PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

CENTRO TÉCNICO CIENTÍFICO

DEPARTAMENTO DE INFORMÁTICA

CONSISTÊNCIA DE DADOS EM COMPUTAÇÃO MÓVEL

Disciplina: Computação Móvel

Aluno: José Maria Monteiro

Professor: Markus Endler

Rio de Janeiro, Novembro de 2004

RESUMO

As recentes evoluções ocorridas nos computadores portáteis em conjunto com os avanços nas recentes tecnologias de comunicação sem fio está possibilitando que os usuários de dispositivos móveis mantenham a conexão com a rede enquanto se movimentam livremente, tendo acesso a recursos, serviços e informações compartilhadas. Este paradigma computacional é denominado de computação móvel. A computação móvel possibilita o desenvolvimento de novas e sofisticadas aplicações em banco de dados. Tais aplicações necessitam recuperar dados atuais e consistentes. Entretanto, devido às limitações inerentes aos ambientes móveis, como reduzida largura de banda e freqüentes desconexão dos dispositivos portáteis, torna-se necessário que mudanças no gerenciamento e nos mecanismos de garantia de consistência dos dados sejam implementadas. Este trabalho, investiga, descreve e classifica as principais abordagens para manter a consistência dos dados em ambientes de computação móvel.

Palavras-chave: Consistência de Dados, Controle de Concorrência, Transações, Computação Móvel, Comunicação sem Fio.

CONTEÚDO

1. INTRODUÇÃO

A integração dos computadores portáteis com as recentes tecnologias de comunicação celular, redes de comunicação sem fio e serviços via satélite está possibilitando que os usuários de dispositivos móveis mantenham a conexão com a rede enquanto se movimentam livremente, tendo acesso a recursos, serviços e informações compartilhadas. Este paradigma computacional é denominado de computação móvel. Neste ambiente, os usuários têm acesso a informações e recursos compartilhados independente de onde estejam localizados e de sua mudança de localização (mobilidade). A computação móvel possibilita o desenvolvimento de novas e sofisticadas aplicações em banco de dados. Entretanto, para que estas aplicações possam executar de forma efetiva, torna-se necessário que mudanças no gerenciamento e nos mecanismos de garantia de consistência dos dados sejam implementadas.

Essa necessidade surge das restrições impostas pelos ambientes de comunicação sem fio, tais como: Limitação na largura de banda dos canais de comunicação sem fio, mobilidade e freqüentes desconexões dos dispositivos móveis, mobilidade dos dados e grande número de usuários.

O gerenciamento de dados e o controle de concorrência em ambientes de computação móvel tem sido objeto de uma variedade de pesquisas e produtos comerciais. Novos modelos computacionais, arquiteturas, modelos transacionais, protocolos e algoritmos têm sido propostos. Neste trabalho, fazemos uma revisão bibliográfica das principais abordagens propostas para garantir a consistência dos dados em ambientes de computação móvel e propomos uma taxonomia a fim de classificar e organizar estes trabalhos. Além disso, apresentamos em detalhes como o sistema Bayou [40,41,4243] pode ser utilizado para garantir a consistência dos dados em redes móveis fracamente (parcialmente) conectadas, garantindo alta disponibilidade de dados.

Este trabalho está organizado da seguinte forma: o capítulo 2 discute os ambientes de computação móvel; o capítulo 3 apresenta uma taxonomia para as abordagens de controle da consistência dos dados; o capítulo 4 descreve em detalhes o sistema Bayou; o capítulo 5 conclui esta monografia.

2. AMBIENTES DE COMPUTAÇÃO MÓVEL

2.1 INTRODUÇÃO

Devido aos recentes avanços da computação e das tecnologias de comunicação sem fio podemos afirmar que a informática está entrando em uma nova revolução, tão profunda quanto as que ocorreram com o surgimento dos computadores pessoais e das redes de computadores [50]. A massificação dos dispositivos computacionais portáteis e sua integração com as redes de comunicação sem fio estão possibilitando o acesso à informação em qualquer lugar e a qualquer momento. Este cenário, denominado de computação móvel, está alterando profundamente a maneira como as pessoas trabalham, estudam e usam seu tempo, além de mudar a forma como as empresas oferecem seus produtos e serviços e interagem com seus clientes.

Neste capítulo, descrevemos e analisamos as principais propriedades e características de um ambiente de computação móvel. O capítulo está organizado como descrito a seguir: a seção 2.2 apresenta uma arquitetura referência para um ambiente de computação móvel e a seção 2.3 discute as características, limitações e problemas referentes a estes ambientes.

