirmadas poderá resultar em dependências e cancelamentos não desejados. Desde que nenhuma operação de atualização de dados em uma unidade móvel desconectada possa ser incorporada no banco de dados do servidor, as transações subseqüentes que utilizam os mesmos itens de dados não poderão acontecer até que a reconexão aconteça e a transação principal seja confirmada.

Ainda segundo WALBORN & CHRYSANTHIS (1998), para que um sistema de gerenciamento de banco de dados suporte o processamento de transações em uma unidade móvel e desconectada, deve incluir as seguintes características:

a) uma forma de comunicação com os servidores de banco de dados estacionários;

b) um subsistema *stand-alone* de processamento de transações para executar na unidade móvel;

c) um método para reconciliar as transações processadas durante o período de desconexão, com o servidor de banco de dados;

d) sistemas de *logging, checkpoint* e recuperação de falhas suficientes para diminuir o custo das falhas;

e) uma forma para gerenciar a replicação e consistência de dados necessários.

Tal sistema deveria prover a serialização quando necessária, mas da mesma forma deveria suportar critérios de correção mais relaxados e diferentes níveis de isolamento das transações, quando necessário.

3.4 Controle de concorrência

O controle de concorrência tem a função de garantir que as transações concorrentes enviadas para processamento no banco de dados sejam executadas em isolamento. A utilização de bloqueios garante o isolamento das transações, pois o acesso ao item de dados só será permitido àquela transação que obtiver o bloqueio solicitado.

Segundo MOLINA et al. (2001), a garantia de que a execução das transações concorrentes preserve a consistência do banco de dados é oferecida pelo controle de concorrência. O escalonamento das ações de transações tem por objetivo garantir que as transações levem o banco de dados de um estado consistente para outro igualmente consistente. Este escalonamento de ações trabalha com a compatibilidade entre os diferentes tipos de bloqueios, bem como com a ordem em que estas ações serão executadas. Um escalonamento será considerado **serial** caso garanta que todas as ações de uma determinada transação serão executadas antes das ações de uma segunda transação.

Basicamente todos os tipos de bloqueios nascem a partir de dois tipos principais: **bloqueio compartilhado (S)** e o **bloqueio exclusivo (X)**, esta é uma classificação simplória, mas servirá ao propósito de facilitar o entendimento do processo. Ao receber um bloqueio do tipo compartilhado sobre um item de dados, uma transação garante a possibilidade de leitura, mas não de escrita sobre o item em questão. Ao receber o bloqueio do tipo exclusivo terá o direito tanto de leitura quando de escrita sobre o item de dados. A serialização das ações das transações fará uso da compatibilidade entre os tipos de bloqueios, identificada pela matriz de compatibilidade. A figura 9 apresenta uma possível representação desta matriz para os bloqueios compartilhado e exclusivo.

	Compartilhado (S)	**Exclusivo (X)**
Compartilhado (S)	***Compatível***	***Não compatível***
Exclusivo (X)	***Não compatível***	***Não compatível***

Figura 9 - Matriz de compatibilidade

Um protocolo bastante utilizado na manutenção das transações é o chamado protocolo de bloqueio em duas fases (*two phase locking protocol*). Segundo MOLINA et al. (2001), este protocolo garante que em toda transação todas as solicitações de bloqueios precedem às solicitações de desbloqueio. Em SILBERSCHATZ et al (1999), o protocolo 2PL é definido possuindo duas fases, as fases de emissões de solicitações de bloqueios e desbloqueios exigidas pelo protocolo. Na **fase de expansão** a transação poderá apenas obter os bloqueios, não sendo permitida a liberação dos bloqueios nesta

fase. Já na **fase de encolhimento,** a transação poderá apenas liberar os bloqueios obtidos, não poderá, no entanto, obter um novo bloqueio.

Segundo PITOURA & BHARGAVA (1995), as propostas de controle de concorrência para ambientes de bancos de dados distribuídos precisam ser adaptadas para o ambiente móvel, de modo a atender os seguintes requisitos:

- Oferecer suporte à operação desconectada das unidades,
- Suporte para as necessidades de consumo de largura de banda de rede reduzida.
- Adaptação à instabilidade das conexões.
- Suporte à mudança de localização das unidades móveis.

Na proposta de PITOURA & BHARGAVA (1995), os dados semanticamente relativos ou com localização aproximada, são agrupados para formar um *cluster*. A consistência total é requerida para os itens de dados internos ao *cluster*, níveis de consistência são permitidos para os dados replicados em diferentes *clusters.* Os autores consideram os *clusters* como unidades de dados consistentes, desta forma os dados mantidos no interior de um *cluster* são totalmente consistentes, enquanto que entre os *clusters* poderá existir um nível de inconsistência. As unidades móveis podem ser consideradas como *clusters* já que, localmente, mantêm a consistência dos dados. Desta maneira, a operação desconectada de uma unidade móvel é garantida pela utilização das transações do tipo *weak.* As transações que acessam cópias de dados que se encontram dentro do mesmo *cluster* são classificadas em transações *weak read* e *weak write*, utilizadas para operações de leitura e escrita, respectivamente. As operações de leitura e escrita que acessam dados entre *clusters* são classificadas em *strict read* e *strict write*. Os autores representam as operações da seguinte forma:

- *W_Read[x]* – leitura de um item de dados *x* mantido localmente no *cluster.*
- *W_Write[x]* – operação de escrita em um item de dados *x* mantido localmente no *cluster.*
- *S_Read[x]* – operação de leitura de um item de dados *x* escrito pela última operação *strict write*.
- *S_Write[x]* – operação de escrita de um item de dados *x*.

Da mesma forma que na matriz de compatibilidade de bloqueios utilizada nos protocolos tradicionais de controle de concorrência, os autores apresentam uma matriz de conflito entre as transações *weak* e *strict*. A matriz é considerada sobre a operação de duas transações concorrentes (*i* e j) atuando sobre um item de dados *a* mantido em um *cluster k*. A figura 10 representa a matriz apresentada pelos autores, as linhas representam as operações efetuadas pela transação i, enquanto que as colunas representam as operações da transação j. As operações conflitantes estão marcadas pelo " X ".

	$W\_Read_j(a_k)$	$S\_Read_j(a_k)$	$W\_Write_j(a_k)$	$S\_Write_j(a_k)$
$W\_Read_i(a_k)$			**X**	**X**
$S\_Read_i(a_k)$				**X**
$W\_Write_i(a_k)$	**X**		**X**	**X**
$S\_Write_i(a_k)$	**X**	**X**	**X**	**X**

Figura 10 - Matriz de conflito entre as operações *weak* e *strict*

Segundo BARBARÁ & MOLINA (1993), a estratégia básica de controle de concorrência é garantir que as operações de leitura excluam as de escrita, e as de escrita excluam outras operações de escrita. Os autores descrevem este controle de concorrência através dos *quoruns* de leitura e escrita. Os *quoruns* de leitura especificam os *sites* onde bloqueios de leitura precisam ser obtidos quando um objeto é lido, enquanto que os *quoruns* de escrita indicam onde os bloqueios de escrita serão emitidos. Os bloqueios podem ser requisitados de duas maneiras: **pessimista** e **otimista.** Na forma pessimista, os bloqueios são requisitados antes das operações de leitura ou escrita acontecerem. Já na forma otimista, os bloqueios são requisitados quando a transação estiver se preparando para confirmar a sua ação (*commit*).

No estudo de PITOURA & BHARGAVA (1995), os autores apresentam uma proposta de controle de concorrência baseado na utilização de agentes. Os agentes possuem a capacidade de interação entre diversos *sites* com a mínima intervenção do usuário, além do encapsulamento de código e dados. O controle de concorrência em um ambiente baseado em agentes é significativamente diferente de um ambiente distribuído. No ambiente baseado em agente o processamento é descentralizado, não

existe a figura de um gerente global de transações, tendo em vista que os agentes podem ser submetidos a partir de diversos *sites*. Os autores sugerem a criação de um algoritmo baseado em *timestamp* para garantir a serialização global dos processos. Cada agente de aplicação recebe um valor de *timestamp,* que será uma combinação entre os valores do tempo e a identificação do usuário. O valor do *timestamp* será a ordem do agente dentro da serialização global. Os métodos conflitantes serão serializados com base no valor do *timestamp*. Segundo os autores, um método que não comuta (não pode ter a sua execução alternada sem prejuízo da consistência) com outro método já submetido, não poderá ser executado de forma concorrente. Desta forma se um método com um *timestamp* menor que *AT* for recebido, e não for comutável, será abortado. Os métodos comutáveis podem ser executados concorrentemente.

3.5 Processamento de consultas.

O custo de realização de uma consulta em um ambiente móvel deve levar em conta, além das restrições normais (acesso a disco e memória), o consumo de energia, a mobilidade, e o tráfego de informações a partir da unidade móvel. Os resultados de consultas executadas em ambiente móvel podem depender tanto da localização do usuário quando do instante de tempo em que a consulta for emitida. No estudo de IMIELINSKI & BADRINATH (1992) foi apresentado o conceito de consultas com restrições de localização, as restrições poderiam envolver a localização dos usuários de computadores móveis. Alguns exemplos deste tipo de consulta poderiam ser os seguintes:

a) *"encontre o médico mais próximo da minha casa" ;*

b) *"encontre X, Y e Z de forma que eles estejam localizados no mesmo edifício e Y esteja entre X e Z".*

A discussão de uma família de protocolos que integram o *Global Positioning System (GPS)* com o endereço IP com o objetivo de habilitar a criação de serviços dependentes da localização, pode ser encontrado em IMIELINSKI & NAVAS (1996). SPREITZER & THEIMER (1996) descrevem uma arquitetura para gerenciar a informação de localização. Na arquitetura proposta, a informação pessoal é gerenciada

por *agentes usuário*, enquanto que um parcialmente descentralizado serviço de consulta de localização é utilizado para facilitar as operações baseadas em localização do usuário. Existe a figura de um *agente usuário* para cada usuário móvel, o agente coleta e controla todas as informações relativas àquele usuário. As aplicações só possuem o *agente usuário* para receber informações a respeito de cada usuário do sistema. As fontes de coleta de informações para o agente incluem identificadores com dispositivos infravermelhos, GPS, câmeras e sensores de movimento e entrada de dados por parte do usuário.

3.6 Replicação de dados

Através do processo de replicação acontece o envio de cópias de itens de dados mantidos nos servidores da rede estacionária, para armazenamento no *cache* das unidades móveis. Este processo garante a possibilidade de trabalho com a estação totalmente desconectada da rede estacionária. A redundância de informações obtida pela replicação dos itens de dados garante ainda um aumento na confiabilidade do sistema, uma vez que o usuário não depende exclusivamente de um único local para buscar a informação desejada.

3.6.1 Modelos de replicação de dados

Amparados no trabalho de ITO (2001) vamos apresentar, brevemente, alguns dos modelos de replicação de dados mais conhecidos e aplicados atualmente.

3.6.1.1 Replicação *Peer-to-peer*

Sob este modelo de replicação todas as réplicas são iguais, além disso, é possível que a comunicação entre as unidades móveis aconteça sem a intervenção do servidor localizado na rede estacionária. Segundo PITOURA & SAMARAS (1998), o BAYOU é um sistema replicado que utiliza como modelo de replicação o *peer-to-peer*. Este sistema conta com unidades móveis fracamente conectadas e com reduzida capacidade de armazenamento e consistência dos dados. Cada coleção de dados é totalmente

replicada para um número de servidores Bayou, os clientes podem ler e escrever em qualquer cópia residente em qualquer servidor com o qual possa se comunicar.

3.6.1.2 Replicação cliente – servidor

Todas as estações contêm o mesmo conjunto de dados, o gerenciamento da replicação de dados é centralizado no servidor. Neste modelo a atualização dos dados é executada no sentido servidor → cliente, a comunicação entre os clientes só é possível através da intermediação do servidor da rede estacionária.

A arquitetura do sistema CODA consiste de um determinado número de clientes e um menor número de servidores localizados na rede estacionária. Conjuntos de dados são replicados em mais que um servidor, onde as réplicas são mantidas coerentes. Para fins de desempenho e disponibilidade, os clientes podem armazenar dados nos seus *caches* locais, porém estas réplicas são por natureza, menos seguras e completas em relação às réplicas mantidas nos servidores.

3.6.1.3 Modelo WARD (Wide Area Replication Domain)

Este modelo pode ser considerado como uma combinação dos modelos anteriores (*peer-to-peer* e cliente-servidor). Um WARD é um conjunto de estações próximas conectadas e pertencentes a uma mesma área geográfica, alguns parâmetros devem ser considerados para a construção de WARDs, por exemplo: localização geográfica, largura de banda, custos e modelos de compartilhamento. O WARD *master* possui características semelhantes às do servidor do modelo cliente-servidor. Com exceção do WARD *master* todos os WARDs são pares iguais e passíveis de serem habilitados para a condição de *master* a qualquer momento, quando então poderão realizar operações de sincronização direta entre os seus membros.

3.7 Recuperação de falhas em transações de bancos de dados móveis

O ambiente da computação móvel, devido a sua característica da mobilidade, está mais sujeito à ocorrência de falhas no processamento das suas transações, que em um ambiente estático. Segundo ALONSO & KORTH (1993), algumas garantias deveriam ser supridas por um sistema de banco de dados móvel na ocorrência de uma falha anunciada do sistema:

a) O processamento de uma transação poderia ser transferido para uma estação da rede estacionária sem que nenhuma solicitação do usuário seja necessária;

b) Dados remotos podem ser "baixados" em antecipação a desconexão já prevista, para que o processamento da transação possa continuar na unidade desconectada;

c) Registros de *log* poderiam ser transferidos de uma estação móvel para um computador da rede estacionária.

d) A unidade móvel poderia declarar-se "desligada" removendo-se do conjunto de protocolos de distribuição de informação que estiver participando.

A capacidade e estabilidade de armazenamento das unidades móveis é um aspecto que tem relação direta com a recuperação de falhas em ambientes de bancos de dados móveis. A possibilidade de transferência dos registros de *log* da unidade móvel para uma estação da rede fixa é um ponto crucial para a capacidade de recuperação de falhas.

Quando a unidade móvel reconecta com a rede fixa, uma grande quantidade de processamento retido acontecerá para efetivar a consistência das transações. As transações locais da unidade móvel podem necessitar de dados remotos para se completarem, transações remotas podem necessitar de dados locais, além das funções internas (por exemplo a transferência dos registros de log da unidade móvel para a rede fixa) que por serem consideradas críticas têm urgência de processamento

Segundo PITOURA & BHARGAVA (1994) as unidades móveis estão mais propensas a falhas que as unidades da rede fixa. Desta forma, garantir a durabilidade do

processamento executado em ambientes móveis distribuídos é uma tarefa muito mais árdua que nos ambientes estacionários. O conceito de *checkpoints* é bastante utilizado nas propostas para recuperação de falhas em bancos de dados móveis. Na ocorrência de falhas, o sistema utiliza o último registro de *checkpoint* salvo como ponto de partida para a recuperação. Um *checkpoint* global consiste de *checkpoints* locais. Análogo ao *checkpoint* global existe o conceito de estado global, os estados globais são utilizados para a recuperação de falhas dos sistemas.

3.8 Aplicações de banco de dados em ambientes de computação móvel

Neste item apresentaremos alguns exemplos de aplicações de banco de dados em ambiente de computação móvel. Alguns casos já totalmente implementados na vida real, e a possibilidade de utilização da plataforma em aplicações do dia a dia.

Iniciaremos a descrição das aplicações a partir do sistema GENESIS implementado sobre a proposta do ATIS (*Advanced Traveler Information Systems*) conforme a publicação SHEKHAR & LIU (1994). Da publicação de BUKHRES & MORTON (1997) apresentaremos as seguintes aplicações: (i) aplicação de banco de dados em computação móvel no cenário da guerra da Bósnia, (ii) aplicação da *International Mission Board*, (iii) aplicação de banco de dados móveis da companhia *Stolt Parcel Tankers*. A descrição do sistema implementado pelo Departamento de Polícia de Daytona Beach na Flórida – USA – seguirá após as aplicações anteriores.

3.8.1 Genesis e ATIS (Advanced traveler Information Systems)

O sistema ***ATIS*** presta auxílio aos viajantes através de serviços de planejamento, percepção, análise e tomada de decisão para aumentar a segurança, conveniência e eficiência da viagem. O sistema obtém informações de diversas origens, tais como: relatórios de tráfego, eventos de tráfegos agendados, sensores, mapas etc. Através do sistema, os viajantes podem consultar bancos de dados através de um canal *wireless* para buscar informações sobre tráfego ou rotas disponíveis para sua movimentação. Informações sobre condições do tempo e estradas, desvios, zonas de construção ou

paradas de ônibus são disponibilizadas aos viajantes ou qualquer usuário que faça uso do sistema. O sistema pode oferecer todas estas informações através do uso de algumas funções reunidas sob o contexto de cinco serviços que iremos citar a seguir:

i) **Serviço de informação ao viajante:** este serviço inclui um banco de dados que armazena informações sobre serviços disponíveis na estrada, atrações turísticas, mapas, hotéis e restaurantes.

ii) **Serviço de informação pré - viagem:** provê informações que serão úteis para o planejamento da viagem: condições da estrada e do tempo.

iii) **Serviço de orientação de rota:** após a escolha, pelo usuário, de um ponto de partida e um ponto de chegada o sistema, através deste serviço, poderá sugerir a rota mais adequada para cumprir aquele roteiro.

iv) **Serviço aconselhador do motorista em trânsito:** disponibiliza informações alcançadas durante a viagem: trechos em obras, congestionamentos, acidentes, condições do tempo e desvios.

v) **Serviço de segurança pessoal e notificação de emergência:** quando iniciado pelo motorista, ou automaticamente em caso de acidentes, propicia a capacidade de acionar os veículos de socorro bem como informar a localização do usuário.

O ***GENESIS*** é um projeto construído sobre a plataforma do *ATIS* em Minnesota (EUA), através de uma parceria entre a Universidade de Minnesota o Departamento de Transportes de Minnesota e algumas empresas do setor privado. Inicialmente os dispositivos móveis envolvidos no projeto serão *pagers, notebooks* e *PDAs*. O sistema conta com estações de coleta de dados, como as *TMC (Traffic Management Center)* e as *MTC (Metropolitan Transit Commission)*, um banco de dados localizado na rede estacionária, um canal de comunicação *wireless* e diversas estações móveis *(PCDs – Personal Communication Devices)*. O *TMC* é o computador central que controla a comunicação e gerencia o tráfego das estradas de ligação entre as cidades da região metropolitana. O *MTC* é o responsável pela coleta de dados sobre a localização atual dos congestionamentos e atrasos nos agendamentos relativos ao trânsito do setor público.

O sistema dissemina informações não previstas conforme as suas ocorrências, a figura de um gatilho trabalha com esta função, na ocorrência de determinado fato acontece a disseminação das informações. Além da disseminação de dados o cliente pode, dependendo da sua capacidade de armazenamento, efetuar o *caching* de páginas do banco de dados, permitindo que atue mesmo sob a forma desconectada, processando as requisições diretamente nas estações móveis.

3.8.2 Computação móvel em ambulatório militar

O ambiente da aplicação em questão é um campo de batalha, desta forma algumas características específicas devem ser ressaltadas: (i) o contingente médico possui acesso bastante restrito para qualquer hospital; (ii) os médicos precisam minimizar a movimentação de soldados e do corpo clínico no campo de batalha; (iii) os bancos de dados das unidades móveis e das estações base estão sujeitos a serem danificados a qualquer instante; (iv) o equipamento das estações base precisa oferecer facilidade de transporte e manuseio. O ambiente do campo de batalha foi dividido em cinco áreas de atuação, esta divisão abrange deste o local onde se encontra o soldado ferido até a localização física do hospital da cidade, as unidades médicas estão espalhadas por toda esta faixa.

As unidades móveis são os veículos médicos monitorados *(M3V)* equipados com computadores e médicos de campo que carregam *Personal Digital Assistant (PDA)*. Os soldados posicionados na primeira área do campo de batalha carregam consigo um equipamento chamado *Personal Status Monitor (PSM),* que monitora os sinais vitais do soldado e fornece a localização geográfica através de um canal *wireless*.

Na hipótese de um soldado encontrar-se ferido, o *PSM* transmite um sinal repetitivo informando os sinais vitais do soldado bem como a sua localização. Tanto um médico de campo quanto um *M3V* pode receber estes sinais e, neste mesmo instante iniciar uma transação para consultar as informações sobre o soldado, residentes no próprio computador. Cada médico tem uma cópia parcial do banco de dados de soldados que reside na rede estacionária. Após prestar o primeiro atendimento a equipe médica

pode iniciar uma nova transação atualizando os dados do soldado para que a equipe no centro médico do campo de batalha possa preparar a recepção do soldado ferido.

Em um local considerado como estável para armazenamento, as transações criadas pelas unidades móveis podem acessar tanto os dados móveis quanto às informações da rede estacionária. Neste mesmo local é mantida uma tabela com a localização de todas as unidades móveis. A figura do *base station agent (BSA)* é proposta para o gerenciamento e monitoração das transações sob o efeito das possíveis desconexões, o *BSA* está localizado em cada estação base da área 3.

Quando a equipe médica encontra um soldado ferido, utiliza a estação móvel para verificar se os dados daquele soldado estão armazenados no banco de dados local. Em caso de sucesso utiliza a cópia local. Se as informações sobre o soldado não estiverem armazenadas no banco de dados da unidade móvel, será necessário que se submeta uma transação para a rede solicitando a informação necessária. Se a estação móvel estiver desconectada da rede, a transação será efetivada na cópia local dos dados e posteriormente utilizada no processo de sincronização dos dados, que acontecerá na reconexão com a rede.

3.8.3 Aplicação da *International Mission Board (IMB).*

A *International Mission Board* (www.imb.org) baseada em Richmond na Virginia, presta suporte a vários eventos de relações públicas para a promoção de missões religiosas ao redor do mundo. A entidade utiliza uma aplicação multimídia para acessar os seus bancos de dados móveis. Em intervalos, quando os membros da equipe retornam para seus escritórios, sincronizam seus bancos de dados móveis com um banco de dados *master desktop* que por sua vez sofre atualizações regulares de um banco de dados central. Os agentes de campo viajam regularmente para mais de 131 países, carregando bancos de dados móveis nos quais mantêm informações sobre os donativos financeiros disponíveis bem como informações sobre os missionários que se encontram trabalhando ao redor do mundo. Os detalhes pessoais dos missionários e requisições para donativos serão atualizados nos bancos de dados móveis quando os agentes visitarem os missionários. A utilização de bancos de dados móveis para esta solução

permite que o agente trabalhe com as informações localmente, garantindo desta forma a continuidade do trabalho mesmo durante os períodos de desconexão.

3.8.4 Aplicação da companhia *Stolt Parcel Tankers.*

A empresa *Stolt Parcel Tankers* (www.stolttransportation.com) transporta diversos tipos de líquidos em navios tanque e cargueiros através do mundo inteiro, é a líder mundial no transporte marítimo de cargas químicas. A empresa utiliza uma aplicação de banco de dados móveis chamada *Stolt Tanker Operator Workstation (STOW)* para o gerenciamento de cargas. O sistema é baseado na arquitetura *client/server,* busca reservas de carga de uma central AS/400 e habilita ao operador a atribuição da reserva para um específico tanque no navio, através da movimentação *dragging* e *dropping* de ícones representando cargas para o interior de ícones representando os tanques.

O *STOW* efetua milhares de verificações com o objetivo de garantir que a carga possa ser acomodada no tanque designado. Os planos do *STOW* são enviados via satélite para as embarcações. Uma versão do sistema encontra-se instalada a bordo de tal forma que os tripulantes possam manipular o arranjo das cargas baseados na informação corrente. O sistema instalado a bordo do navio armazena as transações em *logs* e envia um *batch* uma vez por dia via satélite. A mesma técnica é utilizada para os *logs* que são recebidos dos navios. A necessidade de armazenamento e gerenciamento local de informações torna imperativo a utilização de bancos de dados para dar suporte às operações da empresa. A necessidade de interação com os dados mesmo que localmente, ou seja, na ocorrência de desconexões, aponta o uso de bancos de dados móveis como a solução mais adequada para o ambiente no qual encontra-se inserido o sistema.