2.2 Arquitetura Clássica para um Ambiente de Computação Móvel

Os avanços nas tecnologias de computadores portáteis, comunicação celular, redes de comunicação sem fio e serviços via satélite estão proporcionando o surgimento de um novo paradigma em computação, chamado de computação móvel. Neste cenário, os usuários, de posse de dispositivos móveis, têm acesso a informações e recursos compartilhados independentemente de onde estes usuários estejam localizados e de sua mudança de localização (mobilidade) [50].

O termo "móvel" implica na capacidade que um computador possui de mover-se enquanto mantém uma conexão com uma infra-estrutura fixa de comunicação. Assim, de maneira semelhante aos ambientes de computação distribuída, um dispositivo computacional tem acesso a serviços, recursos e dados distribuídos; e, adicionalmente, pode mover-se livremente. Desta forma, a computação móvel amplia o conceito tradicional de computação distribuída. Um modelo abstrato de uma arquitetura para um ambiente de computação móvel é mostrada na figura 2.1. Esta arquitetura consiste de um conjunto de computadores móveis (*MH - Mobile Host*) e uma rede de alta velocidade que interconecta fisicamente computadores fixos (*FH - Fixed Host*) e estações de suporte à mobilidade (*MSS - Mobile Support Station*). Um computador móvel é um dispositivo de computação inteligente que pode movimentar-se livremente ao mesmo tempo em que mantém sua conexão com a rede fixa através de um *link* (conexão) sem fio. As estações de suporte à mobilidade fornecem uma interface sem fio que permite aos computadores móveis interagir com a rede fixa, tendo acesso a informações e recursos compartilhados. Cada *MSS* é responsável por uma determinada região geográfica, chamada de célula, tendo como uma de suas tarefas o endereçamento dos computadores móveis localizados nesta célula. Uma MSS somente pode se comunicar com as unidades móveis que estiverem fisicamente localizada em sua célula. Um computador fixo é um computador de propósito geral, que pode disponibilizar serviços para os computadores móveis, mas que não é capaz de comunicar-se diretamente com estes [50].

Em uma plataforma de computação móvel, podemos nos referir aos computadores móveis como clientes ou usuários, os quais requisitam serviços disponibilizados por servidores (computadores fixos) localizados na rede fixa. Clientes móveis e estações de suporte à mobilidade se comunicam através de canais de comunicação sem fio. Este canal, em geral, consiste de dois *links*; o *uplink*, utilizado para mover dados dos clientes para os as MSS's, e o *downlink*, usado para transportar dados das MSS's para os clientes.

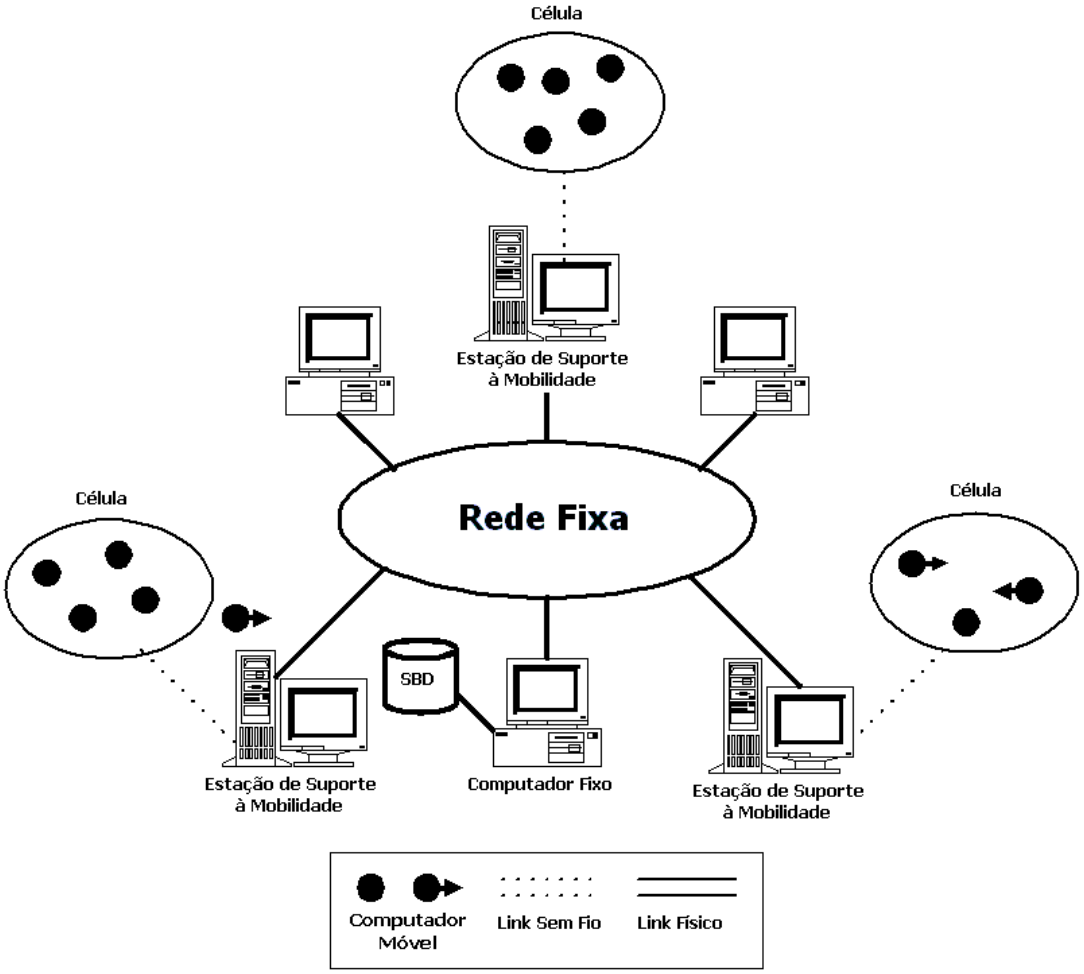

Figura 2.1. Arquitetura Clássica para um Ambiente de Computação Móvel

2.3 Características de um Ambiente de Computação Móvel

Nesta seção iremos apresentar e discutir as principais características de uma plataforma computacional baseada no paradigma de computação móvel.

Como afirmado anteriormente, um ambiente de computação móvel apresenta características bastante particulares. Dentre elas, destacam-se:

(i) Limitação de Energia nos Computadores Móveis

Os computadores móveis dependem de baterias para funcionar. Atualmente, as baterias disponíveis no mercado são relativamente pesadas e só conseguem armazenar energia para algumas horas de uso. Este problema é visto como o maior empecilho no uso de computadores móveis. Infelizmente, a tecnologia de construção de baterias não tem acompanhado o crescimento de outros segmentos da informática e a evolução prevista não muda esse cenário [53]. Logo, o gerenciamento de energia é um problema importante e deve ser tratado tanto pelo hardware quanto pelo software. Portanto, energia é um recurso limitado em computadores móveis, e o seu consumo deve ser minimizado. A fim de conservar energia e estender o tempo de vida da bateria, o cliente entra no modo *doze* (*stand by mode*), no qual o cliente não está ativamente escutando o canal de comunicação com o servidor. Os clientes gastam uma quantidade de energia significativamente menor no modo *doze* que no modo ativo, logo uma das principais metas em computação móvel é minimizar o tempo que o cliente deve gastar no modo ativo para recuperar os itens de dados do seu interesse.

(ii) Longa Desconexão dos Clientes

Um computador móvel pode passar longos períodos de tempo desconectado, devido às limitações de energia da bateria e da própria mobilidade da máquina, pois pode mover-se para uma área não coberta pelo sistema de comunicação.

(iii) Transações de Longa Duração

A eventual e freqüente desconexão dos clientes pode fazer com que as transações móveis que acessam dados dos servidores sejam de longa duração.

(iv) Deve ser Escalonável

O número de clientes nas aplicações em ambientes móveis tende a crescer rapidamente. Desta forma, uma plataforma de computação móvel deve estar apta a suportar um grande número de clientes.

(v) Comunicação Assíncrona entre Clientes e Servidores

Em uma rede de comunicação sem fio o custo para manter um servidor *statefull* chega a ser proibitivo. Por isso, nestes ambientes, os servidores têm uma largura de banda relativamente alta para disseminar informações (*dowlink*), enquanto os clientes não podem transmitir dados ou, se o fazem, é sobre um *link* com baixa largura de banda (*uplink*).

3. UMA TAXONOMIA PARA AS ABORDAGENS DE CONSISTÊNCIA DE DADOS EM COMPUTAÇÃO MÓVEL

3.1 INTRODUÇÃO

Neste capítulo, investigamos e descrevemos as diversas abordagens que têm sido propostas para garantir a consistência dos dados em ambientes de computação móvel. Além disso, propomos uma taxonomia a fim classificar e organizar as soluções encontradas. A taxonomia proposta difere da classificação apresentada por Dunham et al. [26] no sentido em que tenta levar em consideração a arquitetura e o funcionamento das soluções estudadas.

3.2 CLASSIFICAÇÃO

Um sistema de bancos de dados móvel é um ambiente computacional que pode ser visto como a extensão de um sistema distribuído. Özsu e Valduriez propuseram uma excelente classificação para sistemas de bancos de dados distribuídos. Esta taxonomia baseia-se nas características de autonomia, distribuição, e heterogeneidade. Dunham et al. [26] estendem esta classificação adicionando um ponto no eixo da distribuição. Isto deve-se ao fato de que um sistema de computação móvel pode ser visualizado como um sistema distribuído dinâmico, onde os canais de comunicação entre os computadores da rede mudam dinamicamente.