3.8.5 Aplicação da polícia de *Daytona Beach – USA*

A população residente de *Daytona Beach* é de aproximadamente 65.000 pessoas, porém na alta temporada de turismo este número pode quadruplicar, nestas ocasiões o

serviço da força policial que conta com 130 membros tende a dificultar. Para suprir esta necessidade foi desenvolvido um sistema que trabalha sobre o conceito de bancos de dados móveis.

Antes da implementação do sistema a comunicação da força policial era feita utilizando três canais de radiofreqüência. O serviço do operador de teletipo é executar consultas para bancos de dados locais, estaduais e federais na tentativa de localizar registros de possíveis infratores detidos na cidade. O sistema desenvolvido converte a informação do banco de dados e permite o acesso pelo *link wireless* com o uso computadores portáteis. Desta maneira os oficiais podem efetuar o acesso às informações enquanto continuam escutando a comunicação entre os oficiais pelo canal de rádio. A manutenção de um banco de dados local com cada oficial permite que sejam emitidos relatórios localmente, sem a necessidade de manutenção da conexão com a base de dados central.

3.8.6 Aplicação Moses H. Cone Health System

O ambiente descrito a seguir encontra-se totalmente instalado e em utilização na rede de hospitais *Moses H. Cone (www.mosescone.com)* na Carolina do Norte – USA, este sistema de banco de dados móveis tem por objetivo a disseminação de informações sobre os pacientes entre a equipe médica e funcional do hospital.

A equipe médica pode sincronizar suas estações móveis com o banco de dados central que mantém as informações dos pacientes. Não existe mais a necessidade de interpretar receituários escritos a mão, anexar relatórios com resultados de exames para a ficha de cada paciente. Além da segurança no gerenciamento da informação a velocidade de resposta para qualquer questionamento foi sensivelmente incrementada através da adoção desta plataforma. A sincronização de dados foi adotada como a forma de atualização das informações. Foram implantados pontos de sincronização em cada andar dos hospitais. O setor de farmácia foi envolvido porque existe a necessidade de comparar relatórios laboratoriais com os resultados desejados. A plataforma móvel do setor de farmácia conta com um módulo que controla a tolerância a determinadas drogas para cada paciente, na hipótese de se detectar a prescrição de uma droga não tolerada

pelo paciente esta ocorrência será imediatamente repassada ao operador e até sugerida uma possível troca por outro tipo de formulação.

3.8.7 Caso da equipe de vendas

Neste exemplo os vendedores viajam para locais remotos cobrindo praticamente todo o país, conforme a necessidade e os objetivos da empresa. Nestes deslocamentos carregam consigo computadores portáteis, que armazenam um banco de dados com as informações dos produtos disponíveis. Este mesmo banco de dados comporta o armazenamento dos pedidos efetuados e das vendas concretizadas. No final de um dia de trabalho conectam seus computadores, através de algum enlace de rede disponível, com a rede estacionária do seu escritório central e dão início ao processo de sincronização dos dados de seus portáteis com as informações mantidas no banco de dados central. Através dos pedidos concretizados pelos clientes e informados para o banco de dados central por meio do processo de sincronização, é efetivada a atualização dos estoques, com base nas vendas efetuadas inicia-se o cálculo das comissões do vendedor. Os vendedores podem atravessar o país e ficar durante várias horas desconectados sem que sofram qualquer forma de prejuízo, é lógico que no caso da venda de bens tangíveis existirá sempre a limitação do estoque físico, neste caso faz-se necessário uma freqüência de sincronização maior do que a requisitada para a venda de bens intangíveis (seguros, serviços etc) onde não existe a restrição do estoque.

3.8.8 Caso do cientista em pesquisa de campo

O exemplo apresentado será baseado no trabalho de RENNHACKKAMP (1997). Neste exemplo, um botânico que viaja ao redor do mundo na pesquisa por novas plantas ou variedades delas, precisa ter suporte para que possa registrar e processar seus dados de pesquisa não importando o quanto esteja distante do seu laboratório base.

Visualizações e detalhes das amostras coletadas estão armazenados no seu banco de dados móvel, a manutenção destes dados é crucial para o seu trabalho, tendo em vista a possibilidade de retorno para localizações anteriores no intuito de investigar o progresso de algumas plantas conforme as mudanças climáticas. Os dados relativos à localização podem ser bastante extensos, é necessário na descrição da localização que se

trace uma rota com todos os detalhes que possam auxiliar na busca pelo local exato, facilitando desta forma que outros pesquisadores possam através da informação mantida encontrar o mesmo local para efetuar suas pesquisas, as coordenadas geográficas não conseguem informar a localização exata de uma determinada planta.

Além das informações textuais sobre o objeto que o pesquisador está estudando, acontece também o armazenamento de imagens e dados da localização geográfica no banco de dados do pesquisador. De tempos em tempos o pesquisador irá estabelecer conexão com seu laboratório central através de um *link wireless*, para que se efetue o processo de sincronização dos dados da estação móvel com o banco de dados mantido na rede estacionária. A necessidade de trabalhar com informações armazenadas localmente independente da manutenção ou não da conexão com a rede estacionária é crucial neste caso, o uso da tecnologia de banco de dados móveis atende a todas os requisitos de armazenamento local e remoto de dados. A instabilidade do enlace de comunicação entre as unidades móveis e a rede estacionária é característica marcante deste ambiente e considerada desta mesma forma para a aplicação de banco de dados móveis.

3.9 Considerações

Neste capítulo foram apresentadas as necessidades de adaptação dos conceitos de bancos de dados para o seu uso em um ambiente de computação móvel. Gerenciamento e modelos de transações, processamento de consultas e controle de concorrência foram discutidos sob os aspectos da mobilidade. As adaptações de algumas regras de integridade são necessárias neste ambiente. Esta necessidade é relacionada com a capacidade de recuperação de falhas destes bancos de dados. O próximo capítulo aborda especificamente as propostas de recuperação de falhas nestes ambientes, além de apresentar um comparativo entre estas propostas.

CAPÍTULO IV

Recuperação de Falhas em Bancos de Dados Móveis

O processo de recuperação de falhas em um ambiente de banco de dados tem por objetivo preservar a consistência das informações após a ocorrência de uma falha. Para que este processo seja executado com integridade, é necessário que no momento da recuperação dos efeitos das transações, as propriedades *ACID* sejam satisfeitas. No entanto, em um ambiente de banco de dados móvel é necessário uma adequação destas propriedades, visando atingir um grau de autonomia que o próprio ambiente exige. Esta adequação passa pela permissão de um determinado nível de inconsistência do banco de dados. Este capítulo apresentará algumas propostas de mecanismos de recuperação de falhas em bancos de dados móveis.

A seguir será feito um detalhamento do nível de adaptação de cada uma das propriedades *ACID*, configurados para dar suporte à mobilidade, esta apresentação será feita com base no estudo de BERTINO et. al (2001).

Atomicidade: para esta propriedade é proposta a possibilidade de divisão das transações em subtransações, cada uma delas sendo emitida de uma célula por onde a unidade móvel estiver passando naquele instante. Estas subtransações poderiam ser executadas concorrentemente e intervaladas com outras transações. O cancelamento de uma destas subtransações não necessariamente implicará no cancelamento de toda a transação, conseqüentemente a confirmação (*commit*) da transação principal pode acontecer mesmo que as subtransações não tenham sido confirmadas, o que configura um relaxamento do conceito tradicional de atomicidade.

Consistência: a adequação desta propriedade acontece através da divisão do banco de dados em agrupamentos (*clusters*), estes agrupamentos poderiam manter os itens de dados com uma semelhança semântica ou com localização próxima entre eles. Os dados pertencentes aos mesmo agrupamento (*cluster*) teriam um alto grau de consistência, enquanto que algum nível de inconsistência poderia ser tolerado entre os

agrupamentos. Desta maneira, poderão existir divergências entre as cópias de dados mantidas na unidade móvel (MH) e as cópias mantidas nas estações fixas (FH).

Isolamento: o relaxamento desta propriedade é, basicamente, uma conseqüência da remodelação das propriedades anteriores. É possível que os resultados intermediários de uma transação possam ser observados por outras transações, isto caracteriza o relaxamento da propriedade de isolamento. A divisão de transações em subtransações permite que após a confirmação de uma subtransação, os resultados desta (que são os resultados intermediários da transação principal) possam ser acessados por uma outra transação.

Durabilidade: esta propriedade é totalmente afetada pela possibilidade da mobilidade do cliente, uma transação poderia acontecer localmente na unidade móvel enquanto ela estivesse desconectada e, no instante anterior à reconexão a unidade móvel poderia falhar (falha de hardware, por exemplo) levando a perda total daquela transação. Na literatura encontramos algumas propostas de relaxamento desta propriedade, nestas propostas existem dois tipos de transações e dois níveis de confirmação (*commit*). As **transações de primeira classe** são aquelas executadas pelas estações fixas ou pelas unidades móveis enquanto estiverem conectadas. As **transações de segunda classe** são aquelas transações executadas pelas unidades móveis durante o período de desconexão. Uma estação móvel (MH) somente poderá confirmar uma transação **localmente** se esta transação não entrar em conflito com qualquer outra transação executada na mesma estação durante o período de desconexão. Na reconexão, as transações são confirmadas **globalmente**, a não ser que entrem em conflito com outras transações já executadas e confirmadas por outras unidades móveis. Os algoritmos apresentados neste capítulo fazem uso das adaptações das propriedades *ACID* apresentadas anteriormente, este relaxamento das propriedades faz-se necessário para que as propostas possam ser implementadas.

A recuperação do estado de um banco de dados após a ocorrência de uma falha é possível tendo em vista a permanência das informações em algum local estável dentro do ambiente da rede. Neste local estável serão encontradas as informações necessárias para a reconstrução do estado do banco de dados, estas informações são mantidas sob a forma de registros de *log*. A utilização de *checkpoints* no processo de recuperação de falhas de bancos de dados possibilita limitar a pesquisa dos registros do arquivo de log.

Através da emissão de *checkpoints* elimina-se a necessidade da releitura total do arquivo de *log*, podendo iniciar a leitura a partir do último *checkpoint* emitido e registrado pelo sistema. Esta característica contribui para a otimização do processo de recuperação do banco de dados pois, segundo OZSU & VALDURIEZ (1999), o *checkpoint* pode ser considerado como um limitador indicando que o banco de dados até aquele ponto está atualizado e consistente. Desta forma, o processo de refazer as transações terá início a partir daquele ponto limite, enquanto que o processo de desfazer as transações irá retornar somente até aquele mesmo ponto.

Ainda segundo OZSU & VALDURIEZ (1999), o processo de construção do *checkpoint* é chamado de *checkpointing* e é construído através de três passos:

1 Escrever um registro de *begin_checkpoint* no *log*.

2 Coletar os dados de *checkpoint* para dentro de um local de armazenamento estável.

3 Escrever um registro de *end_checkpoint* no *log*.

Os passos 1 e 3 garantem a atomicidade do processo de *checkpointing,* se uma falha do sistema acontecer durante este processo, a rotina de recuperação não irá encontrar um registro do tipo *end_checkpoint* e considerará que aquele processo de *checkpointing* não está completo.

A capacidade de recuperação do estado consistente de um banco de dados após a ocorrência de falhas é uma das principais necessidades de um sistema de banco de dados, seja ele estacionário ou móvel. Segundo OSZU & VALDURIEZ (1999), os processos de recuperação devem trabalhar com alguns tipos principais de falhas:

Falha de transações: entradas de dados incorretas de usuários, problemas de controle de concorrência e bloqueios de recursos são as principais causas de falhas englobadas neste tipo. Normalmente uma operação de *abort* será utilizada nestas ocorrências.

Falha de *sites*: este tipo de falha pode ser originada por problemas no hardware ou no software do *site*. No caso da falha acontecer no hardware, a parte do sistema que estiver armazenada em memória será perdida. Em sistemas distribuídos poderemos trabalhar com a situação em que uma falha de *site* acarreta na divisão da rede. Parte do banco de dados que estiver armazenado nas dependências do *site* com problema será

perdida, enquanto que o restante das informações permanecerá intacto juntamente com a porção da rede que estiver intacta.

Falha de comunicação: as falhas de comunicação são específicas para ambientes distribuídos ou móveis. A perda ou a falha na entrega de mensagens do sistema são as ocorrências englobadas nesta classificação. Estas ocorrências podem também contribuir para a partição da rede, a manutenção da consistência do banco de dados neste tipo de ocorrência é um desafio para os algoritmos de recuperação propostos.

Os três tipos de falhas citados na classificação anterior podem acontecer tanto em ambientes de banco de dados distribuídos quanto em ambientes móveis. Tratando-se de bancos de dados móveis algumas características do ambiente devem ser levadas em conta na construção de algoritmos que suportem a recuperação de falhas, segundo MATEUS & LOUREIRO (1998) as características seriam as seguintes:

- durante uma operação de processamento a unidade móvel pode movimentar-se entre as células de cobertura;
- a unidade móvel pode possuir memória estável utilizada para armazenar estados consistentes do banco de dados;
- a comunicação entre as unidades poderá ser concretizada através da utilização de um enlace sem fio (*wireless*).

Os conceitos apresentados no decorrer deste capítulo terão como base o ambiente descrito na figura 11. O ambiente será composto de uma rede estacionária onde estarão localizadas as estações fixas (FH – *fixed hosts*), algumas destas estações estarão equipadas com *interface wireless* para permitir a comunicação com as unidades móveis (MH – *mobile hosts*). Estas estações (BS – *base stations*) terão uma área de cobertura geográfica, onde o canal *wireless* estará disponível para a comunicação das unidades móveis (MH).

Em um ambiente móvel as estações clientes podem ficar temporariamente desconectadas do restante da rede, esta desconexão pode derivar de diversas causas: a mudança de área de cobertura geográfica, o esgotamento da capacidade da bateria, a falha ou mesmo a perda da unidade móvel são algumas destas causas.

Os algoritmos de recuperação de falhas devem levar em conta que, após um período de desconexão a unidade móvel poderá solicitar a sua reconexão com a rede fixa. No processo de reconexão da unidade móvel algumas consistências de informações

serão necessárias que aconteçam entre a unidade móvel e a estação base. Este processo de consistência visa preservar e garantir um estado de integridade de todo o banco de dados. Durante o processo de desconexão a unidade móvel poderá trabalhar com itens de dados mantidos em seu *cache* local, garantindo desta maneira o processamento de forma desconectada. Na etapa da reconexão, as informações que foram atualizadas na unidade móvel e os itens de dados que foram atualizados na rede fixa, irão sofrer um processo de sincronização das informações no intuito de manter a integridade do banco de dados.

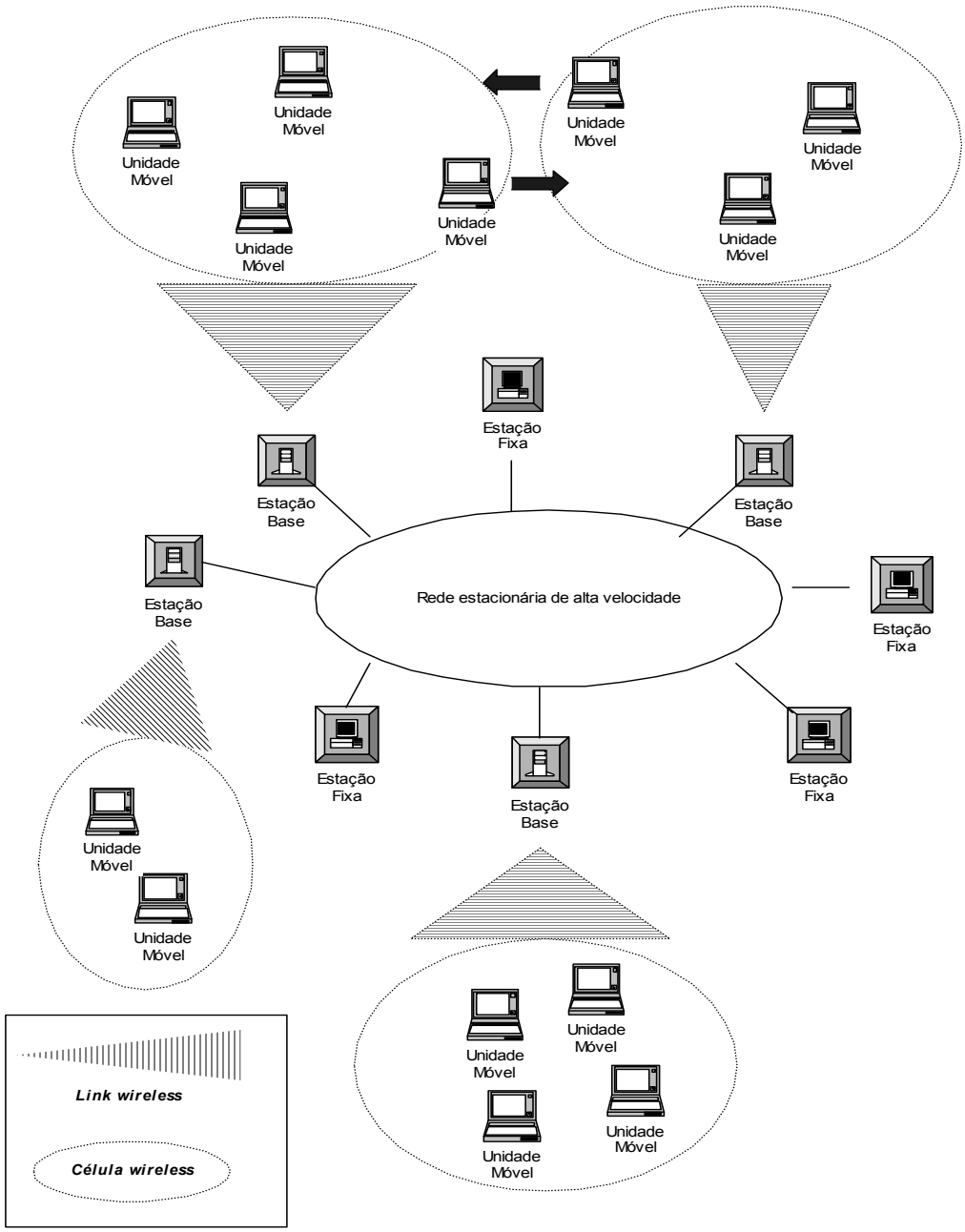

Figura 11 - Ambiente de computação móvel

4.1 Taxonomia dos algoritmos de recuperação de falhas.

A apresentação dos algoritmos neste capítulo seguirá a classificação apresentada a seguir, a figura 12 auxilia na visualização das classes. O primeiro nível de classificação indica a necessidade ou não de um mecanismo de coordenação das unidades móveis para a obtenção de um estado consistente do banco de dados. Este nível engloba as seguintes classificações: **algoritmos coordenados**, **não coordenados** e **compostos**. Os algoritmos classificados neste primeiro nível, sob o título de **compostos**, apresentam tanto características de coordenação quanto de não coordenação das suas unidades móveis.

O próximo nível da taxonomia classifica os algoritmos quanto ao local da ocorrência da falha: unidades de suporte, unidades móveis ou ambas. Neste nível de classificação os algoritmos classificados sob o item **unidades de suporte** proporcionam mecanismo para a recuperação de falhas que aconteçam nas unidades de suporte (MSS ou BS). Os algoritmos classificados sob o item **unidades móveis** oferecem o mecanismo de recuperação para as falhas localizadas exclusivamente nas unidades móveis.

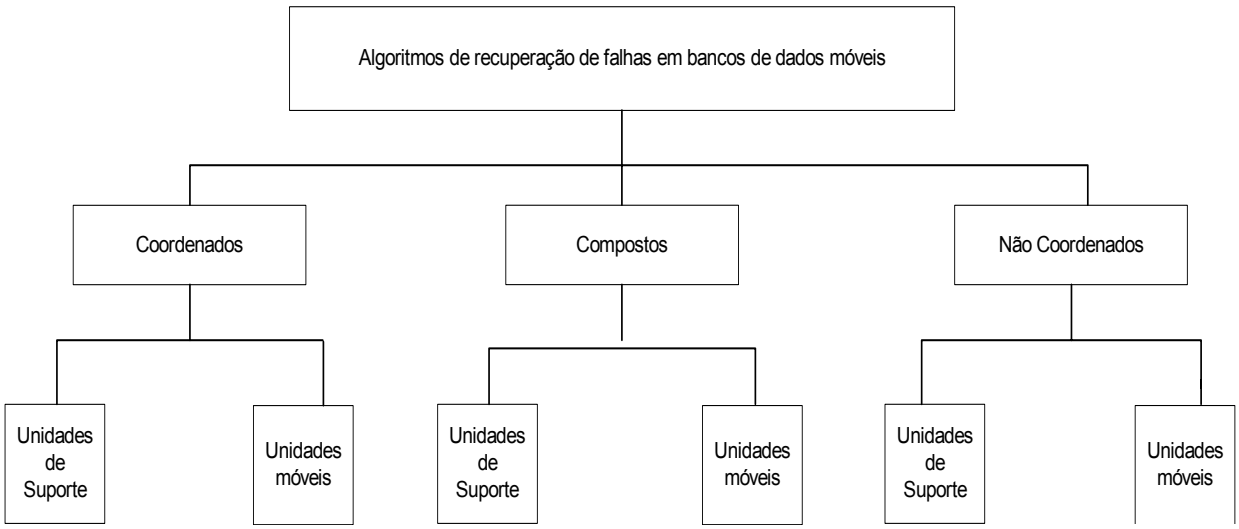

Figura 12 - Classificação dos algoritmos de recuperação de falhas

4.2 Protocolos de recuperação de falhas do tipo coordenados

Dentro deste item iremos apresentar os protocolos agrupados sob a classificação de protocolos coordenados. Os algoritmos classificados sob este item apresentam um mecanismo de sincronização das unidades móveis para a obtenção de um estado consistente do banco de dados.

4.2.1 Protocolo de *checkpoint* segundo NEVES & FUCHS (1997)

Segundo os autores, o alto grau de troca de mensagens durante a criação do *checkpoint* ou a coleta de informações durante o processo de recuperação são alguns dos principais problemas com os algoritmos de recuperação de falhas em ambientes móveis. Outro problema igualmente importante, segundo os autores, é o fato de que alguns protocolos não adaptam os seus ambientes para as características da conexão de rede corrente. A proposta deste protocolo é oferecer um mecanismo de recuperação de falhas adaptado às alterações de disponibilidade do enlace de comunicação das unidades móveis com a rede estacionária. Os autores sugerem a alternância de tipos de *checkpoints* na intenção de adaptação com o meio em que se encontra inserida a unidade móvel. O protocolo proposto estaria apto a armazenar estados de recuperação consistentes sem necessitar da troca de mensagens para este fim.

Os processos utilizam um *timer* local para determinar o momento da execução dos *checkpoints*. Nesta proposta os *checkpoints* podem ser salvos e armazenados tanto em locais de armazenamento estável quanto nas próprias estações. Os *checkpoints* armazenados localmente não consomem recursos de rede e podem ser construídos em muito pouco tempo, porém no caso de uma falha da unidade móvel podem ser permanentemente perdidos. Durante a execução da aplicação, o protocolo mantém um estado global em um local de armazenamento estável e um outro estado global fragmentado entre as unidades móveis e o local de armazenamento estável. O primeiro estado global é utilizado para a recuperação de falhas permanentes[16] enquanto que o segundo estado global é utilizado na recuperação das falhas passageiras[17].

[16] Falhas permanentes: este tipo de falha danifica permanentemente a unidade móvel
[17] Falhas passageiras: são as falhas cuja ocorrência não danifica permanentemente a unidade móvel, como por exemplo, a falta de energia ou um problema no sistema operacional.

Quando se desloca para outro endereço de rede, a unidade móvel confia os serviços de envio e recebimento dos seus pacotes de rede para o *foreign agent*, este agente possui uma interface *wireless* que permite o repasse dos pacotes relativos a unidade móvel. O processo de desconexão ocorre quando a unidade móvel está fora do alcance de todas as células de cobertura, o *home agent* representa a unidade móvel quando ela está fora da sua rede padrão (*home network*). O *home agent* intercepta os pacotes direcionados para a unidade móvel e os repassa para o *foreign agent* corrente. O *home node* é informado pela unidade móvel sobre a mudança dos *foreign agents*.