Nesta seção, propomos uma taxonomia para os sistemas de bancos de dados móveis que baseia-se na arquitetura e no funcionamento destes ambientes.

3.2.1. Replicação/Caching de Dados no Cliente

Devido à freqüente desconexão das unidades móveis, causada pelas limitações das baterias e pela própria mobilidade do dispositivo, a comunicação entre os clientes móveis e o servidor fixo se torna bastante instável. Além disso, o custo de se utilizar uma infra-estrutura de comunicação sem fio pode ser bastante elevado, uma vez que a velocidade de transmissão nestes *links* é baixa e a taxação pode ser feita com base no tempo de conexão. Portanto, há um incentivo sensato para que os computadores móveis estejam desconectados por longos períodos de tempo. Desta maneira, torna-se necessária uma forma de possibilitar aos usuários de dispositivos portáteis continuarem a execução de suas operações (leitura e escrita) sobre os itens do banco de dados, mesmo na ausência de uma conexão com o restante do sistema de comunicação, em particular, com o servidor de banco de dados. Chamamos esta característica de suporte à desconexão (A figura 3.1 mostra uma possível arquitetura para este ambiente). A fim de se alcançar este objetivo, uma cópia dos dados (ou de parte deles) é armazenada localmente nos clientes. Em caso de desconexão, as transações locais irão executar suas operações de leitura e escrita sobre sua cópia local, o que proporciona uma maior disponibilidade dos dados e um melhor desempenho na execução das transações. Entretanto, um mecanismo adicional para manter a consistência entre os dados armazenados em cada cliente e no servidor central torna-se necessário.

Duas abordagens principais para manter a consistência de dados, nos ambientes onde o cliente armazena localmente uma cópia dos dados, podem ser encontradas na literatura. A primeira abordagem, trata a cópia local dos clientes como sendo uma réplica total ou parcial do banco de dados central (armazenado na rede fixa) [2,3,4,6,7,8,9,11,12,14,19,24,39]. Já a segunda, considera a cópia local como um *cache* [1,13,15,16,17,20,21,22,23,38].

3.2.1.1. Replicação e Reconciliação

A fim de possibilitar que os usuários de dispositivos portáteis executem operações (leitura e escrita) sobre os itens do banco de dados através de um ambiente com fraca conectividade, o mecanismo de replicação otimista (*lazy replication*) é adotado com bastante freqüência. Nesta estratégia, os dispositivos portáteis podem armazenar localmente uma réplica de um subconjunto dos itens do banco de dados. Quando o dispositivo estiver desconectado do sistema de comunicação, as transações locais (móveis) irão executar suas operações sobre os dados replicados. Tão logo o dispositivo esteja re-conectado, as atualizações realizadas pelas transações locais devem ser propagadas para o servidor de banco de dados (e para os demais clientes que contêm réplicas). Entretanto, uma vez que estas operações podem conflitar com operações executadas no servidor, durante o período de desconexão do cliente, torna-se necessária uma forma de detectar estes conflitos e solucionar os impasses. Esta atividade é chamada de reconciliação. Caso, durante a execução deste processo, um conflito seja detectado e não possa ser solucionado, então, a transação móvel deve ser abortada (cancelada). A replicação otimista aumenta a eficiência e reduz o tempo de resposta das transações móveis, uma vez que utiliza sua cópia local, quando o dispositivo portátil não estiver conectado ao sistema. Por outro lado, caso ocorra uma grande quantidade de conflitos, esta estratégia pode resultar em um número inaceitável de cancelamentos de transações, e conseqüentemente, na insatisfação do usuário. Desta forma, a reconciliação apresenta-se como um problema fundamental em bancos de dados móveis. Devido à essa importância muitas abordagens têm sido propostas. A seguir, discutiremos algumas delas.

Barbará e Molina, em [3], estudam como os mecanismos tradicionais para gerência de dados replicados em redes fixas podem ser adaptados para ambientes móveis.

Em [6], Huang, dentre outros, apresentam e analisam vários algoritmos para alocação de dados, os quais podem armazenar réplicas nos clientes, com o objetivo de otimizar o custo da comunicação entre os dispositivos móveis e os computadores fixos que possuem bancos de dados.

Gray et al. [2] propõem um algoritmo de replicação em dois níveis, o qual permite que as aplicações nos clientes móveis submetam um tipo especial de transação denominada transações de tentativa (*tentative transactions*). Essas transações são executadas sobre as réplicas armazenadas nos clientes quando estes estão desconectados. Quando o cliente móvel se re-conecta, as atualizações das transações de tentativa são re-processadas como transações convencionais no nó que armazena a cópia principal (*master*) dos dados. As transações de tentativa podem falhar durante o re-processamento (devido a conflitos com outras transações). Caso isso aconteça, o cliente móvel que originou a transação é informado sobre a falha e sobre o motivo que causou o erro, e em geral, a transação é abortada.

Pitoura e Bhargava [11] propõem um esquema de replicação que suporta conectividade fraca e desconexão dos dispositivos móveis fazendo um balanceamento entre a disponibilidade de dados e as garantias de consistência.

Em [12], Hara, dentre outros, propõem um algoritmo para escolher o método mais efetivo, dentre duas abordagens: *token* e otimista, para tratar as atualizações concorrentes feitas nas réplicas durante os períodos de desconexão dependendo da probabilidade de que as transações ocorram e do tempo de desconexão. Esta abordagem considera que o ambiente é composto por computadores fixos e móveis, sem fazer distinção entre eles. Além disso, cada *host* possui uma réplica do banco de dados e as conexões podem ocorrer entre pares de *hosts* (como por exemplo, em uma rede *ad hoc*).

Lin, dentre outros, [14], propõem uma abordagem baseada em probabilidade para o processamento de transações em ambientes de computação móvel. Ela utiliza a especificação da qualidade de serviço e alcança a serializabilidade das transações e a consistência dos dados de forma dinâmica. A idéia básica é decidir a melhor ordem para execução das transações e o melhor momento para a reconciliação baseado na probabilidade de conflito.

Em [8], Yee, dentre outros, estendem um trabalho anterior, o qual trata a atualização de réplicas baseado em grupos de clientes, e incluem um modelo de custos detalhado para a atualização de réplicas e um algoritmo guloso baseado em heurística para a determinação dos grupos de clientes.

3.2.1.2. CACHING DE DADOS

Uma vez que a largura de banda de um canal de comunicação sem fio é um recurso limitado, cada dispositivo móvel deve minimizar a utilização deste meio a fim de reduzir a contenção no uso do canal. Armazenar os dados freqüentemente acessados em um *cache* nos computadores móveis tem se mostrado uma técnica bastante útil para reduzir a contenção no meio de comunicação sem fio. Além disso, os mecanismos de *cache* possibilitam que os clientes continuem a executar suas operações de leitura (ou escrita) sobre os dados, mesmo se estiverem desconectados do restante do sistema de comunicação.

A estratégia de usar *cache* pode trazer ganhos de performance de duas maneiras. Primeiramente, a utilização de *cache* pode eliminar a necessidade do cliente enviar múltiplas requisições para acessar o mesmo dado. O segundo aspecto que pode melhorar a performance é a redução de trabalho do servidor remoto.

Entretanto, a utilização de *cache* requer alguns cuidados. Como os itens armazenados no *cache* dos clientes estão sendo constantemente atualizados no servidor central, estes novos valores devem ser propagados para atualizar as cópias nos clientes. Chamamos este mecanismo de consistência ou coerência de *cache*. Dizemos que o valor de uma cópia (em *cache*) de um determinado item é consistente se este valor for igual ao valor do item no servidor central [36].

Nos tradicionais SGBD's cliente/servidor, é relativamente simples manter a consistência das cópias armazenadas no *cache* de um cliente através da utilização de uma combinação de bloqueios e mensagens de invalidação. Os algoritmos existentes para estes arquiteturas, onde a localização e a conexão dos clientes não mudam, podem ser divididos em duas categorias:

Avoidance-based Approach: O servidor envia mensagens de invalidação diretamente aos clientes que possuem cópias a serem atualizadas.

Detection-based Approach: Os clientes enviam consultas ao servidor para validar os dados armazenados em *cache*.

Infelizmente, em computação móvel o problema de coerência de c*ache* não é tão simples. Uma vez que os dispositivos móveis podem se desconectar da rede fixa com certa freqüência e por um prolongado período de tempo, a primeira abordagem (Avoidance-based Approach) não funciona apropriadamente. Por outro lado, devido à limitada largura de banda do canal de comunicação sem fio, os clientes móveis têm uma baixa capacidade de enviar mensagens, o que inviabiliza a utilização da segunda abordagem (Detection-based Approach). Assim, surgiu a necessidade de mecanismos de consistência de *cache* adequados para bancos de dados móveis.