Segundo os autores, um protocolo de *checkpoint* coordenado salva os estados globais de uma aplicação que é executado por um ou mais processos. Estes processos executam em estações fixas ou móveis e utilizam mensagens para efetuar as trocas de dados. Desta forma, um estado global irá incluir o estado de cada processo ao longo da aplicação e algumas mensagens. A recuperação da falha é executada revertendo o processo para o último estado armazenado, então o processo re-executa o programa de aplicação e faz a releitura das mensagens "*logadas*". O protocolo de *checkpoint* adaptativo utiliza intervalos de tempo para coordenar a criação de estados globais, os processos salvam seus estados periodicamente sempre que um intervalo de tempo local para o *checkpoint* expira.

O protocolo utiliza dois tipos distintos de processos de *checkpoint*: *soft checkpoints* e *hard checkpoints*. Os processos do tipo *soft checkpoints* utilizam-se de *checkpoints* salvos localmente na unidade móvel para resolver as falhas *soft*. Os *checkpoints* do tipo *hard* utilizam *checkpoints* armazenados em plataformas consideráveis estáveis para a recuperação de falhas do tipo *hard.* Estes *checkpoints* necessitam ser enviados através de um enlace de comunicação, para que, através da rede estacionária se concretize o armazenamento em local estável, são, portanto, mais caros na construção e mais confiáveis no armazenamento. Este protocolo possibilita a adaptação com o ambiente de rede, intercalando a construção de *soft* e *hard checkpoints* conforme a disponibilidade de largura de banda. O número de *soft checkpoints* armazenados por *hard checkpoints* é chamado de *maxsoft*. Um estado global pode incluir tanto *soft* quanto *hard checkpoints*. Para assegurar que a recuperação seja sempre possível, o protocolo precisa manter em cada momento um estado global contendo somente *hard checkpoints*. Este estado global é utilizado para a recuperação de falhas

do tipo *hard*. O sincronismo do protocolo adaptativo obtido por meio dos intervalos de tempo para a criação de *checkpoints* evita a troca de mensagens durante este processo. Os processos salvam seus estados sempre que o contador de tempo local expirar, de forma independente dos outros processos. Cada processo carrega junto com suas mensagens a informação do intervalo de tempo restante para a ocorrência do próximo *checkpoint*.

Quando um processo recebe uma mensagem, compara seu intervalo local com aquele que está recebendo, se o intervalo recebido é menor, o processo ajusta seu contador de tempo com o valor recebido. O protocolo mantém um contador de número de *checkpoint* (CN), o valor de CN é incrementado sempre que o processo cria um novo *checkpoint*, e envia este valor junto com cada mensagem. O processo cria um novo *checkpoint* antes da entrega da mensagem para a aplicação se o CN*m* for maior que o CN local. Todas as mensagens que podem estar em trânsito são registradas em *log* do emitente, estas mensagens são todas aquelas que não foram confirmadas pelos destinatários no momento do *checkpoint*. O processo emitente também registra em *log* os CN emitentes e destinatários, estes contadores são utilizados para a detecção de perda ou duplicidade de mensagens devido às retransmissões. Durante o período de desconexão, a unidade móvel encontra-se fora da área de cobertura das células, desta forma a unidade não poderá acessar as informações mantidas em local estável (rede estacionária). A criação de um novo estado global antes da desconexão evitará que o trabalho efetuado na unidade móvel seja revertido. Sem a existência deste estado global a aplicação irá retroceder para o último estado global que foi armazenado, sem avisar a unidade móvel. No momento da reconexão, os processos da unidade móvel serão avisados sobre a falha e irão igualmente reverter, desfazendo todo o trabalho executado enquanto estivera desconectada.

Os processos de recepção de mensagem, criação e transmissão de registro de *checkpoint* são explanados através do algoritmo a seguir:

P_{m} = identificador do emitente
CN_{m} = número do *checkpoint* atual do emitente.
$tempoCkp_m$ = intervalo de tempo para o próximo *checkpoint* enviado pelo emitente
$tempoCkp_l$ = intervalo de tempo para o próximo *checkpoint* do processo local
msg_{m} = conteúdo da mensagem

Recebimento de Mensagens

***Recebemensagem*(**P_m, CN_m, $tempoCkp_m$, msg_m**):**
Se (($CN = CN_m$) **e** ($tempoCkp_l > tempoCkp_m$))
Ajusta contador de tempo para $tempoCkp_m$;
Senão Se ($CN < CN_m$)
CriaCkp( );
Ajusta contador de tempo para $tempoCkp_m$;
Fim Se
Fim Se

Criação e envio de *checkpoints*

CriaCkp( ):
$CN := CN + 1;$
Ajusta contador de tempo para $T;$
Se ($CN = maxsoft$)
Envia registro de *checkpoint* para a área de armazenamento estável;
Senão
Armazena o registro de *checkpoint* localmente;
Fim Se

4.2.2 Protocolo de *checkpoint* segundo PRAKASH & SINGHAL (1995)

O problema a ser resolvido através da implementação deste protocolo é, segundo os autores, oferecer um mecanismo de recuperação de falhas que minimize o custo computacional envolvido nesta recuperação. Para que este objetivo seja atingido, os autores sugerem a manutenção de estruturas de dependências, estas estruturas disponibilizam o relacionamento existente entre os nós que devem ser envolvidos em um processo de recuperação de falha. A limitação das unidades envolvidas no processo de recuperação é um dos maiores responsáveis pela diminuição no custo deste processo.

Os autores consideram que durante a operação normal, nenhuma mensagem é perdida ou modificada em trânsito, o sistema não possui nenhuma memória compartilhada ou um cronômetro global, desta forma, toda comunicação ou

sincronização acontece através de mensagens. As mensagens geradas pelas tarefas de processamento da aplicação serão referidas como **mensagens de computação**, enquanto que as mensagens geradas pelos nós (MH ou MSS) para a construção de *checkpoints,* manusear as falhas e para os processos de recuperação serão referidas como **mensagens de sistema**.

Uma das características do algoritmo é a limitação de envolvimento das unidades no processo de *checkpoint*, reduzindo o número de unidades envolvidas no procedimento. Este resultado pode ser atingido pela implementação de uma cadeia de relacionamento entre os nós (unidades). O relacionamento de causa entre as unidades é construído através da utilização de mensagens conforme será descrito a seguir.

O nó P_i mantém um vetor boleano R_i, de n componentes. No nó P_i o vetor é iniciado através das seguintes definições:

- $R_i[i] = 1;$
- $R_i[j] = 0 \text{ se } j \neq i;$

Quando o nó P_i envia uma mensagem para P_j , ele acrescenta R_i na mensagem. Este procedimento informa o nó P_j sobre os nós que lhe afetam de alguma maneira. No processamento da mensagem, P_j extrai o vetor $m.R$ da mensagem e utiliza-o para atualizar o vetor R_j de acordo com a fórmula: $R_i[k] \leftarrow R_i[k] \vee m.R[k]$. Este procedimento atualiza a informação de dependência das unidades. Se o emitente de uma mensagem for dependente de um nó P_k antes de enviar uma mensagem, o receptor da mensagem também será dependente do mesmo nó P_k. A figura 13 ilustra o dinamismo da manutenção de um vetor de dependência através do envio de mensagens. O nó P_4 é dependente de P_2 após ter recebido a mensagem m_3. Tendo em vista que P_2 era dependente de P_1 antes do envio da mensagem m_3 P_4 será dependente de P_1 quando receber a mensagem m_3.

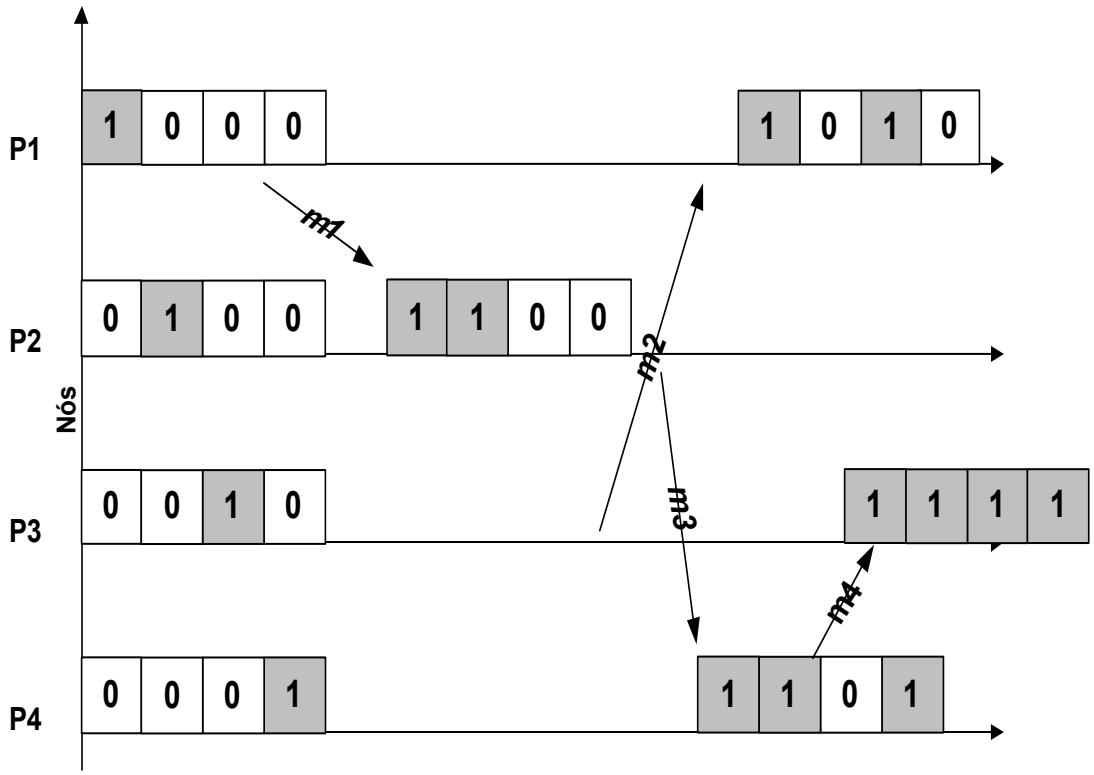

Figura 13 - Propagação da informação de dependência

Fonte: PRAKASH & SINGHAL (1995)

A utilização da informação de dependência permite a redução do esforço para a construção de um *checkpoint* global. No momento da construção do *checkpoint,* somente os nós (unidades) que tenham relação de dependência com a entidade que iniciou o processo serão afetadas pelo *checkpoint*. Após o término da coleta coordenada de *checkpoints,* os nós que não participaram da coleta podem tirar seus próprios *checkpoints* de uma forma *quasi-asynchronous.* Além do vetor boleano R_i, cada nó mantém um conjunto de estruturas de dados que serão apresentados a seguir:

- ***interval_number:*** um valor inteiro mantido em cada nó, incrementado a cada vez que o nó constrói seu *checkpoint* local.
- ***interval_vector:*** um vetor de *n* inteiros em cada nó, onde *interval_vector[j]* indica o *interval_number* da próxima mensagem esperada do nó P_j. Para o nó P_i , *interval_vector[i]* é igual ao seu *interval_number.*

- ***trigger:*** uma tupla (*pid, inum*) mantida para cada nó. O *pid* indica o *checkpoint* de início que forçou este nó a tomar seu último *checkpoint*. *Inum* indica o *interval_number* no nó *pid* quando ele tomou o seu próprio *checkpoint* local no início da coleta do *checkpoint.* A tupla *trigger* é agregada para cada mensagem de sistema (mensagens para a construção do *checkpoint*) e para a primeira mensagem de computação (mensagens de processamento do aplicativo) que um nó envia para todos os outros nós após construir o seu *checkpoint* local.

- ***send_infect:*** um vetor boleano de tamanho *n* mantido para cada nó em um local de armazenamento estável. A cada vez em que um *checkpoint* é coletado em um determinado nó, o valor do vetor *send_infect* é atualizado para zero. Quando um nó P_i envia uma mensagem de computação para um nó P_j , o valor do vetor *send_infect[j]* é atualizado para 1. Desta forma este vetor indica os nós para os quais mensagens de computação foram enviadas desde o último *checkpoint.*

- ***propagate:*** um vetor boleano de tamanho *n* mantido para cada nó em um local de armazenamento estável. É utilizado para manter um rastreamento dos nós para os quais requisições (*REQUESTs*) de *checkpoints* foram enviadas pelo nó, o vetor é iniciado com valor zero.

- ***weight:*** um valor real não negativo com máximo de 1, utilizado para detectar o término da coleta de *checkpoints*.

Os *interval_numbers* e os *interval_vectors* são iniciados para 1 e para um conjunto de 1s, respectivamente, em todos os nós. A tupla *trigger* no nó P_i é iniciada para (*i, 1*). O valor *weight* do nó é iniciado para 0. Quando o nó P_i envia uma mensagem, ele agrega seu valor *interval_number* e o vetor de dependência R_i, para a mensagem.

A ocorrência de falhas de nós (unidades) durante um processo de coleta de *checkpoints* deve ser considerada e passível de resolução pelo algoritmo. Supondo que o nó da falha (P_i) não seja o nó iniciante do processo de *checkpoint*, só existirão duas possibilidades: P_i pode falhar antes de receber a mensagem de *request* para a coleta do *checkpoint,* ou pode falhar após uma construção de um *checkpoint* local. Quando um nó

vizinho tenta enviar uma mensagem de *request* para o nó P_i, toma conhecimento da falha do nó e envia uma mensagem de *abort* para o iniciante do processo de coleta. Ao receber a mensagem de *abort* o iniciante dissemina uma mensagem de *discard.* Todos os nós que possuírem uma tentativa de *checkpoint* para este iniciante irão descartar este *checkpoint* na chegada da mensagem de *discard.* Quando um nó que havia apresentado falha reaparece no processo de comunicação, poderá se encontrar apenas em um de dois possíveis casos: possui uma tentativa de *checkpoint* local construída antes da falha, ou pode ter falhado antes do recebimento da mensagem de *request.* No caso de possuir uma tentativa de *checkpoint* local, irá verificar com o iniciante correspondente sobre o estado daquele *checkpoint.* Caso o iniciante tenha confirmado (*commit*) aquele *checkpoint,* o nó irá converter a tentativa de *checkpoint* para um permanente. Na hipótese do iniciante ter descartado aquele *checkpoint*, o mesmo procedimento será tomado pelo nó. Se a falha ocorreu em um nó iniciante de alguma coleta de *checkpoint*, e aconteceu antes do nó enviar mensagens de *commit* ou *discard,* ao retornar da falha o nó irá disseminar mensagens de *discard.* Se a falha ocorreu após o envio de mensagens de *commit* ou *discard*, nada mais será necessário a ser feito.

A proposta do presente algoritmo é envolver no processo de recuperação apenas aqueles nós que mantenham algum tipo de dependência, fazendo uso das estruturas de dados descritas anteriormente. Supondo que um nó P_i falhe e reverta o seu processamento (*rollback*), se nenhuma dependência de P_i para P_j tiver sido criada desde o último *checkpoint* de P_i, não existirá a necessidade de reverter o processamento de P_j. Somente aqueles nós que possuem alguma dependência com o nó da falha a partir do último *checkpoint,* terão a necessidade de reverter (*rollback*) para manter a consistência global do banco de dados.

O vetor *send_infect* permite que a unidade (nó) mantenha um registro dos nós para os quais tenha enviado mensagens de computação. O envio destas mensagens gerou uma dependência entre os nós. É necessário que tais nós revertam seu processamento da mesma forma que o nó da falha. Suponha que o nó P_i tenha sofrido uma falha, ele terá seu estado revertido (*rollback*) até o seu último *checkpoint,* mensagens de *rollback* serão enviadas para todos os nós que estão registrados em seu vetor *send_infect*. Junto com as mensagens, serão enviadas cópias do vetor *send_infect,* quando a unidade recebe uma mensagem de *rollback* irá executar as seguintes ações:

- P_j reverte (*rollback*) para o seu último *checkpoint*
- P_j envia uma mensagem de *rollback* para todas as unidades que constam do seu vetor *send_infect* mas não constavam do vetor do nó P_i . O vetor obtido pela operação de *OR* entre o vetor *send_infect* de P_j e o vetor recebido será enviado com as mensagens de *rollback.*

O nó que iniciou o processo de *rollback* tem um valor inicial para *weight* igual a um. Uma parte deste valor de *weight* é enviada pelo nó iniciante com cada mensagem de *request.* A cada vez em que um nó propaga uma requisição de *rollback,* também envia uma porção do seu *weight* com a mensagem. Ele também retorna seu valor residual de *weight* para o iniciante após o processo de *rollback* ter acabado. A fase de *rollback* terá término no momento em que o valor *weight* do iniciante se tornar igual a um. Após esta fase, o sistema foi restaurado para um estado consistente.

Falha em P_i

P_i reverte para o último estado consistente;
weight = weight – weight / 2;
Se *send_infect[j] = 1*
    P_i *request rollback* para P_j
    P_i envia *send_infect* e *weight* para P_j
Fim Se;

Recebimento *request rollback* e *send_infect* em P_j

P_j reverte para o último estado consistente;
Se *send_infect[k]* de (P_j) = 0 e *send_infect[k]* de (P_i) = 1
    P_j *request rollback* P_k
    P_j envia *send_infect[k]* de (P_j) $\vee$ *send_infect[k]* de (P_i) para P_K
Fim Se;
Retorna valor de *weight* para P_i;

Após executar uma reversão (*rollback*) e no início do processo de recuperação (*recovery*), o nó dissemina uma mensagem *recovering(i)* contendo o vetor

received[1..n]. Quando um nó P_j recebe uma mensagem *recovering(i),* retransmite cópias das mensagens significativas para P_i (mantidas em armazenamento estável), cujos valores de *sent[i]* são maiores que o valor de *received[i]* na mensagem disseminada. Após a disseminação da mensagem *recovering(i),* as mensagens que estão chegando de outros nós, são recebidas e processadas segundo os seguintes critérios:

1. se o valor de *sent[i]* na mensagem é igual ao valor de *received[j] + 1* em P_i e o ponteiro é não nulo e direciona para P_i na fila *QUEUE,* então a mensagem pode ser processada imediatamente. O ponteiro será então, movido para o próximo valor de *QUEUE*.
2. se o valor de *sent[i]* na mensagem for menor ou igual ao valor do *received[i],* então esta é uma mensagem duplicada e será ignorada.

4.2.3 Protocolo unilateral (*UCM*) segundo BOBINEAU et al. (2000)

A proposta do protocolo é apresentar um mecanismo de recuperação de falhas que faz uso de um mecanismo de confirmação de transação adaptado ao ambiente de computação móvel. O protocolo sugerido pelos autores é uma flexibilização do 2PC, pela eliminação da fase de voto do protocolo, adaptando este processo de confirmações ou abortos de transações para o ambiente da computação móvel. Os autores argumentam que o protocolo 2PC não atende as necessidades de funcionamento de bancos de dados em ambientes de computação móvel. Segundo o estudo, o protocolo 2PC apresenta algumas deficiências quando utilizado em um ambiente de computação móvel:

- **Processamento *off-line:*** uma transação executada em um dispositivo desconectado, não pode ser confirmada (*commit*) localmente. Desta maneira, continua tentando a confirmação até que a unidade se reconecte e o protocolo 2PC seja executado nos servidores fixos.
- **Movimentação dos participantes:** o protocolo 2PC incorre em um aumento significativo de mensagens, este problema poderá ser acentuado se o coordenador estiver localizado em uma unidade móvel ou se os participantes puderem se movimentar. Além da possibilidade de

movimentação, é possível que aconteçam desconexões durante a aplicação do protocolo 2PC.

Para resolver os problemas do protocolo 2PC no ambiente de computação móvel, os autores apresentam o protocolo *Unilateral Commit for Mobile* (*UCM*) que, segundo a proposta, apresenta as seguintes propriedades:

- Uma transação executando em uma unidade *off-line* poderá ser confirmada (*commit*) assim que seu *log* for transferido na rede fixa, sem a necessidade de aguardar pela confirmação dos servidores.
- O protocolo não necessita da presença de todos os servidores no momento da confirmação (*commit*).
- O protocolo é composto de um único ciclo de mensagens, permitindo a redução dos custos de comunicação.

A eliminação da fase de voto do protocolo 2PC é a idéia principal do protocolo *UCM*, desta forma, o protocolo de confirmação é reduzido para única fase, que é a disseminação da decisão do coordenador para todos os participantes.

Segundo os autores, para um melhor entendimento do protocolo, as fases de término e execução de uma transação devem ser consideradas juntas. Na execução e término de uma transação é possível a identificação de quatro componentes, os quais serão descritos a seguir:

- **Aplicação:** solicita a execução de uma seqüência de operações.
- **Participantes:** responsáveis pela execução das operações.
- **Coordenador:** controla o término do protocolo.
- **Pagents:** um por participante, representa os participantes no término do protocolo e exerce um papel decisivo no processo de recuperação.

O coordenador estará localizado na rede fixa, os outros componentes podem ser hospedados por uma unidade móvel. A figura 14 ilustra uma configuração tradicional , nesta configuração a aplicação (App) , o logagent (LA) e o coordenador (C) estão localizados em um *site* da rede estacionária. Os participantes (P) são as unidades móveis e os seus pagents (PA) estão localizados nas MSSs.

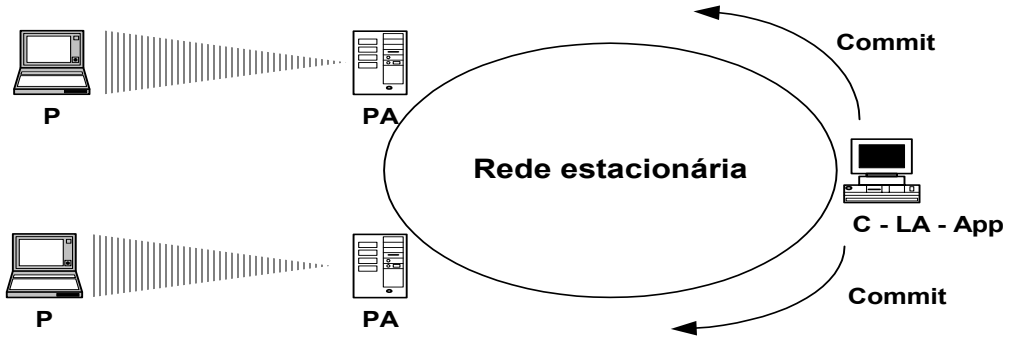

Figura 14 - Localização dos componentes no modelo *UCM*

Os autores descrevem o funcionamento do algoritmo através da simulação das ações de um único participante P_k, T_{ik} representa a porção local da transação T_i executada no participante P_k.

Segundo os autores, o passo 1 representa a seqüência de operações executadas em benefício de T_{ik}. O componente *logagent* registra no seu *log* todas as operações que são executadas. O registro destas operações é efetivado no formato *non-forced write*. No formato *non-forced write* as informações são armazenadas em um *buffer* da memória principal e não geram bloqueios de *I/O*. Após este processo, as informações são enviadas para o participante P_k e lá processadas localmente. Cada operação é reconhecida para a aplicação.

O passo 2 corresponde ao término da transação, do ponto de vista da aplicação. Após o recebimento pela aplicação, de todos os reconhecimentos, ela irá emitir uma requisição de *commit*. Neste ponto, as propriedades ACI estão localmente garantidas por todos os participantes de todas as transações locais parciais. Os participantes ainda não têm o conhecimento do término da transação, desta forma não podem garantir a propriedade de durabilidade. A propriedade de durabilidade será garantida pelo coordenador, executando a escrita do *log* de T_i produzido pelo *logagent* em armazenamento estável. Esta operação de escrita acontece no formato *force-write*.

Após a conclusão do passo 2, as propriedades ACID estarão completamente garantidas para todas as transações parciais. Então, o coordenador toma a decisão de confirmação da transação (*commit*) e escreve esta decisão (*force-write*) em armazenamento estável. Neste ponto, o processo já se encontra no passo 3. O

coordenador dissemina a decisão *commit* para todos os participantes e aguarda pelos seus reconhecimentos. Quando todos os reconhecimentos forem recebidos, o coordenador libera a transação. Se a transação T_i for abortada, o coordenador descarta todos os registros de *log* da transação e dissemina uma mensagem *abort* para todos os participantes.

Se um participante falhar durante o passo um do protocolo, o coordenador dissemina uma decisão de *abort*. Caso o coordenador não tenha recebido ack_k $(commit_i)$, todas as transações parciais podem ter sido confirmadas com exceção da transação T_{ik}. O participante P_k já assumiu a garantia de honrar as propriedades ACID da transação local T_{ik}. Desta forma, uma vez que P_k completou sua recuperação local, ou T_{ik} foi recuperado com retrocesso se P_k falhou antes da execução do $commit(T_{ik})$, ou T_{ik} está localmente confirmado (commit) se P_k falhou apenas antes do envio da mensagem de reconhecimento da operação de *commit* para o coordenador. Então, se T_{ik} já foi confirmada com sucesso por P_k, o coordenador não fará mais nada. Caso contrário, T_{ik} é re-executada graças ao *log* do coordenador que contém todas as operações $T_{ik.}$

A decisão da necessidade de re-execução da transação T_{ik} é de responsabilidade do componente $PAgent_k$. Quando $PAgent_k$ recebe a decisão do *commit* do coordenador, ele emite uma operação adicional "*write Record* <$commit_i$>" em benefício de T_{ik} antes de solicitar que P_k confirme (*commit*) T_{ik}. Esta operação será tratada por P_k da mesma maneira que todas as outras operações ao longo de T_{ik}. Para tomar o status de uma transação local parcial T_{ik} , após uma falha, o componente $PAgent_k$ verifica a existência do registro <*commit*> em P_k. Se o registro for encontrado, isto prova que T_{ik} foi confirmado (*commit*) com sucesso em P_k antes da falha. Caso contrário, T_{ik} foi recuperado com retrocesso e precisa ser re-executado.

Passo 1
Aplicação requisita operações de transação local (***T_{ik}***) em ***P_k***;
LogAgent registra as operações em *log;*
Solicitações enviadas e processadas em ***P_k***;
Envio de reconhecimento de ***P_k*** para a **Aplicação**;

Passo 2
Envio de requisição de ***commit*** para todos os participantes;
Escrita pelo **coordenador**, dos registros do ***logagent*** em local estável;

Passo 3

Se decisão **coordenador** = *commit*
  Então
    Registra operação *commit* em local estável
    Envia requisição *commit* para todos os participantes
  Senão
    Descarta os registros de *log* da transação
    Envia requisição de *abort* para todos os participantes
Fim Se

4.2.4 Recuperação de falhas no *PRO-MOTION*

O objetivo desta proposta é fornecer a possibilidade de gerenciamento local das transações dos bancos de dados, através da utilização de estruturas de informação que armazenam as instruções e recursos necessários para que este gerenciamento possa acontecer. Desta forma, o *PRO-MOTION* é um sistema de processamento de transações móveis, que oferece o suporte para o processamento de transações no modo desconectado.

Segundo os autores, para reduzir o custo de um processo de recuperação, os efeitos de uma determinada transação não podem estar disponíveis a outras transações até que a mesma seja confirmada e suas alterações sejam efetivadas na base de dados. Porém durante uma operação desconectada, as atualizações efetuadas na unidade móvel não podem ser confirmadas no servidor da rede estacionária até que ocorra a reconexão da unidade móvel. Conseqüentemente, as transações subseqüentes que utilizam os mesmos itens de dados, não poderão ser executadas até que o processo de reconexão e a confirmação da transação aconteçam. Uma alternativa para resolver este problema seria disponibilizar o acesso aos resultados da transação assim que a mesma iniciasse o processo de confirmação (*commit*) na unidade móvel.

O sistema *Pro-Motiom* é composto de um agente cliente móvel chamado de *compact agent*, um *front-end*[18] do servidor estacionário denominado *compact manager*, e um *array* intermediário de *mobility managers* responsável pelo gerenciamento do fluxo de dados e transações entre os componentes do sistema. O componente principal do sistema é o *compact,* englobados neste componente encontram-se as informações

necessárias para o seu funcionamento. Segundo os autores, o *compact* é um objeto que encapsula as seguintes informações:

- Dados de *cache*
- Métodos para o acesso aos dados de *cache*
- Informação sobre o estado atual do *compact*
- Regras de consistência para a garantia da consistência global do item de dados
- Compromissos do sistema, como por exemplo, o limite de tempo necessário a ser aguardado até que determinado processo libere um recurso solicitado.
- Métodos que oferecem uma *interface*[19] através da qual a unidade móvel pode gerenciar o *compact.*

As figuras 15 e 16 ilustram respectivamente, a arquitetura do sistema *Pro-Motiom* e os componentes de um objeto *compact*.

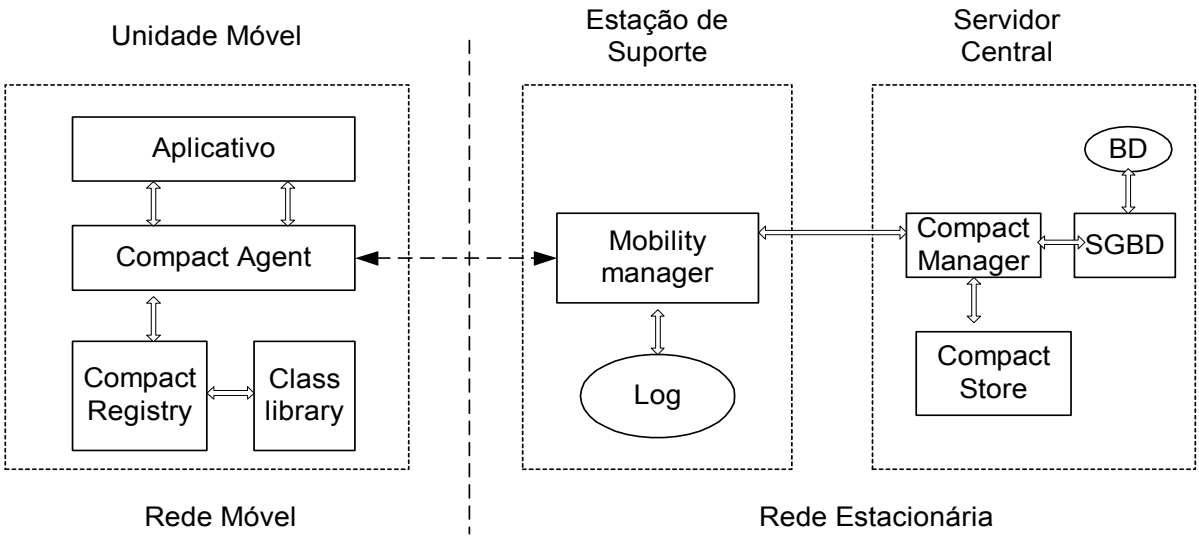

Figura 15 - Arquitetura do sistema *Pro-Motiom*

Fonte: WALBORN & CHRYSANTHIS (1998)

[18] *Front-end:* camada mais externa de acesso aos dados, provida para fornecer a comunicação com o usuário do sistema

[19] *Interface:* fronteira de comunicação entre níveis, sistemas ou usuários, prov6e a troca de informações entre duas camadas de um sistema.

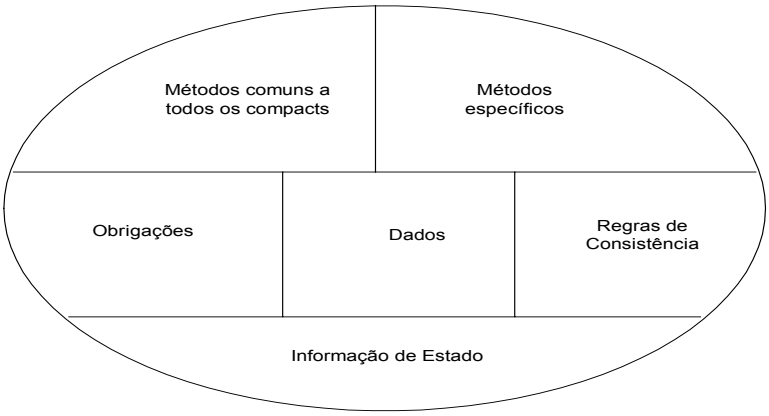

Figura 16 - Componentes do objeto *compact*
Fonte: WALBORN & CHRYSANTHIS (1998)

O gerenciamento dos *compacts* é executado em conjunto com o *compact manager* no lado servidor e o *compact agent* de cada unidade móvel. Quando uma unidade móvel emite uma requisição de itens de dados para o servidor de banco de dados, o servidor verifica se os dados estão disponíveis para o envio. Se a requisição puder ser satisfeita, o servidor constrói com a ajuda do *compact manager,* um *compact* e o registra no *compact store,* após este procedimento faz o envio do *compact* para a unidade móvel solicitante. No *compact* encontram-se os dados e métodos necessários para que a unidade possa executar a sua transação. O *Pro-motion* considera o sistema móvel como uma grande transação, esta transação é executada no servidor e é composta de diversas subtransações executadas nas unidades móveis. Quando uma unidade móvel efetua a reconexão com o servidor de banco de dados, identifica um grupo de *compacts* cujos estados refletem as atualizações das transações confirmadas localmente. Estas transações são separadas das transações não confirmadas e informadas para o *compact manager*, que cria uma transação para este grupo de alterações. Esta transação criada pelo *compact manager* será então confirmada no banco de dados deixando seus efeitos visíveis a todas as outras transações.

Os autores apresentam quatro atividades principais durante o processamento de transações no ambiente do *Pro-Motiom*: ***hoarding,*** **processamento conectado**, **processamento desconectado** e **sincronização.**

A atividade de ***hoarding*** ocorre quando a unidade móvel está conectada, para dar suporte ao trabalho desconectado o *compact manager* armazena *compacts*. Durante o **processamento conectado** a unidade móvel tem total conexão com a rede estacionária, desta forma o *compact manager* apenas processa as transações. No **processamento desconectado** a unidade não tem qualquer conexão com a rede estacionária, o *compact agent* processa as transações localmente. Finalmente no processo de **sincronização**, a unidade móvel retorna à conexão com a rede estacionária, neste momento o *compact agent* executa o processo de sincronização das alterações confirmadas durante o período desconectado, com o banco de dados da rede estacionária.

Segundo os autores, vários *compacts* podem estar envolvidos em uma única transação local, desta forma a confirmação local de uma transação é executada obedecendo ao protocolo 2PC. O sistema *Pro-Motiom* trabalha com um procedimento de contingência que será anexado a cada evento de ***COMMIT*** e registrado no *log* de eventos com o respectivo evento ***COMMIT.*** Este procedimento de contingência será executado na eventualidade de que a transação local não possa ser incorporada ao estado do servidor de banco de dados da rede estacionária. Quando um evento ***COMMIT*** é recebido de uma aplicação, o *compact agent* envia e registra em *log* mensagens de ***PREPARE*** e ***COMMIT***. No recebimento de um evento de ***COMMIT***, cada *compact* participante invalida o estado do objeto salvo anteriormente, valida o novo estado confirmado e conclui aceitando os eventos. No recebimento de um evento de ***ABORT***, o *compact* recarregará o estado consistente anterior e executará o processamento do próximo evento.

O processo de recuperação sugerido pelos autores atinge apenas as falhas que possam ter acontecido durante o período de desconexão das unidades móveis. O protocolo parte do princípio de que todas as transações que se encontravam pendentes no momento da falha serão abortadas. Ao retornar da falha, o *compact agent* verifica a existência, no *log*, de registros relativos a transações pendentes, os registros encontrados serão então removidos. Se o *compact agent* tenta emitir uma operação para um *compact* que não se encontra na memória principal, o *compact registry* cria uma instância do *compact* e requisita o método *recover( ).*

Caso o *compact* tente recarregar o estado a partir do local de armazenamento e descubra que o processo estava em fase de confirmação, precisará consultar o *log* para

determinar se o processo de confirmação foi efetivado. Caso o processo tenha sido confirmado, irá recarregar o estado correto. Os *compacts* que não estavam envolvidos no processo de confirmação, no momento da falha, irão recarregar o último estado consistente a partir do objeto de armazenamento e concluir o processamento.

O procedimento de sincronização pode ser considerado uma extensão deste protocolo, já que atua no sentido de recuperar o banco de dados para um estado consistente após o trabalho desconectado das unidades móveis. Este processo poderá envolver até quatro etapas que serão descritas a seguir:

- **Primeira etapa:** nesta etapa o *compact registry* irá identificar os identificadores de todos os *compacts* que foram modificados desde que a unidade efetuou a sua desconexão. Caso todos os *compacts* sejam considerados válidos, o processo de resincronização poderá ser completado.
- **Segunda etapa:** nesta fase o *compact registry* transmite a identificação dos *compacts* modificados e expirados e tenta renovar o *compact* e aumentar o tempo de vida do mesmo. Caso nenhuma outra entidade tenha bloqueado ou modificado os dados do *compact*, o *compact manager* poderá restabelecer o *compact* e validar o *compact client-side* como se nunca tivesse sido expirado.
- **Terceira etapa:** este passo será executado caso os *compacts* modificados não possam ser restabelecidos. Os valores originais dos *compacts* válidos ou restabelecidos podem ser obtidos a partir do *compact manager,* já os *compacts* que não puderam ser restabelecidos são considerados inválidos. O procedimento será a re-execução do *log* para todas as transações confirmadas. Se uma transação lê ou modifica um *compact* inválido, a transação será considerada abortada e todos os *compacts* modificados pela transação abortada serão marcados como indisponíveis. O mesmo procedimento acontecerá para as transações que efetuaram alguma operação de leitura com base nas transações indisponíveis. Ao final da re-execução do *log*, o conjunto restante dos *compacts* válidos representará um sub conjunto das transações confirmadas localmente que podem ser incorporadas ao banco de dados da rede estacionária. O procedimento de

contingência das transações que anteriormente foram marcadas como confirmadas, mas sofreram o processo de aborto, precisará ser executado.

- **Etapa final:** nesta fase, o *compact registry* verifica o conjunto de *compacts* buscando os *compacts* válidos e que foram modificados. Para cada *compact* será gerado um evento ***OP*** que será retornado para o *compact server-side* no intuito de atualizar os dados do servidor com os dados da unidade móvel. No final de todas as operações, a unidade móvel envia uma mensagem de ***COMMIT*** para o *compact manager*. No lado servidor, todas as operações de atualização do processo de sincronização são agrupadas como uma única transação que é confirmada no banco de dados. Para que este processo aconteça com correção, o *compact manager* tenta emitir bloqueios em todos os itens que estão sendo confirmados. Caso os bloqueios possam ser mantidos pelo servidor, os relativos *compacts* tanto do lado servidor como do lado cliente permanecem válidos e disponíveis. Caso não possa manter os bloqueios solicitados, os *compacts server-side* e os *client-side,* serão invalidados.

Ao final do processo de sincronização, os *compacts* serão liberados e a unidade móvel poderá retornar ao seu estado de *hoarding*.

4.3 Protocolos de recuperação de falhas do tipo não-coordenados

Dentro deste item iremos apresentar os protocolos agrupados sob a classificação de protocolos não-coordenados. Os algoritmos classificados sob este item não apresentam mecanismos de sincronização das unidades móveis para a obtenção de um estado consistente do banco de dados.

4.3.1 Protocolo de *checkpoint* por BADRINATH & ACHARYA. (1994)

Segundo os autores, um *checkpoint* global é construído a partir de um conjunto de *checkpoints* locais consistentes. Em um ambiente de computação móvel alguns problemas poderão ocorrer na coleta destes *checkpoins* locais. A movimentação da

unidade forçará que aconteça uma pesquisa sobre a localização atual da unidade, incrementando os custos com a troca de mensagens no ambiente de rede. Além da movimentação existe ainda a possibilidade da desconexão da unidade, durante este período a coleta dos *checkpoints* locais ficará prejudicada ou até mesmo impossibilitada. O protocolo apresentado a seguir procura tratar e resolver estes problemas através de um mecanismo de construção, armazenamento e roteamento de *checkpoints* locais a partir das unidades móveis em direção às estações de suporte.

Os autores indicam o número de MHs nas quais a aplicação é executada por N, e o número de MSSs para as quais qualquer MH pode se deslocar como M. Assumem ainda que o protocolo de roteamento das mensagens garante a ordem no recebimento das mesmas. Um *checkpoint* global *G_Ckpt* da aplicação rodando em N MHs consiste de um *checkpoint* local $local\_ckpt_i$ para cada MH h_i tal que:

- O $local\_ckpt_i$ inclua um estado local da MH h_i. O estado local de uma MH altera-se devido a recepção de uma mensagem, envio de uma mensagem ou um evento local que não envolva comunicação. Para uma MH h_i com um estado inicial s_i^0 , o estado após uma seqüência de eventos E_i , é representado como $< s_i^0 E_i >$. Se h_i sofrer o seu *checkpoint* neste estado local, então todos os eventos $e \in Ei$ estarão incluídos no *checkpoint* local e desta forma, no *checkpoint* global *G_Ckpt*.
- O $Local\_ckpt_i$ também inclui uma cópia da cada mensagem *m* enviada da unidade h_i para uma unidade h_j.. Caso o evento *send(m)* esteja incluído no *G_Ckpt* e o respectivo evento *recv(m)* não esteja.

Um *checkpoint* global *G_ckpt* será consistente se atender à seguinte propriedade: para qualquer mensagem *m,* se o evento *recv(m)* estiver incluído no *G_ckpt,* o evento *send(m)* também deverá estar incluído. Para a construção de um *checkpoint* global, é necessário que os *checkpoints* locais sejam armazenados em locais considerados de armazenamento estável. Armazenar os *checkpoints locais* das unidades móveis (MH) no disco das estações fixas (MSS) é a sugestão dos autores. O *checkpoint* sempre será armazenado na MSS que atualmente é considerada a MSS local para a unidade móvel (MH). Tendo em vista a característica de mobilidade das unidades, uma seqüência de *checkpoints* de uma mesma unidade (MH) poderá estar armazenada em MSS distintas,

um para cada célula visitada pela unidade. O algoritmo trabalha com um mecanismo assíncrono de construção de *checkpoints*, cada unidade móvel (MH) constrói seu próprio *checkpoint* e o armazena na MSS sob a qual está em atividade no momento (MSS local). Na ocorrência de desconexão, a MH constrói o seu *checkpoint* prévio e o envia para a sua MSS local.

Os autores apresentam no algoritmo em estudo o funcionamento da regra de duas fases *(two-phase rule),* segundo esta regra uma estação móvel (MH h_i) constrói seu *checkpoint* local da seguinte maneira:

1. Se ($phase_i$ == *send*) e a unidade recebe uma mensagem *M,* ela constrói seu *checkpoint* local e alterna para a fase *recv.* Antes de efetuar a entrega da mensagem altera o valor da fase, *phase:= recv*.
2. Se ($phase_i$ == *recv*), antes de enviar uma mensagem M para outra unidade MH, a unidade alterna para a fase *send, phase:= send.*

Uma MH armazena uma cópia de cada mensagem enviada em um local definido como *message log*. Sempre que uma MH constrói um *checkpoint* local, o *checkpoint* e o registro de mensagens (*message log*) é transmitido para a MSS local. Esta regra garante que nenhuma dependência será criada entre dois *checkpoints* locais para a mesma unidade móvel (MH).

Na execução do algoritmo nas unidades móveis uma série de ocorrências pode ser identificada, um detalhamento destas ocorrências e o respectivo tratamento dado pelo algoritmo serão apresentados no decorrer do texto.

No envio de uma mensagem M, se a $fase_i$ é RECV, então o valor de $fase_i$ será atualizado para SEND. Após a atualização do valor da $fase_i$, uma cópia de $CKPT_i[]$ e $LOC_i[]$ serão enviadas com a mensagem M, ou seja: *M*_CKPT[]:= $CKPT_i[]$ e *M*_LOC[] := $LOC_i[]$.

Se $fase_i$ = RECV
Então
$fase_i$:= RECV
*M*_CKPT[]:= $CKPT_i[]$

M_LOC[] := LOC_i[]

Fim Se

No evento de recebimento de uma mensagem M, emitida a partir de uma MH h_j , se o valor de $fase_i$ for igual a SEND então o algoritmo irá executar um processo de *checkpoint*. O valor da fase será atualizado para RECV após a conclusão do procedimento anterior. Ainda durante o recebimento da mensagem, será feita uma comparação entre os valores de M_CKPT[j] e $CKPT_i$[j]. Na hipótese do valor de M_CKPT[j] ser maior que o valor de $CKPT_i$[j], o valor de $CKPT_i$[j] será atualizado para M_CKPT[j] e o valor de LOC_i [j] para M_LOC[j].

Se $fase_i$ = SEND
Então
Checkpoint( )
$fase_i$:= RECV
Se M_CKPT[j] > $CKPT_i$[j]
Então
$CKPT_i$[j] := M_CKPT[j]
LOC_i [j] para M_LOC[j]
Fim Se
Fim Se

A mobilidade das estações permite que ocorra a troca de células de cobertura. Desta forma, a unidade pode ter a sua MSS local alterada por diversas vezes, conforme a sua movimentação dentro do ambiente. Para dar suporte a esta troca de MSS, o algoritmo executa um *checkpoint* na unidade móvel atual e mantém este *checkpoint* registrado na MSS atual. Após este registro de *checkpoint* a unidade móvel se deslocará para a nova célula (MSS_p). Como passo final desta etapa de troca de MSS, o valor de LOC_i[i] será atualizado para p (LOC_i[i] := p).

Mudança de célula de cobertura
Checkpoint(i);
Registro do *Checkpoint(i)* em MSS_i;
Mudança de MSS;
LOC_i[i] := p;

A possibilidade do trabalho desconectado é característica única do ambiente de computação móvel, antes de efetivar a desconexão da unidade em relação à rede estacionária, o algoritmo sugere que a unidade execute um procedimento de *checkpoint* local. Para registrar um *checkpoint* local, a unidade registra um estado local (*local_state*$_i$) da aplicação distribuída executada na unidade h_i, após este registro os valores de *local_state*$_i$, *log*$_i$, CKPT$_i$[] e LOC$_i$[] serão transferidos para a MSS local. O valor de CKPT$_i$[i] será então atualizado para CKPT$_i$[i] + 1.

Registro de *checkpoint* local

Registro de *local_state*$_i$ (h_i);
Transf_MSS_Local (*local_state*$_i$, *log*$_i$, CKPT$_i$[] e LOC$_i$[]);
CKPT$_i$[i] := CKPT$_i$[i] + 1;

Da mesma forma que nas unidades móveis (MH), uma parte específica do algoritmo é executada nas unidades fixas (MSS), o funcionamento desta porção do algoritmo será descrito na seqüência.

Para construir um *checkpoint* global, uma MSS precisa coletar um conjunto consistente de *checkpoints* locais. Cada *checkpoint* local é constituído de *local_state*$_i$, *log*$_i$, CKPT$_i$[] e LOC$_i$[]. A unidade fixa (MSS) iniciante envia uma mensagem de requisição, *request*(h_j, *CKPT*$_i$*[j]),* para a MSS$_p$, onde p é o valor LOC$_i$[j], solicitando o *checkpoint* local de h_j com um número de seqüência CKPT$_i$[j]. Se o *checkpoint* local da MH h_j com um número de seqüência CKPT$_i$[j] está disponível na MSS que recebeu a mensagem de *request(),* então a MSS responde para a MSS iniciante com o *checkpoint* local solicitado. Caso contrário, a unidade fixa (MSS) irá esperar que h_j emita seu próximo *checkpoint* com número de seqüência igual a CKPT$_i$[j]. A unidade móvel irá registrar e transferir o *checkpoint* requerido antes da sua próxima desconexão ou antes do recebimento ou envio das próximas mensagens. A MSS poderá forçar que a MH construa seu registro de *checkpoint.*

Na MSS que recebe a solicitação

Se número de seqüência *checkpoint* = CKPT$_i$[j]
Então

Envia *checkpoint* para MSS iniciante;

Senão

Emite *checkpoint* = $CKPT_i[j]$;

Envia *checkpoint* para MSS iniciante;

Fim Se

No processo de reconstrução de um *checkpoint* global consistente, o iniciante aguarda o recebimento de um *checkpoint* local de cada MH. Se uma MSS responde que um determinado *checkpoint* local de uma MH não está disponível, então o algoritmo aborta o processo de construção do *checkpoint* global. O *log* de mensagens (log_i) de todas as mensagens enviadas por h_i, está disponível ao algoritmo de construção do *checkpoint* global, pois faz parte do *checkpoint* local. É necessário ainda identificar quais são as mensagens consideradas em trânsito, ou seja, as mensagens que foram enviadas por h_i mas não foram recebidas antes do *checkpoint* de destino acontecer. Para que esta tarefa seja executada dois vetores são mantidas nas MHs: $SENT_i[N]$ e $RECV_i[N]$. Os valores destes vetores são salvos sempre que h_i construir seu *checkpoint* local. Antes do envio de uma mensagem de aplicação para uma MH h_j , h_i incrementa $SENT_i[j]$ e acondiciona este valor na mensagem. Quando h_j recebe a mensagem, o valor de $RECV_j[i]$ será incrementado. Desta forma, as mensagens do log_i com um número de seqüência maior que $RECV_j[i]$ serão identificadas como sendo as mensagens em trânsito.