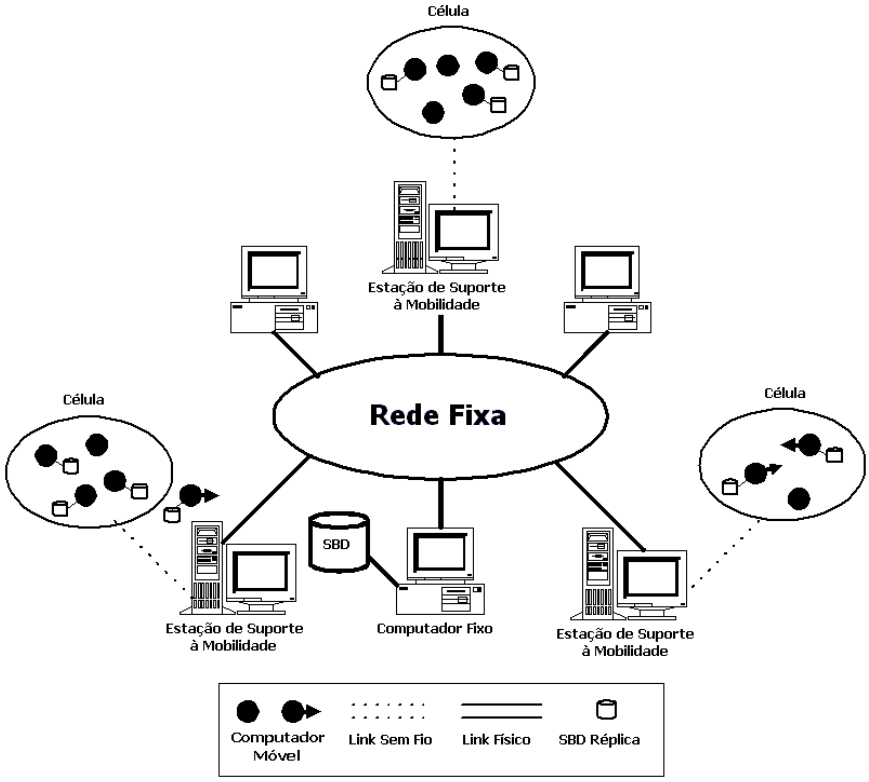

Figura 3.1. Arquitetura para um Ambiente com Replicação/Caching de Dados no Cliente

Walborn e Chrysanthis, em [1], examinam se a semântica dos objetos pode ser explorada para facilitar a operação em modo desconectado.

Jing e Elmargarmid [17] propõem um algoritmo de invalidação de *cache* adaptativo, o qual ajusta o tamanho do relatório a fim de otimizar o uso do *link* sem fio.

Barbará e Imielínski, em [15], estabeleceram uma taxonomia para as estratégias de gerenciamento de cache e através de simulações verificaram que para sistemas onde os clientes ficam a maior parte do tempo desconectados a melhor estratégia de invalidação de cache é o mecanismo baseado em assinatura prévia, já para os sistemas onde os dispositivos móveis permanecem conectados a maior parte do tempo, a melhor estratégia é o relatório de invalidação enviado por broadcast, o qual indica que itens foram atualizados.

Em [21], Lee et al. propõem um protocolo chamado OCC-UTS (*Optimistic Concurrency Control with Update TimeStamp*), a fim de garantir a consistência de *cache* em ambientes de computação móvel através da utilização de um mecanismo para invalidação de *cache* baseado em *broadcast*. Nesta abordagem, a verificação da consistência no acesso aos dados e o protocolo de *commit* são eficientemente implementados de forma completamente distribuída como parte integrante do processo de invalidação de *cache*.

Cao [13] propõe um mecanismo de gerenciamento de *cache* adaptativo baseado no conceito de "*prefetch access ratio*", o esquema proposto pode dinamicamente otimizar a performance ou o consumo de energia, de acordo com os recursos disponíveis e os requisitos de performance.

Hou et al. [38] discutem a construção de relatórios de invalidação com a finalidade de alcançar taxas mínimas de invalidações falsas. Os autores observaram que as taxas de invalidação falsas estão intimamente ligadas com o padrão de re-conexão dos clientes, ou seja, da distribuição da duração de tempo entre a desconexão e a re-conexão. Utilizando o método de Newton, os autores mostram que um relatório com uma taxa de invalidação falsa mínima pode ser construído para um padrão de desconexão qualquer.

3.2.2. SERVIDORES REPLICADOS

Um banco de dados replicado é um banco de dados distribuído no qual múltiplas cópias de parte dos itens de dados são armazenadas em diversos servidores (figura 3.2). A replicação de dados traz as seguintes vantagens:

1. **Melhor Disponibilidade de Dados**: Os dados continuam disponíveis, ou seja, as aplicações podem continuar sua execução, mesmo que alguns dos servidores falhem.

2. **Ganhos de Performance**: Uma vez que existem várias cópias de cada item de dado, as aplicações podem usar as cópias que estiverem fisicamente mais próximas, permitindo um acesso mais rápido aos dados e reduzindo o tráfego no meio de comunicação.

3. **Maior Throughput**: Como existem várias cópias de cada item de dado em diferentes locais, a contenção no acesso aos itens de dados tende a ser menor, o grau de concorrência tende a ser maior e assim o número de transações concluídas por unidade de tempo será maior.