4.3.2 *Checkpoint* e rollback *recovery* por CONTICELLO & SARMA (1997)

Este protocolo é uma otimização da proposta de BADRINATH & ACHARYA (1994). A construção e armazenamento dos registros de *checkpoints* nas unidades de suporte, por parte das unidades móveis continua sendo o objetivo principal da proposta. O acréscimo de mais uma fase no processo de construção dos registros de *checkpoints* é a grande diferença em relação a proposta de BADRINATH & ACHARYA (1994).

Segundo os autores, os procedimentos de *checkpoint* estão voltados para o armazenamento periódico do estado dos processos em locais considerados como de armazenamento estável. Os procedimentos de *recovery* são emitidos na ocorrência da

falha de uma ou mais unidades, estes procedimentos tentam re-executar as operações a partir de algum *checkpoint* prévio. Na introdução da proposta os autores apresentam três classes de processos de *checkpoint* e *recovery:* ***optimistic***, ***pessimistic*** e ***consistent***. Os protocolos ***optimistic*** são baseados no registro em *log* das mensagens e o processo de *recovery* acontece através do re-envio das mensagens para as unidades que apresentaram falhas. Os protocolos do tipo ***pessimistic*** registram as mensagens e a informação de estado em *log* em cada evento que aconteça na unidade. A proposta dos autores é apresentar o terceiro tipo de protocolo: ***consistent checkpoint*** com ***rollback recovery***. A proposta apresentada pelos autores é baseada no conceito de *checkpoints* globais, para a construção de um registro de *checkpoint* global consistente é necessária a coleta de diversos *checkpoints* locais consistentes.

A execução de um processo é dividida em fases (phases) por meio da utilização de *checkpoints*. Uma determinada *phase i* é concluída por um *checkpoint i,* cada processo inicia sua execução a partir da *phase 1*. O conceito de vetores de dependência é utilizado na proposta de protocolo. Partindo do princípio que podem existir *N* processos em execução, um vetor de dependência será mantido em cada *site* e conterá *N* componentes. O $i^{ésimo}$ componente deste vetor representa a mais recente fase no processo *i* para a qual a fase local atualmente depende. De acordo com os autores, a fase corrente depende da fase *k* do processo *i* se:

- Uma mensagem foi recebida da fase *k* (ou maior) no processo *i.*
- Uma mensagem foi recebida da fase *m* no processo *p* que depende da fase *k* no processo *i.*

O vetor de dependência traz as informações sobre os registros de *checkpoint* que precisam ser incluídos para a construção de um *checkpoint* global. O protocolo prevê a manutenção dos vetores de dependência, esta manutenção é concluída em cinco etapas, descritas a seguir:

- Inicialmente todos os componentes terão valor zero, com exceção do componente *self*. (O $i^{ésimo}$ componente no processo *i* é definido como o componente *self*)

- O componente do vetor correspondente para *self* é atualizado sempre para o número da fase corrente.
- O vetor dependência atual é transmitido com cada mensagem enviada.
- Quando uma nova fase inicia, seu vetor dependência é o mesmo da fase precedente que acabou de ser concluída.
- Sempre que uma nova mensagem é recebida o vetor dependência é atualizado tomando o componente máximo entre o vetor de dependência local e o vetor de dependência recebido.

Os autores apresentam ainda o conceito da ***three phase rule***, esta regra permite a construção de *checkpoints* independentes e síncronos em diferentes unidades. Nesta proposta, um determinado *checkpoint* terá o conhecimento de todas as suas dependências após ter sido construído. O conceito de *three phase rule* é a divisão de uma *phase i* de um determinado processo em três subfases:

- ***Receive phase:*** nesta fase o processo mantém o recebimento de mensagens, esta fase termina quando o processo envia uma mensagem. O vetor dependência ao final desta fase torna-se o vetor dependência associado com o *checkpoint i.*
- ***Send phase:*** nesta fase o processo envia mensagens. A fase termina quando o processo efetua um recebimento.
- ***Static phase:*** aqui o processo pode receber ou enviar mensagens. Porém a condição para o recebimento de mensagens é que ela não poderia introduzir nenhuma nova dependência.

O processo de construção de um *checkpoint* global também é abordado nesta proposta. Quando um determinado processo decide construir um registro de *checkpoint* global, ele coleta o último *checkpoint* local e envia uma requisição para cada unidade *i* juntamente com o $i^{ésimo}$ componente do vetor de dependência de *checkpoint.* No recebimento de uma mensagem de *request* um processo verifica se o *checkpoint* requisitado já foi tomado. Caso o *checkpoint* solicitado já tenha sido construído, uma resposta positiva será enviada, caso contrário o *checkpoint* será forçado e a resposta será enviada. O iniciante espera por uma resposta de todos os participantes, observando um

timeout definido, e confirma o *checkpoint* global somente se todos os participantes confirmarem. A partir do momento em que um novo *checkpoint* global tenha sido construído, o registro de *checkpoint* antigo poderá ser descartado, isto é necessário tendo em vista que o processo de *rollback* é executado somente até o mais recente registro de *checkpoint.*

4.3.2.1 Esquema do rollback recovery

Segundo os autores, não existe a necessidade do envio de mensagens para todas as unidades do ambiente solicitando que executem um processo de *rollback* até o seu último *checkpoint* global consistente. A troca de mensagens traria um custo muito alto para o processo e a recuperação do estado consistente não é necessária para todas as unidades. A proposta dos autores pode ser apresentada através dos três itens a seguir:

- No caso de uma falha passageira, o processo que sofreu a falha informa os outros processos da ocorrência da falha. No caso de falhas permanentes, os autores partem do princípio que tais falhas serão detectadas dentro de um período de tempo por outras unidades.
- Em qualquer caso a notificação de falha contém o nome da unidade e o último *checkpoint* disponível na unidade, este *checkpoint* pode ou não ser parte integrante de um consistente *checkpoint* global. *Checkpoints* tomados após este último disponível são considerados como *lost states* da unidade que sofreu a falha.
- Na notificação da falha, cada unidade retrocede para o seu último *checkpoint* local, de tal forma que este *checkpoint* não contenha dependência para qualquer *checkpoint* da unidade que sofreu a falha. Se o vetor dependência atual atender a esta condição, nenhuma operação de *rollback* será necessária.

4.3.2.2 Localização dos *checkpoints*

A localização dos registros de *checkpoints* em um ambiente de computação móvel tem uma importância fundamental no processo de recuperação de falhas. O custo da pesquisa pela localização do *checkpoint* pode inviabilizar todo um processo de

recuperação. No protocolo sugerido por Acharya e Badrinath, ACHARYA & BADRINATH (1994), os *checkpoints* são armazenados nas estações de suporte (MSS), e esta informação é repassada ao longo de vetores de dependência. Os autores do presente protocolo sugerem uma alteração no local de armazenamento do *checkpoint* proposto por ACHARYA & BADRINATH (1994): (i) os *checkpoints* sempre serão armazenados em alguma localização fixa. Esta localização poderá ser o *home agent* da MH ou uma MSS; (ii) o *checkpoint* pode ser armazenado na MSS atual em todos os casos sem processo de *checkpoint* cruzando limites de células. A manutenção de ponteiros da MSS onde a unidade entrou na fase *send* para a MSS atual é proposta pelos autores. Existirá um número limite do tamanho desta cadeia, o *checkpoint* acontecerá quando este limite for excedido.

Os autores também apresentam a proposta da manutenção de *checkpoints* em unidades de armazenamento estável, somente para os registros de *checkpoints* globais. Desta forma, os *checkpoints* locais que não participam da construção de um *checkpoint* global, poderão ser armazenados nas unidades móveis. A estratégia de armazenamento nas unidades móveis foi descrita pelos autores da seguinte maneira:

- Os *checkpoints* são construídos e armazenados localmente.
- Quando uma requisição para construir um *checkpoint* global chega, o *checkpoint* relativo à solicitação é transmitido para a MSS e uma resposta positiva será enviada.

Quando uma unidade móvel desconecta da rede fixa, seu *checkpoint* pode não estar disponível para outras unidades construírem seus *checkpoints*. Neste caso, a unidade móvel poderá transmitir seu registro de *checkpoint* para a MSS antes do momento da sua desconexão. Em todos os casos a MSS irá atuar como um agente para a unidade móvel, no processo de construção de um *checkpoint* global.

4.3.3 Proposta de tolerância à falhas segundo ALAGAR et al. (1993).

Nesta proposta, os autores consideram o problema do desenvolvimento de sistemas de computação móvel que suportem a ocorrência de falhas nas estações de suporte. Os autores propõem dois esquemas para a tolerância de falhas de estações de suporte. Nos esquemas propostos a informação armazenada em uma estação de suporte

é replicada através de diversas estações de suporte secundárias. Na ocorrência de falha de uma estação de suporte, a unidade móvel sob a sua cobertura poderá se deslocar para o raio de ação de uma outra estação.

Os autores fazem uso no artigo, de algumas estruturas de dados que serão apresentadas a seguir. O vetor ***st_info(h)*** representa a informação de estado da unidade móvel ***h*** armazenada na MSS *h*. Todo o processo de comunicação envolvendo uma unidade móvel passa necessariamente pelo serviço da sua MSS. Desta forma, quando uma MSS sofre uma falha, a informação de estado da MH naquela célula será perdida. A unidade móvel não poderá operar até que a informação de estado esteja novamente disponível. Os autores sugerem a correção deste problema mantendo ***st_info(h)*** em diversas MSS. Para cada unidade móvel (MH) será associado um conjunto de MSSs secundárias que serão referenciadas por ***sec_mss(h)***. A MSS primária de uma determinada unidade móvel é denotada por ***pri_mss(h)***, é a MSS da célula onde *h* reside. Para cada *h, sec_mss(h)* representa a MSS onde a informação *st_info(h)* está replicada. Os autores ainda fazem referência a alguns termos: ***id(h)*** representa a identificação da MH *h,* ***sender(m)*** e ***dest(m)*** representam, respectivamente, o emitente e o destinatário da mensagem *m.*

O algoritmo proposto baseia-se na possibilidade de utilização das MSSs secundárias quando da ocorrência de falhas na MSS primária, os autores sugerem algumas estratégias para a escolha das MSSs secundárias. Na primeira estratégia os autores trabalham com o padrão de movimentação da unidade móvel para selecionar as MSSs secundárias. Segundo os autores, se o movimento da unidade móvel *h* está dentro de uma certa localização, então será suficiente considerar um conjunto fixo de MSSs daquela região como candidatas para a MSS secundária de *h.* A MSS dentro da qual a unidade móvel *h* reside é a *pri_mss(h),* e todas as outras MSSs dentro daquela região serão consideradas como participantes do conjunto de MSSs secundárias. Uma cópia de *st_info(h)* é mantida em cada MSS participante deste conjunto. Na ocorrência de uma falha de *pri_mss(h),* a unidade móvel *h* migra para uma das MSSs do conjunto *sec_mss(h)*, marca esta MSS como *pri_mss(h)* e executa o seu processamento.

Na segunda estratégia para a seleção das MSSs secundárias, os autores propõem a utilização das MSSs cujas células sejam adjacentes à célula da MSS primária ou mesmo

parcialmente sobrepostas a ela. Esta proposta parte do princípio que a movimentação da unidade acontecerá através das células vizinhas, sem levar em conta um padrão de movimentação como na proposta anterior. Uma cópia de *st_info(h)* necessita ser enviada para todas as novas MSSs secundárias de *h.*

4.3.3.1 Replicação pessimista para tolerância a falhas das MSSs.

Durante a operação normal, quando MSSs_1 envia uma mensagem (cujo destino seja a unidade móvel *h*) para *pri_mss(h)*, MSSs_1 armazena a mensagem em um *buffer*. Após a confirmação do recebimento da mensagem por *pri_mss(h),*MSSs_1 remove a mensagem do seu *buffer*. Caso s_1 receba uma mensagem notificando a falha de *pri_mss(h),* s_1 irá buscar a nova *pri_mss(h)* de *h* e retransmitirá a mensagem. Da mesma forma, quando uma unidade móvel envia uma mensagem para sua *pri_mss(h),* ficará no aguardo do recebimento da confirmação por parte da sua MSS primária. Caso receba um aviso da falha da sua MSS primária, irá estabelecer conexão com outra MSS para a retransmissão da mensagem. Este procedimento garante que nenhuma mensagem será perdida na ocorrência de falhas da MSS.

Quando a *pri_mss(h)* falha, *h* pode mudar para uma das MSSs em *sec_mss(h),* por exemplo *s.* A partir deste instante a MSS *s* torna-se a MSS primária de *h.* Todas as novas MSSs no conjunto *sec_mss(h)* possuem uma cópia de *st_info(h)* e todas as MSSs antigas que não participam mais do conjunto *sec_mss(h)*, excluem *st_info(h).* Se *h* muda-se para uma MSS que não está em *sec_mss(h),* então a informação *st_info(h)* é obtida de uma das MSSs que participam de *sec_mss(h).*

4.3.3.2 Proposta de replicação otimista para tolerância a falhas das MSSs.

A MSS primária de *h* atualiza *st_info(h)* no recebimento da mensagem *m* enviada para *h,* ou no recebimento da mensagem *m* de *h.* De maneira similar à proposta anterior, *pri_mss(h)* envia uma mensagem *copy(id(h),m)* para todas as MSSs em *sec_mss(h)*. Porém não irá aguardar pela confirmação de *sec_mss(h)*, a mensagem *m* será processada imediatamente pela *pri_mss(h)* e a atualização de *st_info(h)* acontecerá localmente. Após este processo a mensagem *m* será entregue para *dest(m).* Quando *pri_mss(h)* processa uma mensagem para a unidade móvel *h*, *st_info(h)* será atualizada

imediatamente após *pri_mss(h)* enviar uma mensagem *copy(id(h),m)* para todas as MSSs na *sec_mss(h).* Desta forma, *st_info(h)* na *pri_mss(h)* pode estar sendo atualizada enquanto que a mensagem *copy(id(h),m)* possa estar em trânsito para um ou mais membros de *sec_mss(h).* Esta situação contribui para a possibilidade da ocorrência de inconsistência de *st_info(h)* entre *pri_mss(h)* e *sec_mss(h)*. Na ocorrência de falha na *pri_mss(h)*, definida por alguma MSS responsável pela célula s_1, a unidade móvel se conectará com uma segunda MSS responsável pela cobertura da célula s_2. Os autores apresentam duas maneiras de sincronizar *st_info(h)* na célula s_2 e o estado de *h.* Uma das propostas seria o aguardo do recebimento e processamento por s_2 de todas as mensagens em trânsito, antes de permitir que *h* execute seu processamento. A outra proposta é reverter (*rollback*) *h* para um estado consistente com a nova MSS primária s_2.

4.3.3.3 Recuperação de falhas das MSSs (*mobile support station*)

O processo de recuperação de falhas das MSSs é executado segundo os processos tradicionais de recuperação. O que os autores especificam no estudo são os passos seguidos pela MSS após o seu processo de recuperação de falha. Supondo que MSS s_1 tenha falhado e, utilizando os métodos tradicionais de recuperação, voltou ao funcionamento normal, alguns passos terão necessidade de serem executados:

- Quando s_1 se recupera após uma falha, ela informa a todas as outras MSSs sobre a sua recuperação.
- Quando s_2 recebe a mensagem de recuperação de s_1 , s_2 informa sobre a recuperação de s_1 para todas as unidades móveis da sua célula.
- Caso alguma unidade móvel tenha migrado para s_2 como resultado da falha de s_1 , então s_2 envia para s_1 a lista de todas as unidades móveis que migraram de s_1 para s_2 .
- No recebimento da lista, s_1 irá apagar os registros de *st_info(h)* para todas as unidades da lista. Se s_1 é uma MSS secundária para uma unidade móvel em s_2, s_2 envia *st_info(h)* para s_1.

4.3.3.4 Recuperação de falhas das unidades móveis

Segundo os autores, se *st_info(h)* armazenado na *pri_mss(h)* de uma unidade móvel *h* contém o estado completo de *h,* ou informações suficientes para recriar o estado de *h* antes da falha, o processo de recuperação será simples. A unidade móvel *h* recebe seu estado diretamente da *pri_mss(h)* e executa o seu processamento.

Caso as informações mantidas em *st_info(h)* não sejam suficientes para a recuperação da unidade móvel, dados adicionais precisarão ser mantidos em *pri_mss(h)* para que se conclua a recuperação. Uma possibilidade seria o registro em um *log* das mensagens recebidas pela unidade móvel *h* na *pri_mss(h),* e a construção do *checkpoint* do estado de *h*. Para a recuperação da falha, *h* recebe seu estado mais recente de *checkpoint* e executa novamente as mensagens armazenadas após o estado recente que sofreu o *checkpoint.* A unidade móvel processa as mensagens para a sua recuperação até o instante anterior à falha.

Outra possibilidade é contar com a informação completa do estado da unidade móvel armazenada em *pri_mss(h)*. Quando uma mensagem *m* é enviada para uma unidade móvel *h,* o estado de *h* é alterado. O estado de *h* também será alterado localmente em *pri_mss(h)* usando a mensagem *m.* Desta forma, o estado de *h* armazenado em *pri_mss(h)* é sempre mantido em sincronia com o estado da unidade móvel *h.*

Segundo os autores, os esquemas de recuperação de falhas em unidades móveis podem ser integrados com os processos para tolerância a falhas das MSSs, garantindo desta forma, a disponibilidade do ambiente móvel. Para uma unidade móvel *h,* seu estado com o respectivo *checkpoint* e as mensagens armazenadas em *log* na *pri_mss(h)* podem formar uma parte do *st_info(h)*. Como *st_info(h)* é replicado para as *sec_mss(h)*, se *h* e *pri_mss(h)* falharem ao mesmo tempo, após a recuperação *h* pode alternar para uma das MSS em *sec_mss(h)* com a intenção de pegar seu estado que sofreu o *checkpoint* e as mensagens armazenadas em *log,* sem a necessidade de aguardar pela recuperação de *pri_mss(h).*

4.3.3.5 Procedimento de *handoff*

A possibilidade de movimentação das unidades é a tônica nos ambientes de computação móvel. A unidade móvel *h* estará inicialmente na célula de MSS s_1. A

unidade *h* move-se então para a célula de MSS s_2. A unidade estabelece comunicação com s_2 e envia o identificador da sua MSS antiga (s_1) para s_2. Um procedimento de *handoff* será executado para a transferência das informações relevantes. A MSS s_1 envia *st_info(h)* para s_2. Caso s_2 corresponda a *sec_mss(h), st_info(h)* não será enviado para s_2. Se s_2 não corresponder a *sec_mss(h),* s_1 aguarda o recebimento de todas as confirmações pendentes de *sec_mss(h),* atualiza *st_info(h),* e envia *st_info(h)* para s_2. A MSS s_1 também envia todas as mensagens pendentes para s_2.

4.3.4 Proposta de recuperação de falhas segundo PRADHAN et al. (1996)

No estudo, os autores fixaram a atenção nos esquemas que possibilitam a recuperação de uma falha de unidade móvel independentemente de outras unidades móveis.

4.3.4.1 Estratégias de recuperação *state saving*

Estas estratégias são baseadas nas tradicionais técnicas de *checkpoint* e *log* de mensagens. Nestas estratégias, a unidade salva periodicamente seu estado em local de armazenamento estável. Na ocorrência de falha da unidade, a execução pode ser reiniciada a partir do último *checkpoint* salvo. O armazenamento do *log* e do *checkpoint* é necessário que aconteça em local estável, a unidade móvel não é considerada um local estável. Desta forma, o algoritmo armazena estes valores na BS local da unidade móvel. A BS local é a BS que a unidade está atualmente residindo, é possível então que diversos registros de *checkpoint* estejam armazenados em diversas BSs, conforme a movimentação da unidade móvel. Duas estratégias para *state saving* foram apresentadas pelos autores: (i) *No Logging* e (ii) *Logging.*

Proposta *No Logging:* nesta proposta o estado do processo pode ser alterado tanto na recepção de uma mensagem de outra unidade quanto através de entradas dos usuários. As mensagens ou entradas que modificam o estado são chamados de eventos *write*. O estado da unidade móvel é salvo na *base station* em cada evento *write* da unidade móvel. Após a ocorrência da falha, a unidade envia uma mensagem para a BS,

que transfere a último estado salvo para a unidade móvel. A unidade móvel carrega o último estado e continua o processamento a partir daquele ponto.

Proposta *Logging:* neste esquema a unidade móvel constrói *checkpoints* periodicamente. Os eventos *write* que ocorrem entre os *checkpoints* também farão parte do *log.* Se uma mensagem *write* é recebida de outra unidade, a BS faz o registro desta mensagem em *log* e então a envia para que a unidade móvel execute o seu processamento.

No recebimento de evento *write* de outra unidade
Registro do evento em *log* da *Base Station;*
Envio do evento para execução na unidade móvel;

Sob uma entrada de usuário, o processo acontece da mesma maneira: a unidade móvel envia uma cópia da entrada para que a sua BS registre no *log*. Após a efetivação do registro em *log*, a BS envia um aviso de confirmação para a unidade móvel. A unidade móvel poderá processar a entrada enquanto espera pela confirmação, mas não poderá enviar uma resposta, somente após o recebimento da confirmação a unidade móvel envia sua resposta. O processo de registro em *log* acontece até que um novo processo de *checkpoint* seja copiado para a BS, a partir deste instante a BS retira do *log* os registros antigos. Após uma falha, quando a unidade móvel reinicia, envia uma mensagem para a sua BS, que transfere o último processo de *checkpoint* da unidade e os seus respectivos registros de *log.* A unidade móvel carrega o seu último *checkpoint* e re-executa as mensagens recebidas do *log.*

No retorno da falha (unidade móvel)
Request para *Base Station* solicitando *checkpoint* e *log;*
Restaura ultimo *checkpoint;*
Reexecuta mensagens do *log;*

4.3.4.2 Estratégias de *handoff*

Durante o processo de *handoff* é possível de se efetuar a transferência de algumas informações relativas ao estado da unidade móvel. Os autores propõem três métodos para a transferência desta informação durante o processo de *handoff* : (i) *Pessimistic,* (ii) *Lazy e* (iii) *Trickle.*

Estratégia *Pessimistic:* quando a unidade move-se de uma célula para outra, o processo de *checkpoint* é transferido para a nova BS durante o processo de *handoff*. Se a estratégia *logging* estiver sendo utilizada, o *log* de mensagens é também transferido para a nova BS. No recebimento do processo de *checkpoint* ou do *log*, a nova BS envia uma confirmação de recebimento para a BS antiga. Quando a BS antiga recebe esta confirmação, exclui as suas cópias do *checkpoint* e do *log*. A grande desvantagem desta estratégia é a grande quantidade de dados transferidos durante cada processo de *handoff.*

Estratégia *Lazy*: a proposta desta estratégia não é a transferência dos registros de *checkpoint* e de *log,* ao invés deste procedimento uma lista das BSs visitadas pela unidade móvel será criada. Na ocorrência de um *handoff,* a nova BS terá apenas o registro da última BS daquela unidade móvel. Desta forma, como uma unidade pode movimentar-se de célula para célula, as BSs visitadas formam uma *linked list*. Sempre que uma BS recebe um *checkpoint,* uma notificação será enviada para a última BS solicitando a exclusão dos processos de *checkpoint* e dos registros de *log* daquela unidade móvel, se eles existirem naquela BS. Caso estes registros não existam na BS anterior, a solicitação será repassada para as BS predecessoras, até que o último registro de *checkpoint* seja encontrado. Esta proposta diminui as informações a serem transferidas no processo de *checkpoint*.

Estratégia *Trickle*: nesta proposta, o intuito foi garantir que os *logs* e os *checkpoints* estejam sempre próximos da BS. Precisa existir a garantia de que estes registros estejam localizados na BS predecessora da BS atual. Para que isto seja alcançado, durante o processo de *handoff,* uma mensagem de controle é enviada para a BS predecessora solicitando a transferência das informações relativas à unidade móvel. A BS corrente também envia uma mensagem de controle para a nova BS indicando a última localização da unidade móvel (a célula corrente). Então, a nova BS apenas relembra a última BS da unidade móvel. Se um *checkpoint* for tomado na BS corrente, ela envia uma notificação para a BS que possui os últimos registros de *checkpoint* e *log* solicitando que os mesmos sejam excluídos. Durante o processo de recuperação, se a BS corrente não tem um *checkpoint,* ela irá solicitar o registro para a BS predecessora. A BS transfere as informações para a unidade móvel. A unidade móvel carrega o *checkpoint* e reaplica as mensagens do *log* para alcançar o estado anterior à falha.

4.4 Protocolos de recuperação de falhas do tipo compostos

O protocolo classificado sob este item não apresenta um mecanismo de coordenação para as transações entre agrupamentos. As transações intra agrupamentos acontecem com a utilização de um mecanismo de coordenação.

4.4.1 Estrutura de agentes segundo PITOURA & BHARGAVA (1995)

Nesta proposta, os autores abordam a utilização de agentes para sugerir uma estrutura que forneça suporte a recuperação de falhas em ambientes móveis. O problema principal a ser tratado é a adequação das propriedades ACID para um ambiente de banco de dados móvel. A utilização de agentes permite uma maior independência das unidades e menor tráfego de mensagens na rede. Segundo os autores, um agente é, basicamente, um objeto que encapsula dados e procedimentos a serem executados pela unidade receptora do agente. A execução de um agente ocorre através da execução dos seus procedimentos sobre os seus dados. Ao invés de contar com o tráfego de mensagens, a rede irá carregar os agentes para o seu destino final.

Cada sistema de banco de dados envolvido no ambiente, apresenta um conjunto de operações denominadas *primitive database methods*. O relacionamento das aplicações é possível através da utilização de *application agents*. Um *application agent* só poderá acessar os itens de dados através da utilização dos *primitive database methods.* A coordenação do acesso aos dados é responsabilidade dos *database agents,* que ainda agem nos processos de consistência dos sistemas de banco de dados e recuperação de falhas. A comunicação entre os agentes de aplicação é possível pela solicitação de métodos de acesso aos dados mantidos em outros agentes. Segundo os autores, estas operações são selecionadas de um conjunto pré-definido de *primitive methods* chamado de *primitive application methods*. No estudo, duas definições para agentes e métodos são apresentadas pelos autores, estas definições são descritas abaixo:

- **Agente:** um agente é um objeto (*D, M, SD, P*), onde *D* é um conjunto de dados locais, *M* é um conjunto de métodos, *SD* é um conjunto de dependências estruturais entre métodos e *P* um conjunto de pontos de quebra e realocação.

- **Método:** os autores definem método em três possibilidades, *(i)* uma primitiva de banco de dados ou método de aplicação que acessa dados de um banco de dados ou outro agente, *(ii)* um método local que acessa os dados locais do agente, *(iii)* uma combinação de métodos e primitivas locais.

Os agentes de aplicação são constituídos de um conjunto de métodos e um determinado número de fluxos de controle. O fluxo de controle de um determinado agente determina a ordem de execução dos métodos do agente, esta ordem de execução é baseada em uma estrutura determinada dependência estrutural. A cooperação entre os agentes é uma característica do ambiente, os *breakpoints* definem os pontos de interação entre os agentes, já os *relocation points* definem o ambiente de execução do agente. A seguir serão apresentadas as definições de **dependência estrutural**, ***breakpoint*** e ***relocation points*** conforme a proposta dos autores:

- **Dependência estrutural:** a dependência estrutural é uma tripla (*C, M, S*), onde *C* é uma especificação, *M* é um conjunto de métodos e *S* é um conjunto de estados controláveis do conjunto de métodos *M.* Para uma primitiva *M,* um estado controlável em *S* poder ser *commit, abort, prepare-to-commit* ou *submit*.

- ***Breakpoint:*** um *breakpoint* de um agente *A* é uma tripla (*Bs, Be, {(Ai, Mj)})* onde *{(Ai, Mj)}* é um conjunto de pares de métodos e agentes. *Bs e Be* são estados controláveis de métodos em *A* que permitem membros de *{(Ai, Mj)}* serem executados entre os estados *Bs e Be* de *A*.

- ***Relocation point:*** um *relocation point* de um determinado agente *A* é uma tripla *(Bs, Be, (A', B's))* onde *Bs* e *Be* são estados controláveis de métodos em *A*. *A'* identifica um segundo agente, *B's* é um estado controlável de um método em *A'* e determina que a parte de *A* entre os estados *Bs* e *Be* é executada como parte de *A'* após *B's*.

No estudo, os autores definem a informação que necessita ser recuperada após uma falha como ***context***. No *context* estarão englobados também, os dados locais do agente e o seu estado de processamento. As informações relativas a cada agente estarão armazenadas em um banco de dados chamado de *context database*. Os autores assumem que o banco de dados *context* será armazenado em um local estável e livre de falhas.

As atualizações nas informações do banco de dados *context* não serão sobrescritas. A proposta é manter registros correspondentes às várias versões de atualização da informação. Desta forma, o último valor escrito para um determinado item de dados através de um método que já foi confirmado (*commit*), será reconhecido como o último valor confirmado (*committed*) para aquele item de dados. Um estado confirmado do banco de dados *context* com relação a uma determinada execução é o estado no qual cada item de dados apresenta o seu último valor confirmado.

Segundo os autores, quando uma falha de um determinado *site* ocorrer depois que um método foi executado com sucesso, mas antes dos seus resultados terem sido tornados permanentes, o método em questão será desfeito (undo). No decorrer do processo o método poderá ser executado novamente ou considerado abortado. As falhas decorrentes da execução de um agente cooperado serão manuseadas utilizando o *context* local do agente, neste caso um estado confirmado do banco de dados *context* será restaurado. Nas falhas de comunicação, quando um agente é perdido, ele será reconstruído utilizando o seu respectivo contexto. Segundo os próprios autores, nesta proposta não existe a possibilidade da aplicação da propriedade de atomicidade na sua totalidade. Isto decorre do fato de que vários agentes podem tomar decisões independentes com relação à confirmação ou aborto de um determinado método submetido por um agente de aplicação.

4.5 Análise comparativa entre as propostas de protocolos para recuperação de falhas em bancos de dados móveis

Neste item sera apresentada uma análise comparativa entre as propostas de protocolos citadas neste capítulo. Esta análise foi construída através da verificação do

tratamento dado, pelo protocolo, a algumas características definidas. As características que servem como parâmetro para esta análise são as seguintes:

- **Localização da área de armazenamento estável:** no ambiente de bancos de dados móveis, uma área da rede é considerada como de armazenamento estável, as propostas alternam em definir a localização desta área.
- **Local de armazenamento dos registros de *checkpoints* e *log*: a** localização dos registros de *checkpoint* e *log* poderá traduzir-se em maior ou menor tráfego de mensagens no ambiente móvel.
- **Previsão da ocorrência da falha:** alguns protocolos partem do princípio de que as ocorrências de falhas serão antecipadas, seja pelo sistema ou pelo usuário. Outros protocolos já trabalham com a possibilidade de ocorrência da falha sem antecipação prévia.
- **Utilização de estrutura de dependência entre as unidades:** alguns protocolos, na tentativa de limitar o número de unidades envolvidas na criação de um estado global consistente, fazem uso de uma estrutura de dependência entre as unidades. Limitando desta forma, o custo na obtenção do estado global.
- **Construção de registros de *checkpoints* através da troca de mensagens:** a troca de mensagens para a construção de estado globais, em um ambiente de computação móvel, pode elevar o custo computacional com a comunicação entre as unidades.

No decorrer do capítulo, para cada proposta será feita uma análise comparativa de todas as características citadas, verificando desta forma a aplicação do item pela proposta em questão. Ao final deste capítulo, encontra-se a tabela (Tabela 1) comparativa resumindo as características dentre as diferentes propostas analisadas.

4.5.1 Análise da proposta de NEVES & FUCHS (1997)

Localização da área de armazenamento estável: os autores consideram apenas as estações da rede estacionária como capazes de oferecerem locais de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: os registros de *checkpoints* do tipo *hard* são armazenados nas estações de suporte, localizadas na rede estacionária. Enquanto que os registros de *checkpoint* do tipo *soft* são mantidos nas unidades móveis.

Previsão de ocorrência da falha: Nesta proposta os autores apresentam um protocolo adaptado tanto para as falhas possíveis de antecipação quanto para as não possíveis. Nas falhas onde a antecipação é possível, a criação de um novo estado global antes da sua ocorrência evitará que o trabalho efetuado seja revertido. Quando a antecipação não for possível, o trabalho efetuado por aquela unidade será revertido até que se encontre o último estado global armazenado.

Utilização de estrutura de dependência entre as unidades: nesta proposta, os autores não apresentam nenhuma estrutura de dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: a construção dos *checkpoints* não acontece através da troca de mensagens, mas sim através de um período de sincronismo determinado.

4.5.2 Análise da proposta de PRAKASH & SINGHAL (1995)

Localização da área de armazenamento estável: nesta proposta as unidades móveis são consideradas também como capazes de prover locais de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: as mensagens registradas no arquivo de *log* são armazenadas exclusivamente nas estações de suporte à mobilidade, localizadas na rede estacionária. Já os registros de *checkpoints* são armazenados tanto nas estações de suporte quanto nas unidades móveis.

Previsão de ocorrência da falha: os autores desta proposta consideram que a ocorrência das falhas não poderá ser prevista antecipadamente.

Utilização de estrutura de dependência entre as unidades: nesta proposta os autores apresentam uma estrutura de dependência entre as unidades, apenas as unidades que apresentam alguma dependência estarão envolvidas no processo de construção do estado global.

Construção de registros de *checkpoints* através da troca de mensagens: a construção dos registros de *checkpoints* é obtida através da troca de mensagens entre as unidades relacionadas.

4.5.3 Análise da proposta de BADRINATH & ACHARYA. (1994)

Localização da área de armazenamento estável: apenas as unidades de suporte situadas na rede estacionária são consideradas como de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: os autores sugerem o armazenamento dos *checkpoints* (globais e locais) e das mensagens de *log* nas unidades de suporte à mobilidade (MSS). Cada unidade móvel possui uma MSS definida como local em um determinado instante de tempo, desta forma, durante a movimentação da unidade é possível que registros de *checkpoints* válidos estejam disponíveis em várias MSS. O protocolo apresenta um mecanismo para minimizar o custo da busca pelos *checkpoints* válidos.

Previsão de ocorrência da falha: Nesta proposta os autores consideram que as falhas nas unidades móveis podem ser antecipadas. Desta forma, antes de acontecer a falha a unidade móvel precisa construir o seu registro de *checkpoint* local para que as unidades suporte (MSS) possam ter acesso a este registro.

Utilização de estrutura de dependência entre as unidades: não existe nenhuma estrutura que apresente a dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: nesta proposta os autores sugerem a construção *checkpoints* fazendo uso da troca de mensagens.

4.5.4 Análise da proposta de ALAGAR et al. (1993)

Localização da área de armazenamento estável: nesta proposta o local de armazenamento estável considerado pelos autores é a estação de suporte. Os autores propõem ainda a disseminação das informações mantidas em local estável por várias estações de suporte.

Local de armazenamento dos registros de *checkpoints* e *log*: no protocolo sugerido pelos autores, os registros de *checkpoint* e de *log* são mantidos em várias unidades de suporte (MSS). A replicação destas informações por diversas MSSs sugere

o aumento da disponibilidade do sistema. A disseminação destas informações pode atender a um padrão de movimentação da unidade ou apenas englobar as MSSs vizinhas a MSS local.

Previsão de ocorrência da falha: a impossibilidade de previsão da ocorrência de falha é mantida nas duas situações possíveis: falha da MSS ou falha das unidades móveis.

Utilização de estrutura de dependência entre as unidades: esta proposta apresenta a utilização de uma estrutura que permite a identificação da dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: todo o processo de construção de *checkpoints* acontece através da troca de mensagens entre as unidades envolvidas.

4.5.5 Análise da proposta de PRADHAN et al. (1996)

Localização da área de armazenamento estável: apenas as unidades de suporte são consideradas como locais de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: os autores apresentam a possibilidade de trabalhar ou não com registros de *log*. Em qualquer uma das formas de trabalho, o registro dos *checkpoints* e *log* (se houver) estarão armazenados nas estações de suporte (MSS) localizadas na rede estacionária.

Previsão de ocorrência da falha: este protocolo parte do princípio de que as falhas das unidades móveis podem ser previstas e antecipadas as suas ocorrências para as estações de suporte (MSS). Uma forma de prever a ocorrência da falha seria, segundo os autores, o envio periódico de mensagens, a partir da unidade móvel, indicando a sua atividade.

Utilização de estrutura de dependência entre as unidades: não existe, nesta proposta, nenhuma estrutura de dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: nesta proposta, os autores apresentam a construção dos *checkpoints* tanto através da troca de mensagens quanto através de um período determinado, sem a necessidade de troca de mensagens.

4.5.6 Análise da proposta de CONTICELLO & SARMA (1997)

Localização da área de armazenamento estável: as unidades móveis, nesta proposta, podem armazenar alguns registros de *checkpoints*, porém somente aqueles registros que não estejam participando de um *checkpoint global*. Para os *checkpoints* globais apenas as estações de suporte são consideradas como de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: esta proposta permite o armazenamento de *checkpoints* tanto nas unidades móveis quanto nas unidades de suporte (MSS). Os registros de *checkpoints* globais serão sempre armazenados nas MSSs. Enquanto que os registros de *checkpoints* locais poderão ser armazenados nas unidades móveis, desde que não estejam participando da construção de um registro de *checkpoint* global, neste caso o armazenamento acontecerá nas MSSs.

Previsão de ocorrência da falha: os autores da proposta trabalham com a possibilidade de antecipação da falha classificada como passageira, já as falhas do tipo permanentes serão detectadas pelas unidades dentro de um determinado período de tempo.

Utilização de estrutura de dependência entre as unidades: o conceito de vetores de dependência é utilizado nesta proposta, no intuito de limitar a pesquisa pelas unidades envolvidas no processo.

Construção de registros de *checkpoints* através da troca de mensagens: a construção dos *checkpoints* envolve o envio de mensagens entre as unidades envolvidas.

4.5.7 Análise da proposta de protocolo unilateral (UCM)

Localização da área de armazenamento estável: esta proposta apresenta apenas as unidades da rede estacionária como local de armazenamento estável.

Local de armazenamento dos registros de *checkpoints* e *log*: segundo os autores, os registros de *checkpoint* e de *log* são armazenados em unidades da rede estacionária.

Previsão de ocorrência da falha: nesta proposta os autores partem do princípio de que as falhas podem acontecer a qualquer instante, sem o aviso para o coordenador do protocolo.

Utilização de estrutura de dependência entre as unidades: nesta proposta, não existe nenhuma estrutura de dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: todo o mecanismo do protocolo envolve a troca de mensagens entre os participantes.

4.5.8 Análise da proposta de PITOURA & BHARGAVA (1995)

Localização da área de armazenamento estável: neste estudo, os autores definem como *context* a informação que necessita ser recuperada após uma falha. As informações relativas a cada agente estarão armazenadas em um banco de dados chamado de *context database*. Os autores assumem que o banco de dados *context* será armazenado em um local estável e livre de falhas, sem especificar qual seria este local.

Local de armazenamento dos registros de *checkpoints* e *log*: o conceito de *checkpoint* e *log* não é abordado pelos autores, os agentes teriam englobados na sua composição as instruções e dados relativos à necessidade da recuperação de falhas.

Previsão de ocorrência da falha: os autores deste protocolo consideram que as desconexões (falhas) das unidades móveis com relação à rede estacionária são passíveis de previsão.

Utilização de estrutura de dependência entre as unidades: não existem estruturas de dependências nesta proposta.

Construção de registros de *checkpoints* através da troca de mensagens: a troca de mensagens foi substituída por requisições para os agentes, que por sua vez irão trafegar e interagir no ambiente para atender à requisição.

4.5.9 Análise da proposta de WALBORN & CHRYSANTHIS (1998)

Localização da área de armazenamento estável: a proposta do *Pro Motiom* adota como local de armazenamento estável as estações de suporte à mobilidade (MSS), estas estações são unidades da rede estacionária através das quais a comunicação entre as unidades móveis e a rede fixa é possível.

Local de armazenamento dos registros de *checkpoints* e *log*: na proposta do *Pro Motiom* existe um local para armazenamento de registros de *log* nas unidades de suporte (MSS), estas unidades de suporte podem estar localizadas fisicamente no mesmo equipamento que abriga o servidor de banco de dados quanto em um *site* distinto. Nas unidades móveis o *compact agent* mantém um registro de *log,* este registro de *log* armazena os eventos executados durante a operação desconectada da unidade.

Previsão de ocorrência da falha: na proposta do *Pro Motiom,* os autores consideram a possibilidade da ocorrência de falhas intencionais (previsíveis) bem como de falhas não intencionais (não previsíveis), adaptando o protocolo para o funcionamento nas duas situações.

Utilização de estrutura de dependência entre as unidades: não está prevista nesta proposta, a utilização de estruturas de dependência entre as unidades.

Construção de registros de *checkpoints* através da troca de mensagens: da mesma forma que na proposta de PITOURA & BHARGAVA (1995), os agentes estão encarregados de trabalhar com as solicitações das unidades.

4.6 Classificação das propostas de recuperação de falhas

Conforme a taxonomia adotada, a tabela 1 apresenta a classificação de cada proposta apresentada nos itens anteriores.

Tabela 1 - Classificação das propostas de recuperação de falhas

Classificação das propostas de recuperação de falhas

Coordenados		**Compostos**		**Não coordenados**	
Unidades de Suporte	**Unidades Móveis**	**Unidades de Suporte**	**Unidades Móveis**	**Unidades de Suporte**	**Unidades Móveis**
	NEVES & FUCHS (1997)		PITOURA & BHARGAVA (1995)		BADRINATH & ACHARYA. (1994)
	PRAKASH & SINGHAL (1995)				CONTICELLO & SARMA (1997)
	Protocolo *UCM*			ALAGAR et al. (1993)	ALAGAR et al. (1993)
	Pro Motiom				PRADHAN et al. (1996)

4.7 Resumo da análise comparativa entre os protocolos

Neste item é apresentado um resumo da análise das propostas, a Tabela 2 auxilia na visualização das características de cada protocolo estudado.

Tabela 2 - Resumo da análise comparativa entre os protocolos.

	Localização da área de armazenamento estável	**Local de armazenamento dos registros de *checkpoint* e *log***	**Previsão da ocorrência de falhas**	**Estruturas de Dependência**	**Troca de mensagens**
NEVES & FUCHS (1997)	Unidades de suporte	*Hard checkpoints* nas unidades de suporte. *Soft checkpoints* nas unidades móveis	Falhas previsíveis e imprevisíveis.	Não	Não
PRAKASH & SINGHAL (1995)	Unidades de suporte e unidades móveis	Registros de *log*: unidades de suporte. Registros de *checkpoints*: unidades de suporte e unidades móveis.	Falhas previsíveis e imprevisíveis.	Sim	Sim
BADRINATH & ACHARYA (1994)	Unidades de suporte	Unidades de suporte	Falhas previsíveis	Não	Sim
ALAGAR et al. (1993)	Unidades de suporte	Replicação nas unidades de suporte	Falhas previsíveis e imprevisíveis.	Sim	Sim
PRADHAN et al. (1996)	Unidades de suporte	Unidades de suporte	Falhas previsíveis.	Não	Com e sem troca de mensagens
CONTICELLO & SARMA (1997)	Unidades de suporte	*Checkpoints* locais: unidades móveis. *Checkpoints* globais: unidades de suporte.	Falhas previsíveis.	Sim	Sim
Protocolo Unilateral (UCM)	Unidades de suporte	Unidades de suporte	Falhas previsíveis e imprevisíveis.	Não	Sim
PITOURA & BHARGAVA (1995)	*	*	Falhas previsíveis.	Não	Agentes
WALBORN & CHRYSANTHIS (1998)	Unidades de suporte	Unidades de suporte	Falhas previsíveis e imprevisíveis.	Não	Agentes

***** sem especificação

De acordo com os resultados obtidos na análise comparativa fica evidente o fato de que, apesar do desenvolvimento das capacidades de armazenamento e processamento, as unidades móveis não são consideradas locais estáveis para o armazenamento de informações. Apenas a proposta de PRAKASH & SINGHAL (1995) considera a unidade móvel como local estável, no entanto os autores procuram limitar a quantidade de informações armazenadas na unidade móvel.

A análise dos resultados da pesquisa em relação ao local de armazenamento dos registros de *checkpoint* e *log* revela que todos os protocolos, à exceção da proposta de PITOURA & BHARGAVA (1995) que não especifica o local de armazenamento, sugerem as estações de suporte para o armazenamento destes registros. As propostas de NEVES & FUCHS (1997), PRAKASH & SINGHAL (1995) e CONTICELLO & SARMA (1997) sugerem, além do armazenamento nas estações de suporte, a utilização das unidades móveis para o registro de alguns *checkpoints* (*soft* e locais).

Observando os resultados tabulados anteriormente, verifica-se que para todas as propostas onde existe a figura de uma estrutura de dependência, o processo de criação de um registro de *checkpoint* envolve a troca de mensagens. O que confirma a função da estrutura de dependência: limitar a pesquisa no momento de criação do registro do *checkpoint.* Estes protocolos trabalham na otimização do custo com o processamento para a construção dos registros de *checkpoints*.

Da mesma forma pode ser observado que para os protocolos que não envolvem a troca de mensagens, inexiste a figura da estrutura de dependência. Confirmando mais uma vez a função desta estrutura.

Verificou-se, no item de previsão de ocorrência de falhas, que 50% das propostas trabalham com a possibilidade de antecipação da ocorrência da falhas, enquanto que outros 50% trabalham com a impossibilidade de antecipação da falha.

Dos resultados apresentados fica aparente a classificação das unidades móveis como locais de armazenamento não confiáveis. Desta forma as propostas carregam sobre as unidades de suporte a responsabilidade de gerenciar todo o processo de mobilidade do ambiente, além da integridade dos bancos de dados existentes nesta arquitetura. O papel das unidades de suporte, em todas as propostas de protocolos analisadas, é vital para a construção de mecanismos de tolerância à falhas destes

ambientes. Este papel pode ser verificado a partir da análise dos resultados relativos ao local de armazenamento dos registros de *checkpoint* e *log,* todas as propostas armazenam estas informações (vitais para a implementação dos mecanismos) nas unidades de suporte. Porém, somente uma das propostas apresenta algum mecanismo que também contemple a ocorrência de falhas nas unidades de suporte, a ocorrência de falhas nestas unidades compromete todo o funcionamento do ambiente.

Sob os aspectos analisados, a proposta de ALAGAR et al. (1993) apresenta-se como a mais abrangente, nesta proposta os autores trabalham tanto com a possibilidade de antecipação da ocorrência de falhas, quanto com a imprevisibilidade destas ocorrências. O algoritmo aborda a ocorrência de falhas por solicitação do usuário (pedidos de desconexão) e as falhas imprevisíveis (ex: queda do *link* de comunicação). Neste algoritmo não existe a necessidade de qualquer mecanismo de sincronização das unidades móveis, permitindo que a coleta de registros para a construção de um estado consistente do banco de dados ocorra de forma independente entre as unidades móveis. A replicação dos registros relativos ao suporte da mobilidade das unidades, entre as unidades de suporte, sugere um aumento na disponibilidade do sistema. Uma vez que, mesmo que ocorram falhas em alguma unidade de suporte, o ambiente não será afetado, pois as informações poderão ser recuperadas nas unidades de suporte vizinhas, possibilidade garantida pela replicação destas informações.