Entretanto, estes benefícios trazem a necessidade de atualizar todas as cópias de um item quando este for alterado. Assim, as leituras tendem a ser mais rápidas e as escritas mais lentas. Além disso, SGBD deve garantir que a execução concorrente de um conjunto de transações sobre um banco de dados replicado será equivalente a uma execução serial sobre o mesmo banco de dados não replicado (*one-copy database*), o que é denominado de *one-copy serializability* (1SR) pela literatura. Nos tradicionais SGBD's replicados isto é realizado através de um protocolo de controle de réplica (*replica control protocol*), os quais podem ser classificados em três grupos:

1. **Primary-Copy Method:** Este método assume que todo item de dado replicado tem uma ordem pré-definida, ou seja, cada cópia de um determinado item tem um número de seqüência. A cópia diretamente mantida pelo seu proprietário é chamada de *primary copy*, e ocupa o primeiro lugar na seqüência. As demais réplicas são chamadas de cópias secundárias. As operações de escrita são sempre executadas inicialmente sobre a cópia primária (*primary copy*) e depois propagada para as demais réplicas de forma síncrona ou assíncrona. Já uma operação de leitura pode ser executada sobre qualquer cópia. Por isso, este método é chamado de *Read-All-Write-One*. Se uma transação lê varias cópias e depois descobre que elas foram atualizadas e que o processo de propagação estava em andamento, ou seja, que o valor lido estava desatualizado, então ela deve ser abortada. Em caso de falha no nó que mantém uma cópia primária de um determinado item, o próximo nó na seqüência (do item) deve assumir o comportamento de cópia principal. Este mecanismo é vulnerável a falhas de comunicação e proporciona um *overhead* na propagação das atualizações.

2. **Quorum-Consensus Method:** Neste método uma operação só será executada se obter um determinado número de votos de permissão (*quorum*) de um grupo de servidores. Em geral, são definidos um *quorum* para leitura e outro para escrita. Quando uma inconsistência é detectada durante uma votação, o processo de reconciliação é iniciado com a finalidade de sincronizar os valores das cópias (dos servidores envolvidos na votação). A transação que falhou durante a votação é geralmente abortada. Este método apresenta problemas de performance uma vez que tem grande *overhead* de comunicação entre os nós.

3. **Available-Copies Method:** Este método também é conhecido como replicação otimista (*optimistic replication*). Durante uma operação de escrita todas as cópias acessíveis são atualizadas. As leituras podem ser executadas sobre qualquer réplica disponível. As transações podem ser executadas mesmo que algumas copas (servidores) estejam inacessíveis. Porém, quando estes servidores se recuperarem, ou seja, se tornarem novamente acessíveis, as inconsistências devem ser detectadas e algumas operações de convergência devem ser executadas a fim de sincronizar os valores dos dados (cópias). Transações que acessaram dados inconsistentes devem ser abortadas. Neste método, as operações de leitura e escrita podem ser executadas com grande performance. Entretanto, os testes para validação e reconciliação requerem elevado tráfego de mensagens. Quando o período de tempo em que um servidor fica inacessível cresce, a eficiência do método é comprometida devido ao grande número de operações de convergência e ao intenso tráfego de mensagens gerado no processo de reconciliação.

Infelizmente, estes algoritmos não são adequados para ambientes de computação móvel, onde tipicamente há uma grande limitação para a comunicação entre os dispositivos móveis e a rede fixa, além da freqüente e muitas vezes prolongada desconexão destes dispositivos. Por este motivo, várias abordagens têm sido propostas. A seguir, discutiremos algumas delas.