CAPÍTULO V

ANÁLISE DOS MECANISMOS DE RECUPERAÇÃO DE FALHAS IMPLEMENTADOS POR SGBDs COMERCIAIS EM AMBIENTES DE COMPUTAÇÃO MÓVEL

No capítulo anterior foram apresentadas algumas das propostas encontradas na literatura científica, com relação à recuperação de falhas em ambientes de bancos de dados móveis. A proposta deste capítulo é apresentar os mecanismos aplicados nos sistemas gerenciadores de bancos de dados mais utilizados no mercado atual. Não será feita em nenhum momento, e nem é esta a intenção deste trabalho, qualquer análise comparativa com relação a desempenho ou capacidade de processamento entre os aplicativos abordados. Foram escolhidos os aplicativos dos fornecedores que detêm no conjunto, o mercado mundial de sistemas gerenciadores de banco de dados: *IBM (DB2 Everyplace), Microsoft (SQL Server CE), Oracle (Oracle Lite)* e *Sybase (SQL Anywhere Studio)*.

5 SGBDs Comerciais - a recuperação de falhas em ambientes móveis

A pesquisa na área de computação móvel é bastante intensa na atualidade. Tendo em vista a rápida evolução desta área, por vezes propostas de arquiteturas ou algoritmos que surgem na literatura científica não são aplicadas na totalidade nos sistemas comerciais. Serão apresentadas a seguir as propostas de recuperação de falhas empregadas pelos aplicativos citados na introdução deste capítulo.

5.1 SQL Server CE

O *MS SQL SERVER 2000* é a mais recente versão do sistema gerenciador de banco de dados distribuídos da *Microsoft*. A empresa disponibilizou várias edições do produto, cada qual com sua aplicação e capacidades específicas (SQL SERVER CE, 2002): *Enterprise*, *Standard*, *Personal*, *Developer* e*Windows CE***.**

Os bancos de dados no *SQL SERVER 2000* possuem pelo menos um arquivo para o registro dos dados e outro para o registro das transações, o *transaction log*. Todas as informações sobre as transações que alteram o estado do banco de dados são registradas de forma serial no *transaction log*, estas informações possibilitam as operações de *redo* ou *undo* das transações. Os estados dos dados antes e depois das modificações são registrados no *transaction log*, bem como as operações lógicas que foram efetuadas sobre os itens de dados.

Dentro do *transaction log* existem pontos de verificação (*checkpoints*), a cada *checkpoint* o mecanismo do *SQL SERVER* garante que todos os registros do *log* já foram escritos em disco. Durante o processo de recuperação dos bancos de dados, uma transação necessita ser reaplicada somente quando não existe a confirmação de que todas as modificações de dados da transação foram escritas para o disco. Tendo em vista de que o registro de *checkpoint* força o envio de todas as páginas modificadas para o disco, este registro pode ser considerado como o ponto a partir do qual se dará início ao processo de recuperação para a rescrita das transações.

Caso ocorra alguma falha na comunicação entre o cliente e o servidor de banco de dados (ex: conexão da rede com o cliente é interrompida) enquanto a transação estiver ativa, o mecanismo do banco de dados irá retroceder a transação no momento em que for notificado da falha pelo sistema. Caso o mecanismo do banco de dados seja interrompido é possível que existam algumas transações incompletas no momento da interrupção, estas transações deverão ser revertidas pelo mecanismo no momento da volta à atividade. Ainda na interrupção do mecanismo do banco de dados, algumas modificações que foram registradas no arquivo de *log* podem não ter sido descarregadas (*flushed*) para os arquivos de dados, neste caso todas as transações que receberam o registro de *commit* deverão ser reenviadas para os arquivos de dados. Estes procedimentos são necessários para garantir a integridade do banco de dados.

O *SQL SERVER 2000* oferece suporte para transações distribuídas, que são basicamente, transações que envolvem recursos de duas ou mais origens. Para dar suporte a estas transações distribuídas, o produto conta com um gerenciador de recursos e um gerenciador de transações, todo o processo das transações é construído com base no protocolo 2PC. O risco de uma falha no enlace de rede, segundo o fabricante, é tratado pela utilização do protocolo 2PC.

A versão *SQL SERVER 2000 Windows CE* é um banco de dados compacto que estende a capacidade de gerenciamento de dados para os ambientes móveis, segundo o fabricante, esta versão é ideal para aplicações móveis e *wireless*. Algumas das características do *SQL SERVER 2000 Windows CE*:

- Integridade referencial com atualizações e exclusões em cascata
- 32 índices por tabela, índices multi-colunas
- Aninhamento de transações
- Criptografia 128 bits
- Comandos *DDL* e *DML*

O *SQL SERVER CE* oferece suporte a transações, permitindo que o aninhamento das transações aconteça até que atinja cinco níveis. A documentação do produto no entanto, faz algumas considerações sobre as propriedades *ACID* nas transações executadas na plataforma *SQL SERVER CE*:

- Atomicidade: a unidade de trabalho é confirmada ou revertida como uma única unidade
- Consistência: as transações levam o banco de dados de um estado consistente a outro igualmente consistente, no entanto é possível que durante a transação o banco de dados se encontre em um estado inconsistente.
- Isolamento: o trabalho de uma transação não é visível fora do seu escopo enquanto a mesma estiver ativa. O *SQL SERVER CE* suporta apenas o nível de isolamento *read committed*.
- Durabilidade: após a confirmação de uma transação, o trabalho confirmado sobrevive a uma falha de sistema. O *SQL SERVER CE* não garante a durabilidade em todos os casos, segundo a documentação do produto é possível que ocorram perdas de dados no evento de uma falha de sistema.

Os dados podem ser atualizados simultaneamente no dispositivo móvel e no servidor. Caso a unidade móvel esteja fora da área de cobertura do canal *wireless,* o aplicativo no dispositivo móvel pode continuar a ser utilizado, armazenando e

gerenciando os dados dentro do banco de dados local. Após a conexão com a rede estacionária, os dados poderão ser sincronizados seja pelo modelo de replicação *Merge* ou pelo conceito de *RDA (Remote Data Access).*

O *RDA (Remote Data Access)* pode ser utilizado caso o dispositivo esteja total ou parcialmente conectado com a rede estacionária. A aplicação pode submeter comandos *SQL* que serão apenas repassados para o sistema do *SQL SERVER* providenciar a sua execução. Outra possibilidade de utilização do *RDA* é a aplicação submeter uma consulta *SQL* que retorna um conjunto de resultados, o conjunto de resultados é transmitido para o dispositivo móvel onde será armazenado em uma tabela. Todas as alterações feitas pela aplicação serão rastreadas, e por solicitação da aplicação, as linhas atualizadas serão retornadas para o servidor e aplicadas ao banco de dados. Segundo o fabricante, tanto a replicação quanto a arquitetura *RDA* recuperam-se de falhas na transmissão dos dados reiniciando a transmissão a partir do último bloco de dados transmitidos com sucesso.

5.2 DB2 *Everyplace*

O *IBM DB2 Everyplace* é um banco de dados relacional que estende a troca de dados do ambiente empresarial para dispositivos móveis, como por exemplo *PDAs* e telefones inteligentes. Segundo o fabricante, o DB2 Everyplace é constituído de três módulos: o DB2 Everyplace Database o DB2 Everyplace Sync Server e o DB2 Everyplace Mobile Application Builder.

5.2.1 DB2 Everyplace Database

O *DB2 Everyplace Database* é um sistema de banco de dados relacional desenvolvido para utilização em dispositivos móveis. Inclui o aplicativo *Query By Example (QBE)* para utilização em dispositivos *Palm OS*. O DB2 Everyplace Database pode ser utilizado nos seguintes sistemas operacionais: *Palm OS* , *Symbian OS Version 6*, *EPOC Release 5*, *Microsoft Windows CE*, *Win32 (Windows NT and Windows 2000)* , *QNX Neutrino, Linux* e dispositivos com *Linux* embutido.

A seguir serão relacionadas algumas das principais características relativas ao banco de dados relacional oferecido pelo *DB2 Everyplace Database*:

- Acesso aos dados através de *JDBC, C++, ODBC* e *APIs*
- Suporte ao processamento de transações, junções entre tabelas, operações de *Insert* com a utilização de sub consultas, listas *IN,* funções *RTRIM* e *LENGTH.*
- Suporte aos comandos de definição de dados *DDL.*
- Suporte aos comandos de manipulação de dados *DML.*
- Restrições do tipo *CHECK, DEFAULT, PRIMARY KEY* e *FOREIGN KEY.*
- Tipos de dados: *small integer, integer, decimal, char, varchar, Binary Large Object, date, time* e *timestamp.*

5.2.2 DB2 Everyplace Sync Server

Através deste componente é efetuada a sincronização dos dados entre os dispositivos móveis e as fontes de dados. A sincronização pode ser unidirecional ou bidirecional, desta maneira os dados podem ser atualizados tanto no dispositivo móvel quanto no banco de dados central. A operação de sincronização é construída com a utilização de dois componentes que interagem no sentido de gerenciar o processo de sincronia: o *DB2 Everyplace Sync Server* que atua no lado servidor e o *DB2 Everyplace Sync Client* que age no lado cliente. O servidor de sincronização também possui um serviço de verificação e resolução de conflitos.

5.2.3 DB2 Everyplace Mobile Application Builder

Este componente é uma ferramenta de desenvolvimento para aplicações *Palm OS* que utilizam o banco de dados *DB2 Everyplace.* Segundo o fabricante, esta ferramenta possibilita o desenvolvimento de pequenas aplicações sem que o usuário digite qualquer linha de código. O componente é compatível com a plataforma *Palm OS* e o banco de dados *DB2 Everyplace* mantido no dispositivo.

5.2.4 Recuperação de falhas.

O banco de dados relacional *DB2* utiliza-se do conceito de arquivos de *log* para suportar a capacidade de recuperação de falhas. Apresenta como opção *default* o arquivo de *log* do tipo circular. No arquivo de *log* do tipo circular, os arquivos são reutilizados possibilitando apenas que a recuperação aconteça para as falhas de sistema.

A outra possibilidade de arquivo de *log* a ser utilizado pelo *DB2* é o arquivo de *log retain*. Neste formato, os arquivos de *log* não são reutilizados, permitindo então os processos de recuperação do tipo *rollback* e *forward.* Este formato de arquivo permite o gerenciamento tanto das transações confirmadas quanto das pendentes, possibilitando ainda o envio dos registros já efetivados para um arquivo secundário e *off-line*.

A versão *DB2 Everyplace* trabalha com a possibilidade de manter os dados nos dispositivos móveis e posteriormente, através do *Sync Server*, executar a sincronização dos dados atualizados localmente com o banco de dados central. Caso ocorra alguma interrupção durante o processo de sincronização, ele será reiniciado a partir do ponto em que houve a interrupção do sincronismo.

5.3 *Sybase SQL Anywhere Studio*

O *Sybase SQL Anywhere Studio* (SYBASE, 2002) é um conjunto de aplicativos que permitem o gerenciamento e sincronização de ambientes de dados no intuito de propiciar a construção de aplicativos para dispositivos móveis e bancos de dados embutidos. O *Sybase SQL Anywhere Studio* é composto pelos bancos de dados *Sybase Adaptative Server Anywhere* e *Ultralite*, pelas tecnologias de sincronização de dados *SQL Remote* e *Mobilink* e pela ferramenta de desenvolvimento de aplicações *MobileBuilder.*

5.3.1 *Adaptative Server Anywhere*

O *Adaptative Server Anywhere* é um banco de dados desenvolvido para suporte às aplicações móveis ou distribuídas. Oferece suporte para os seguintes sistemas operacionais: *Unix, IBM AIX, HP-UX, Linux, Solaris/SPARC, Novell Netware* e *Windows 95/98/ME/CE/NT/2000/ XP*.

Na relação a seguir, constam as principais características do mecanismo do banco de dados do *Adaptative Server Anywhere*:

- Suporte total a transação
- *Triggers e Stored procedure Java* e *SQL*
- *Stored Procedure* externas (*DLL*)
- Integridade referencial e identidade, incluindo cascateamento das operações de *delete* e *update.*
- Bloqueio ao nível de linha
- Criptografia de arquivos de dados e dos pacotes de transmissão de dados

O *Adaptative Server Anywhere* possui quatro tipos de arquivos necessários para o seu funcionamento: ***database file, transaction log, transaction log mirror*** e o ***temporary file***. O arquivo ***database file*** é o local onde as informações relativas ao banco de dados são armazenadas. Ainda no *database file* são armazenadas as tabelas e índices do sistema, vitais para o funcionamento do mecanismo do banco de dados.

O arquivo ***transaction log*** mantém um registro de todas as alterações que foram efetuadas na base de dados, este procedimento permite a recuperação de dados após a ocorrência de uma falha. O arquivo ***transaction log mirror*** é um arquivo de implementação opcional, este arquivo é uma cópia fiel do *transaction log* original. Já o arquivo ***temporary file*** destina-se apenas a fornecer um local de armazenamento para informações temporárias do mecanismo do banco de dados, é aberto no momento da ativação do banco de dados e fechado quando o mesmo cessa suas atividades.

As informações mantidas nos arquivos *database file, transaction log* e *temporary file* são organizadas sob a forma de espaços de tamanho fixo no disco denominadas de páginas. Existe, porém, dois tipos de páginas destinadas ao processo de recuperação de falhas do banco de dados: ***rollback log*** e ***checkpoint log pages***.

As páginas ***rollback log*** oferecem suporte para o *rollback log*. Para cada conexão com o banco de dados existirá um *rollback log*. Este *log* manterá um registro de todas as alterações enviadas para o banco de dados por uma determinada conexão, a partir do último registro de *commit* ou *rollback.* Este registro permite que as mudanças não confirmadas sejam canceladas caso seja emitido um comando de *rollback* ou na ocorrência de uma falha. Uma vez que a transação tenha sido confirmada ou revertida,

o *rollback log* será apagado e as páginas disponíveis serão oferecidas para o uso do banco de dados. O conceito de *checkpoint* também é utilizado neste produto. Imediatamente após um registro de *checkpoint* as informações mantidas nos *database files* serão iguais às informações mantidas no *transaction log.* Na ocorrência de uma falha, é possível que o servidor não consiga emitir um registro de *checkpoint* antes de um processo de *shutdown*. Neste caso o banco de dados pode retornar a um estado inconsistente, pois algumas transações emitidas a partir do último *checkpoint* podem ter sido efetivadas no disco enquanto que outras podem ter sido perdidas. A solução para este problema vem através da utilização do ***checkpoint log***. O servidor de banco de dados mantém cópias de todas as páginas que foram modificadas a partir do último *checkpoint*, estas páginas são chamadas de *rollback pages* e em conjunto formam o ***checkpoint log***. No processo de recuperação o servidor retorna ao seu último estado consistente através da aplicação das *rollback pages* sobre as páginas correspondentes no *database file*. O próximo passo é a aplicação das alterações efetuadas a partir do último *checkpoint* pela releitura do *transaction log,* na seqüência deste processo todas as transações não confirmadas serão revertidas utilizando os *rollbacks logs*.

5.3.2 *Adaptative Server Anywhere Ultralite*

O *Adaptative Server Anywhere Ultralite* é um produto desenvolvido para prover um mecanismo de banco de dados que possa residir localmente nos dispositivos móveis. Esta tecnologia permite o trabalho da unidade móvel de modo desconectado, trabalhando com o conceito de sincronização para transmitir os dados entre a unidade móvel e a rede estacionária. As plataformas suportadas pela tecnologia são as seguintes: *Palm, Wind River VxWorks, Java e Windows 95/ 98/ Me/ CE/ NT/ 2000/ XP*.

A seguir serão relacionadas algumas características específicas relacionadas à capacidade de armazenamento de informações desta tecnologia:

- Suporte para criptografia dos bancos de dados
- Suporte ao processamento de transações, integridade referencial e operações de *join* entre tabelas
- Suporte à utilização de índices
- Suporte aos dados do tipo *BLOB*

O *Ultralite* utiliza uma tecnologia que permite a sincronização de usuários remotos com os bancos de dados da rede estacionária. Este processo de sincronização permite o acesso aos dados mantidos em servidores de bancos de dados da *Sybase, Microsoft, Oracle* e *IBM*. Na tentativa de reduzir o tempo de sincronização, apenas os itens de dados que foram alterados irão participar do processo de sincronização.

5.3.3 Processos de sincronização

O *Sybase SQL Anywhere Studio* apresenta dois processos de sincronização: *MobiLink* e o *SQL Remote*.

5.3.3.1 Sincronização baseada em mensagens com o *SQL Remote*

Uma instalação típica de sincronização com o *SQL Remote* é composta de um servidor central com uma cópia de dados principal (*master*) e vários bancos de dados remotos em *notebooks* ou *laptops*. O banco de dados central pode estar configurado com uma instalação do *Adaptative Server Anywhere* ou do *Adaptative Server Enterprise*, enquanto que os bancos de dados remotos estarão configurados com uma instalação do *Adaptative Server Anywhere.* Os dados podem ser inseridos tanto no servidor central quanto nos bancos de dados remotos. A sincronização irá acontecer através da troca de mensagens de sistema, por e-mail ou transferência de arquivos.

5.3.3.2 Sincronização baseada em sessão com o uso do *Mobilink*

Através da sincronização baseada em sessão utilizando o *Mobilink* é possível englobar no processo de sincronização pequenos dispositivos que suportam o banco de dados *UltraLite*. Com esta tecnologia é possível utilizar como servidor central de banco de dados várias tecnologias: *Sybase, Microsoft, Oracle* e *IBM*. Vários protocolos de comunicação são suportados neste modelo de sincronização: *TCP/IP, Palm HotSync,* conexão direta através de porta serial e ainda os canais *wireless.*

5.4 *Oracle9i Lite*

O *Oracle9i Lite* é uma plataforma de desenvolvimento voltada para a construção e gerenciamento de aplicações desconectadas móveis. Segundo o fabricante (ORACLE, 2002), a plataforma suporta os seguintes sistemas operacionais: *Palm OS, Symbian EPOC, Microsoft Windows CE*, e *Microsoft Windows 95/98/NT/2000*. Um dos componentes desta plataforma de desenvolvimento é o *Oracle Lite Database,* que é um banco de dados relacional voltado para a utilização em dispositivos móveis. O *Oracle Lite Database* oferece suporte à transações e às restrições de integridade. Outro componente da plataforma *Oracle9i Lite* é o *Mobile Server*, que tem como uma das responsabilidades o gerenciamento de todo o processo de sincronização dos dispositivos móveis com a rede estacionária.

Outro componente igualmente importante da plataforma *Oracle9i Lite* é o *Mobile Development Kit,* este componente oferece aos desenvolvedores a possibilidade escolher entre quatro modelos de aplicações: *Native applications, Java applications, Web-to-Go* e o *Branch Office*.

As ***native applications*** acessam o sistema operacional nativo do dispositivo móvel e provê uma interface *ODBC* para acesso ao *Oracle9i Lite Database.* As *native applications* são normalmente construídas utilizando linguagens como: *C, C++, Visual C++, Visual Basic, ActiveX Data Objects (ADO).*

As ***Java applications*** acessam as unidades móveis através do suporte ao *Java,* o acesso ao banco de dados é efetivado através do *JDBC*. O *Oracle9i Lite Database* também oferece suporte para *Stored Procedures* e *Triggers Java.*

Optando pelo ***Web-to-Go***, o usuário terá a possibilidade de interromper a conexão com a rede estacionária e continuar o trabalho de forma desconectada através do uso do *browser,* buscando as informações necessárias no banco de dados local. Da mesma forma, poderá retornar a conexão com a rede em qualquer momento, sem que este processo interrompa o seu trabalho. Ao solicitar a reconexão com a rede estacionária, terá início um processo de sincronização dos dados da unidade móvel com os dados do servidor central. Já o ***Branch Office*** é uma versão multiusuário do *Web-to-go,* permitindo que alguns usuários se conectem a um cliente *Web-to-go* para fazer uso das capacidades deste modelo. De tempos em tempos acontecerá um mecanismo de sincronização relativo a cada cliente que efetuou esta conexão.

5.4.1 *Oracle9i Lite Database*

O *Oracle9i Lite Database* é um banco de dados relacional construído especialmente para utilização em aplicações móveis. Este banco de dados permite o armazenamento das informações nos dispositivos móveis, abrindo a possibilidade do trabalho desconectado com relação à rede estacionária. A seguir será apresentada uma relação de características de capacidade e propriedades deste mecanismo de banco de dados:

- Acesso a classes *Java*
- Compatibilidade com os servidores de bancos de dados *Oracle*
- Interface *JDBC* nativa
- *ODBC* em todas as plataformas
- Suporte à transação
- *Stored Procedures* e *Triggers SQL* e *Java*
- Integridade referencial

5.4.1.1 Suporte à transação no *Oracle9i Lite*

Quando uma aplicação conecta para um banco de dados *Oracle9i Lite* inicia uma transação com o banco de dados.. Segundo o fabricante, o suporte ao conceito de transação é construído através da observância das seguintes propriedades: atomicidade, consistência, independência e durabilidade.

A propriedade da **atomicidade** é implementada pela não atualização do arquivo de dados até que o banco de dados confirme a transação. Durante a operação de confirmação (*commit*), um *log undo* temporário é criado e então o banco de dados será atualizado. Se ocorrer alguma falha durante este processo de *commit*, o banco de dados será restaurado a partir do *log undo* criado durante a próxima conexão.

A propriedade de **consistência** é implementada impedindo que operações de *commit* aconteçam em transações que violem alguma restrição de integridade da base de dados.

O **isolamento** das transações é obtido por meio dos vários níveis de bloqueio permitidos pelo mecanismo de banco de dados.

A propriedade de **durabilidade** no *Oracle9i Lite* é garantida após a confirmação dos efeitos da transação. Caso ocorra uma falha imediatamente após esta confirmação no banco de dados, os efeitos da transação serão permanentes. Caso ocorra uma falha durante um processo de *commit* ou *rollback* de uma transação, o *log undo* será utilizado para levar o banco de dados a um estado consistente.

5.4.2 *Mobile Sync*

O *Mobile Sync* é um componente residente na unidade móvel que habilita a sincronização dos dados entre as unidades móveis e a rede estacionária. A sincronização acontecerá na troca de dados entre o servidor central e as unidades móveis, o *Mobile Server* tem a função de detectar e resolver possíveis conflitos que possam ocorrer.

5.4.3 *Mobile Server*

O *Mobile Server* provê a sincronização entre os bancos de dados *Oracle Lite* e os servidores *Oracle*. Usando um modelo de publicação/assinatura para a distribuição de dados, o *Mobile Server* habilita dispositivos móveis individuais a assinarem alguma publicação de dados. O *Mobile Server* suporta o protocolo de transporte *http* para a comunicação entre o servidor central e os clientes remotos.

Tabela 3 - Resumo das principais características dos produtos analisados

Produto	**Fabricante**	**Suporte à transações**	**Possibilidade de trabalho desconectado**	**Arquivo de *log***	**Mecanismo de sincronização e detecção de conflitos**	**Suporte à falhas nas transações entre unidades fixas e móveis**	**Sistemas operacionais suportados**
SQL Server CE	*Microsoft*	Sim	Sim	Sim	Sim	Não	*Windows 2000, NT, 98, CE, XP.*

Produto	**Fabricante**	**Suporte à transações**	**Possibilidade de trabalho desconectado**	**Arquivo de *log***	**Mecanismo de sincronização e detecção de conflitos**	**Suporte à falhas nas transações entre unidades fixas e móveis**	**Sistemas operacionais suportados**
Oracle 9i Lite	*Oracle*	Sim	Sim	Sim	Sim	Não	*Windows 2000, NT, 98, CE, XP. Palm,Symbian.*
DB2 Everywhere	*IBM*	Sim	Sim	Sim (*log circular e retain)*	Sim	Não	*Windows 2000, NT, 98, CE, XP. Palm,Symbian, EPOC, Linux*
SQL Anywhere Studio	*Sybase*	Sim	Sim	Sim	Sim	Não	*Windows 2000, NT,98, CE* e *XP.Novell, Solaris, Linux, HP-UX, IBM.*

5.5 Resumo da análise comparativa entre os SGBDs comerciais

A análise comparativa permite a verificação da semelhança entre todos os produtos estudados. As poucas diferenças encontradas foram com relação aos sistemas operacionais suportados. Todos os produtos fazem utilização do arquivo de *log* para o registro das transações, da mesma forma todos oferecem suporte ao conceito das transações em bancos de dados.

O trabalho desconectado é uma característica comum a todos os produtos, igualmente acontece em relação ao mecanismo de sincronização e verificação de conflitos, todos os produtos analisados oferecem a atualização das modificações através de um mecanismo de sincronização.

Nenhum dos produtos apresenta qualquer mecanismo de suporte à ocorrência de falhas nas transações entre as unidades móveis e a rede estacionária. Este fato abre a possibilidade da ocorrência de inconsistências nos bancos de dados sob o ambiente da computação móvel. Os mecanismos de recuperação de falhas destes produtos, resumem-se aos mecanismos implementados para as suas versões estacionárias (computação fixa). Não foi possível identificar a aplicação de nenhuma das propostas estudadas neste trabalho nos produtos analisados, fica claro neste resumo que os fabricantes não englobaram nos seus produtos os estudos mais recentes sobre a recuperação de falhas em ambientes de bancos de dados móveis.

CAPÍTULO VI

CONCLUSÃO

6.1 Resumo do trabalho

O ambiente tradicional de computação móvel consiste de componentes fixos, servidores e estações de suporte, interconectados através de uma rede de comunicação estacionária. Integrante deste ambiente de comunicação surgem as **unidades móveis**, ou **dispositivos móveis**. As unidades móveis são dispositivos dotados de mobilidade, compartilham as informações armazenadas na rede estacionária por meio de um canal de comunicação provido por algumas estações da rede estacionária.

As estações que fornecem o canal de comunicação da rede estacionária com as unidades móveis são identificadas como **estações de suporte**. Cada estação de suporte tem sob sua responsabilidade a cobertura de uma área geográfica com um canal *wireless*. A área de cobertura relativa a cada estação de suporte é denominada de **célula**.

Uma unidade móvel só pode estar sob a coordenação de uma única estação de suporte, no entanto é possível que, devido à mobilidade das unidades, um dispositivo móvel ultrapasse as fronteiras de uma célula de cobertura para outra célula diversa. Neste instante, a unidade estará sob os efeitos das duas estações de suporte, necessitando que se execute um processo de migração das informações de gerenciamento da unidade móvel entre as duas estações de suporte. Este processo é necessário para que o cruzamento entre as fronteiras de cada célula aconteça com o mínimo prejuízo para o usuário da unidade móvel. O procedimento de troca da área de cobertura da unidade móvel é conhecido como ***handoff***.

Os mecanismos de bancos de dados mantidos nas unidades móveis permitem que as consultas e atividades da unidade continuem a acontecer mesmo quando esteja desconectada da rede estacionária. Para que o trabalho desconectado seja possível é necessário que se efetuem cargas de dados da rede estacionária para as unidades móveis. Através do processo de carga local de dados, as unidades móveis podem utilizar os mecanismos de bancos de dados para acessarem os itens de dados necessários. A carga

de dados da rede estacionária para as unidades móveis é identificada na literatura como **difusão de dados**.

A capacidade relativamente pequena de processamento e armazenamento de dados das unidades móveis, o limitado período de vida útil das baterias das unidades móveis e a instabilidade da conexão das unidades com a rede estacionária são alguns dos grandes limitadores das arquiteturas de bancos de dados móveis.

O conceito tradicional de transação necessita ser adaptado para utilização em um ambiente de banco de dados móvel. Diferente dos ambientes tradicionais, este tipo de arquitetura apresenta um novo predicado na construção de consultas: a localização dos dados. Tendo as unidades móveis a capacidade de movimento inerente, as consultas ao serem processadas precisam levar em conta a localização dos dados necessários para apresentarem os resultados solicitados. Tradicionalmente, uma transação leva um banco de dados de um estado consistente a outro estado igualmente consistente. Esta característica das transações só é possível pela observância das propriedades **ACID** (**A**tomicidade, **C**onsistência, **I**ndependência e **D**urabilidade). Em um ambiente de bancos de dados móveis estas propriedades precisam ser relaxadas para que o processamento das consultas aconteça de forma coerente. Algum nível de inconsistência do banco de dados poderá ser permitido em determinado instante de tempo.

A possibilidade de ocorrência de falhas, tanto de *hardware* quanto de *software* é maior em um ambiente de computação móvel. Em um ambiente de banco de dados móvel esta falha deve ser devidamente tratada sob o risco de levar o banco de dados a um estado inconsistente. Algumas propostas de tratamento para recuperação de falhas em bancos de dados móveis surgem na literatura. Basicamente, a utilização de um arquivo que registra as atividades do banco de dados (***log***) e a emissão freqüente de pontos de verificação (***checkpoints***). Neste arquivo de *log* são algumas das premissas destas propostas. A limitação do escopo da falha também é considerada nas propostas. Algumas propostas trabalham apenas com a ocorrência da falha nas unidades móveis, outras abordam a falha nas estações de suporte enquanto que algumas propostas abordam a ocorrência de falhas tanto nas unidades móveis quanto nas estações de suporte. Outro fator igualmente abordado nos protocolos é a possibilidade ou não da previsão da ocorrência da falha, esta possibilidade altera o mecanismo de recuperação da falha.

Além das propostas de algoritmos para a recuperação de falhas em ambientes de bancos de dados móveis, foram abordadas as tecnologias comercialmente existentes na área de gerenciamento de bancos de dados. Os sistemas gerenciadores de bancos de dados apresentados no trabalho foram os produtos dos seguintes fabricantes: *IBM, Microsoft, Oracle* e *Sybase.*

6.2 Conclusões

O ambiente de bancos de dados móveis é um setor em franca expansão, a cada dia surgem novas possibilidades de aplicações deste conceito. A possibilidade de acesso a informações não importando o local ou a conectividade com a rede estacionária abre fronteiras para o desenvolvimento de novas aplicações.

Entretanto, alguns problemas inerentes a esta arquitetura ainda precisam ser solucionados. Entre estes problemas, a recuperação de falhas do ambiente é um dos mais preocupantes, pois pode comprometer a totalidade da aplicação. A possibilidade de tornar um banco de dados inconsistente após a ocorrência de uma falha é um fator determinante do sucesso ou fracasso de uma aplicação móvel.

A análise comparativa entre os protocolos apresentados esclarece o comportamento das propostas com relação à disponibilidade dos bancos de dados em um ambiente de computação móvel. As unidades móveis não são consideradas como locais de armazenamentos estável, desta forma as propostas concentram nas estações de suporte todas as informações necessárias para os processos de recuperação de falhas das unidades móveis. No entanto, apesar do papel crucial destas estações, apenas a proposta de ALAGAR et al (1993) apresenta um mecanismo de recuperação de falhas que contempla a ocorrência de falha nas estações de suporte. Quaisquer das propostas restantes terão o seu funcionamento prejudicado caso ocorra uma falha na unidade de suporte.

Da análise dos mecanismos utilizados pelos aplicativos SGBDs comerciais, ficou evidente a coincidência entre todos os mecanismos, nenhum dos produtos apresenta um mecanismo de recuperação de falha específico para o ambiente da computação móvel. Não foi possível a identificação da aplicação, pelos SGBDs, de nenhum dos protocolos de recuperação de falhas analisados neste trabalho. Os produtos de bancos de dados móveis desenvolvidos pelos fabricantes limitam-se a uma adaptação dos mecanismos

utilizados nas suas versões para ambientes estacionários. Esta adaptação não contempla as possibilidades abordadas pelos mecanismos estudados neste trabalho, limitando desta forma, a capacidade de recuperação de falhas destes produtos.

6.3 Relevância do trabalho

A contribuição deste trabalho para a área de sistemas gerenciadores de bancos de dados móveis é apresentar as necessidades de se adotar um mecanismo de recuperação de falhas coerente com o ambiente da computação móvel. Foram apresentadas algumas propostas encontradas na literatura, e identificadas semelhanças e divergências entre cada proposta. A verificação das propostas de recuperação de falhas nos principais sistemas comerciais gerenciadores de bancos de dados, mostrou que ainda existe a necessidade de oferecer algum mecanismo que garanta a disponibilidade do ambiente nas mais diversas ocorrências de falhas possíveis. Ficou claro que o desenvolvimento de um produto comercial que esteja realmente adaptado para o ambiente de bancos de dados móveis ainda não é uma realidade.

6.4 Perspectivas futuras

Tendo em vista a velocidade de desenvolvimento de aplicações portáveis para a área de bancos de dados móveis, cada vez mais existirá a necessidade de se aumentar a disponibilidade do ambiente. Esta disponibilidade do ambiente passa, obrigatoriamente, pelo tratamento de falhas que possam ocorrer nos mais diversos pontos do processo. As pesquisas nesta área de recuperação de falhas devem, em futuro próximo, traduzir-se em algum mecanismo seguro e abrangente o suficiente, de forma que possa ser utilizado pelos grandes sistemas gerenciadores de bancos de dados.

Um estudo que poderia contribuir para o desenvolvimento desta área seria a verificação do desempenho de cada um dos protocolos estudados. Desta forma, ter-se-ia a relação existente entre disponibilidade oferecida e desempenho alcançada de cada protocolo sugerido. Os resultados destas relações podem indicar a aplicação de cada protocolo para um ambiente de computação móvel específico, verificando as necessidades de desempenho ou de disponibilidade de cada ambiente.

GLOSSÁRIO

API - *Application Programming Interface* – conjunto de rotinas e instruções que podem ser utilizados por um programa aplicativo para a construção ou gerenciamento de alguma solução.

BANDWIDTH Largura de Banda – largura disponível de banda de uma rede, identificando a disponibilidade do meio para transmissão das informações.

BINDING- Associação entre *home address* e *care-of address*.

BROADCAST- Envio de mensagens através da difusão, uma mensagem é enviada para várias unidades ao mesmo instante.

BSC- *Base Station Controller* – Responsável pelo gerenciamento dos recursos do canal sem fio e pela coordenação do processo de *handoff*.

CACHING – Processo de carregamento de uma quantidade de item de dados em uma unidade móvel, permitindo o trabalho desconectado e aumentando a probabilidade de sucesso na busca pelas informações no banco de dados local.

CARE-OF ADDRESS – Endereço associado à unidade móvel a cada ponto de acesso

CDMA – Tecnologia de transmissão de dados que permite que cada estação transmita em todo o espectro de freqüência continuamente, várias transmissões simultâneas são separadas pro meio de uma codificação .

CHECKPOINT - Ponto de Recuperação – Tem a função de armazenar o estado do processo em um *site* conhecido ou em um computador próximo da localização atual da unidade móvel.

FIREWALL - Uma combinação de *hardware* e *software* desenvolvida especialmente para impedir o acesso de usuários não-autorizados a informações do sistema. É utilizado para separar da Internet a rede de uma empresa.

GATEWAY - É o ponto de acesso a uma rede de comunicação ou interligação entre duas redes, inclusive que utilizem protocolos distintos. Equipamento utilizado para estabelecer a conexão e fazer a conversão entre duas redes.

HANDOFF- Processo de gerenciamento da troca de área de cobertura de células, a unidade móvel está ultrapassando as fronteiras de uma determinada célula e sofrendo os efeitos de uma nova célula. As estações de controle de cada célula trocam as informações necessárias para manter a conexão da unidade durante este procedimento.

HLR - *Home Location Register* - É o servidor de localização, onde cada unidade móvel

deverá ser permanentemente registrada.
HOARDING – É um processo que acontece quando os dados utilizados durante uma operação são pré-carregados para a unidade móvel, tornando-se inacessíveis para outros *site*s.

HOME ADDRESS – É o endereço local associado permanentemente à unidade móvel.

HTML - *Hyper Text Markup Language* - Linguagem de desenvolvimento utilizada no ambiente Web.

HUB – Ponto de concentração de conexões de rede. Recebe sinais dos participantes daquela conexão e os transmite indistintamente para todas as conexões ativas naquele momento.

LOCAL LINK ADDRESS- Endereço de ligação local associado à sub-rede.

MSC - *Mobile Switching Center* - Provê a troca de funções, coordena o registro de localização entre outras funções.

PDA - *Personal Digital Assistants* - São portáteis e móveis, como *handhelds* e *palmtops*.

QoS - *Quality of Service* – Garantia de qualidade de processamento que pode ser oferecida ou solicitada por componentes de um sistema.

SQL - *Structured Query Language* **-** Linguagem específica para comunicação com Bancos de Dados Relacionais.

SQLJ - *SQL in Java.*

STORED PROCEDURES - São porções de código SQL que ficam armazenados compiladas no servidor de banco de dados para serem utilizados quando necessário. Podem ser chamadas a qualquer momento, receber parâmetros e retornar valores relativos à sua execução.

SWITCHES - Aparelho que tem a função de conectar diferentes redes. Traduz endereços Ethernet dos nós que residem em cada segmento da rede e permite apenas a passagem do tráfego necessário, gerenciando e otimizando o tráfego na rede.

TCP – *Transmission Control Protocol* - É um protocolo orientado à conexão confiável, permite a entrega, sem erros, de um fluxo de bytes originado a partir de uma máquina da rede.

TDMA - *Time-Division* Multiple Access – Tecnologia de comunicação sem fio, baseia-se na divisão de uma freqüência de rádio em frações de tempo, as quais são usadas em diferentes conexões.

TRIGGERS – Semelhantes às *stored procedures*, porém não possuem a capacidade de receber parâmetros nem podem retornar valores. As *triggers* são "disparadas" pelo banco de dados no momento de ocorrência de um evento determinado. As *triggers* são consideradas reativas, enquanto que as *stored procedures* são pró-ativas.

VLR - *Visitor Location Register* - São os registros de localização de visitante feitos em cada célula quando o usuário se mover para um novo local.

XML - *Extensible Markup Language* – É linguagem uma *meta-markup language* que provê um formato para descrever dados estruturados.

WAP - *Wireless Application Protocol* - Protocolo para aplicações sem fio. Provê um padrão universal para trazer o conteúdo da Internet e serviços agregados para telefones móveis e outros dispositivos sem fio, como telefones celulares, PDAs (*Personal Digital Assistants*), rádios, *pagers*, entre outros.

WIRELESS - Ambiente de comunicação sem fio, constituído por sistemas e equipamentos adaptados para este fim.

WIRELESS LAN - *Local Area Network* - é uma rede local sem fio.

WIRELESS MAN - *Metropolitan Area Network* - é uma rede metropolitana sem fio, situada geralmente dentro de uma cidade.

WIRELESS WAN - *Wide Area Network* – É uma rede de comunicação sem fio que se estende por uma grande área geograficamente.

REFERÊNCIAS BIBLIOGRÁFICAS

ACHARYA, S.; FRANKLIN, M. J.; ZDONIK, S. **Dissemination-based Data delivery Using Broadcast Disks.** IEEE Personal Communications, 2(6), Dezembro, 1995.

ACHARYA, S; FRANKLIN, M.; ZDONIK, S. **Balancing Push and Pull for Data Broadcast.** Proceedings of the ACM Sigmod Conference, 1997.

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

ALAGAR, Sridhar; RAJAGOPALAN, Ramki; VENKATESAN, S. **Toleranting Mobile Support Station Failures.** Proceedings of the First Conference on Fault Tolerant Systems, Madras, India, 1995.

ALONSO, R.; KORTH, H. F. **Database System Issues in Nomadic Computing.** Proceedings of ACM SIGMOD, Junho, 1993, p 388-392.

ALONSO, Rafael; FRANKLIN, M.; ZDONIK, Stanley; ACHARAYA, Swarup. **Broadcast Disks: Data management for Asymmetric Communication Environments.** Proceedings of the ACM SIGMOD Conference, San Jose, CA, Maio 1995.

ARAÚJO, L. V. de; FERREIRA, J. E. **Cache Semântico para Computação Sem Fio Baseado na Abstração de Composição dos Dados.** WorkSIDAM, Workshop de Sistemas de Informação Distribuída de Agentes Móveis. São Paulo, Outubro, 2000, p. 83-89.

BADRINATH, B. R.; ACHARYA, Arup. **Checkpointing distributed applications on mobile computers.** 3rd IEEE International Conference on Parallel and Distributed Information Systems, Outubro, 1994, p.1-4.

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

BARBARÁ, D.; IMIELINSKI, T. N. **Sleepers and Workaholics: Caching Strategies in Mobile Environments.** Proceedings of the ACM SIGMOD Intl. Conference on Management of Data, 1994, p 1-12.

BARBARÁ, D.; MOLINA, G. H. **Replicated Data Management in Mobile Environments: Anything New Under the Sun?** Proceedings of the IFIP Conference on Applications in Parallel and Distributed Computing, Abril, 1994.

BERTINO, E.; PAGANI, E.; ROSSI, G. P. **Fault Tolerance and Recovery in Mobile Computing Systems.** Disponível em http://citeseer.nj.nec.com/update/454451 10/07/2002.

BESTRAVOS, A.; CUNHA, C. **Server-initiated Document Dissemination for the WWW.** IEEE Data Engineering Bulletin, 19(3), Setembro, 1996.

BOBINEAU, C.; PUCHERAL, P.; ABDALLAH, M. **A Unilateral Commit Protocol for Mobile and Disconnected Computing.** Relatório Técnico, PRiSM Laboratory, Universidade de Versailles, França, Março, 2000.

BUKHRES, O.; MORTON, S. **Mobile Computing in Military Ambulatory Care.** 10o. Simpósio IEEE de Sistemas Médicos Baseados em Computadores, 1997.

CHESS, D.; GROSOF, B.; HARRISON, C.; LEVINE, D.; PARRIS, C.; TSUDIK, G. **Itinerant Agents for Mobile Computing.** IEEE Personal Communications, 2(5), Outubro, 1995.

CONTICELLO, Rob; SARMA, Joydeep Sen. **Consistent Checkpointing and Rollback Recovery for a Mobile Computing Environment.** Abril, 1997.

DESPANDE, P. M.; RAMASAMY, K.; SKUKLA, A. ; NAUGTHON, J. F. **Caching Multidimensional Queries using Ckuncks.** Procedente de SIGMOD, 1998, p. 259-270.

DUNHAM, M. H.; HELAL, A . **Mobile Computing and Databases: Anything New ?.** SIGMOD Record, v.24, n. 4, Dezembro, 1995, p. 5-9.

FORMAN, G. H.; ZAHORJAN, J. **The Challenges of Mobile Computing.** IEEE Computer, 27(6): 38-47, Abril, 1994.

FUCHS, W. K.; NEVES, N.; SSU, K. F.; YAO, B. **Adaptive Checkpointing with Storage Management for Mobile Environments.** IEEE Trans. On Realibility, Dezembro, 1999, p. 315-324.

HELAL, A.; HASKELL, B.; CARTER, J.; CARTER, L.; BRICE, R. **Any Time, Anywhere Computing.** Kluwer Academic Publishers, EUA, 1999.

HOUSEL, B. C.; SAMARAS, G.; LINDQUIST, D. B. **WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment.** ACM/Baltzer Mobile Networking and Applications, 1997.

HUANG, Y.; SISTLA, P.; WOLFSON, O. **Data Replication for Mobile Computers.** Proceedings of the 1994 SIGMOD Conference, Maio, 1994, p. 13-24.

HUANG, Y.; WOLFSON, O. **Object Allocation in Distributed databases and Mobile Computing.** Proceedings of the 10th International Conference on Data Engineering, Fevereiro, 1994, p. 20-29.

IBM DB2. http://www.ibm.com, 26/08/2002.

IMIELINSKI, T.; BADRINATH, B. R. **Replication and Mobility.** Proceedings of the 2nd IEEE Workshop on the Management of Replicated Data, Novembro, 1992, p 9-12.

IMIELINSKI, T.; NAVAS, J. C. **GPS Based Addressing and Routing.** Relatório Técnico LCSR-TR262, CS Dept, Rutgers University, Março, 1996.

IMIELINSKI, T.; VISWANATHAN, S.; BADRINATH, B. R. **Energy Efficient Indexing on Air.** Proceedings of the ACM SIGMOD International Conference on Management of Data, 1994, p. 25-36.

ITO, G. C. **Bancos de Dados Móveis: Uma Análise de Soluções Propostas para Gerenciamento de Dados.** Dissertação de Mestrado, UFSC, 2001.

MARTIN, Cris Pedregal; RAMAMRITHAM, Krithi. **Recovery Guarantees in Mobile Systems.** Proceedings of the ACM Int'I Workshop on Data Engineering for Wireless and Mobile Access, Seattle, Washington, Agosto, 1999.

MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução a Computação Móvel.** DCC/IM, COPPE/Sistemas, NCE/UFRJ, 11ª. Escola de Computação, 1998.

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

NEVES, Nuno; FUCHS, W. Kent. **Adaptative recovery for Mobile Environments.** Communications of the ACM, v.1, n.40, Janeiro, 1997, p. 69-74.

ORACLE. http://www.oracle.com, 26/08/2002

OZU, M. Tamer; VALDURIEZ, Patrick. **Principles of distributed databases systems.** New Jersey: Ed. Prentice-Hall, 1999.

PERKINS, C. **Mobile IP: Design Principles and Practices.** Addison-Wesley, 1998.

PITOURA, E.; BHARGAVA, L. B. **Revising Transaction Concepts for Mobile Computing.** 1st IEEE Workshop on Mobile Computing Systems and Applications, Dezembro, 1994, p. 164-168.

PITOURA, E.; SAMARAS, G. **Data management for Mobile Computing,** Kluwer Academic Publishers, 1998.

PITOURA, Evaggelia; BHARGAVA, Bharat. **A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases.** In SIGMOD Record 24(3), Setembro, 1995, p 44-49.

PRADHAN, D. K.; KRISHNA, P. P.; VAIDYA, N. H. **Recovery in Mobile Wireless Environment: Design and Trade-off Analysis.** Proceedings of the 26th IEEE International Symposium on Fault-Tolerance Computing, Sendai, Japão, Junho, 1996, p 16-25.

PRAKASH, Ravi; SINGHAL, Mukesh. **Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems.** IEEE Transactions on Parallel and Distributed Systems, v.7, n.10, Outubro, 1996, p. 1035 – 1048.

RENNHACKKAMP, M. **Tetherless computing liberates end users but complicates the enterprise.** DBMS ON LINE, Setembro, 1997, p. 93. Disponível em http://www.dbmsmag.com/artin304.html#A000034, 26/08/2002.

RUBINSTEIN, M. G.; RESENDE, J. F. **Qualidade de Serviço em Redes 802.11**. 2002.

SAMARAS, G.; PITSILLIDES, A . **Client/Intercept: a Computational Model for Wireless Environments.** Proceedings of the 4 th International Conference on Telecommunications, Melbourne, Australia, Abril, 1997.

SATYANARAYANAN, M. **Mobile Information Access.** IEEE Personal Communications. 3(1), Fevereiro, 1996.

SHEKHAR, S.; LIU, D. **Genesis and Advanced Traveler Information Systems ATIS: Killer Applications for Mobile Computing.** MOBIDATA Workshop, New Jersey, 1994.

SILBERSCHATZ, A.; BREITBART, Y.; KOMONDOOR, R.; RASTOGI, R.; SESHADRI, S. **Update Propagation Protocols for Replicated Databases.** Procedente da ACM SIGMOD, Conferência Internacional de Gerenciamento de Dados, Philadelphia, 1999.

SQL SERVER CE. http://www.microsoft.com, 26/08/2002.

SSU, K.; YAO, B.; FUCHS, K.; NEVES, N. F. **Adaptive Checkpointing with Storage Management for Mobile Environments,** Dezembro, 1998.

SYBASE. http://www.sybase.com, 26/08/2002.

TANEMBAUM, Andrew S., **Redes de Computadores,** Rio de Janeiro. Campus, 1997.

TERRY, D.; DEMERS, A.; PETERSEN, K.; SPREITZER, M.; THEIMER, M.; WELCH, B. **Session Guarantees for Weakly Consistent Replicated Data.** Proceedings of the International Conference on Parallel and Distributed Information Systems, Setembro, 1994, p.140-149.

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

WU, K-L.; YU, P. S.; CHEN, M-S. **Energy-Efficient Caching for Wireless Mobile Computing.** Proceedings of the 12th International Conference on Data Engineering, Fevereiro, 1996.