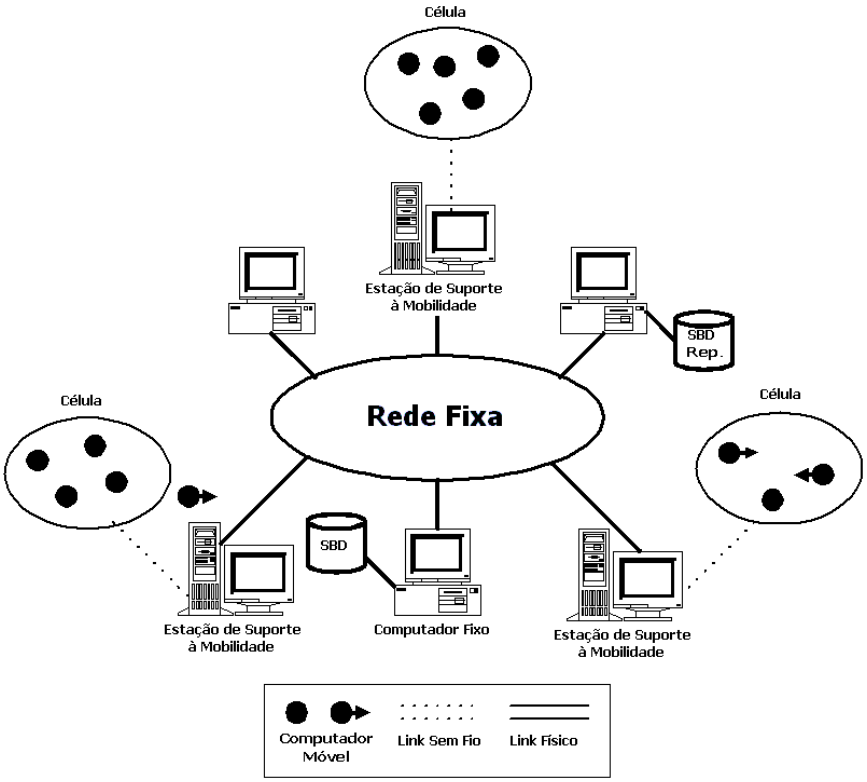

Figura 3.2.a. Uma possível arquitetura de CM com servidores replicados.

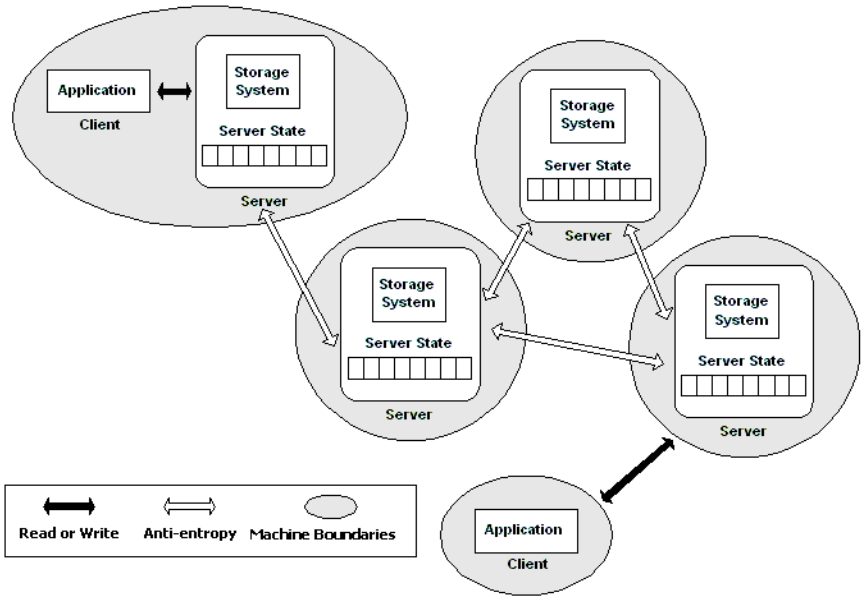

Figura 3.2.b. Uma possível arquitetura de CM com servidores replicados em ambientes parcialmente conectados (fraca conectividade).

3.2.2.1. O SISTEMA BAYOU

O sistema *Bayou* [40,41,42,43] é um sistema de armazenamento que baseia-se na replicação de dados e em um nível fraco de consistência, tendo sido projetado para ambientes de computação móvel que incluam dispositivos portáteis fracamente conectados, ou seja, que apresentam conexão intermitente. Este sistema proporciona uma infra-estrutura para o desenvolvimento de uma variedade de aplicações colaborativas e que não sejam de tempo real, tais como o compartilhamento de calendários, e-mails, edição de documentos para grupos de usuários desconectados. O capítulo 4 descreve com detalhes a arquitetura e o funcionamento do sistema *Bayou*.

3.2.2.2. VOTAÇÃO PONDERADA

Rodrig e Lamarca [44] apresentam um mecanismo para o gerenciamento de dados replicados, o qual consiste em uma variação do clássico esquema de votação ponderada de David Gifford's. As principais características desta abordagem são a descentralização, a utilização de um modelo de consistência já consolidado e o suporte a re-configuração das réplicas. Essa propriedades são interessantes para aplicações no domínio da computação pervasiva (*pervasive computing*). Através da distribuição de meta-dados baseados em versões junto aos dados replicados, e gerenciando tanto os dados quanto os meta-dados com o mesmo quorum, o mecanismo pode ser utilizado em ambientes *peer-to-peer* com membros altamente dinâmicos. O mecanismo proposto provê o acesso aos dados de forma consistente seqüencialmente em ambientes computacionais parcialmente conectados (onde a desconexão é a regra). Os principais objetivos desta abordagem são a necessidade de alta disponibilidade dos dados, de garantias de consistência forte e suporte à desconexão.

3.2.2.3. O SISTEMA DENO

Cetintemel et al. [45] apresentam um esquema, chamado Deno, para gerência de dados replicados que utiliza uma abordagem assíncrona, otimista e descentralizada, sendo voltado especificamente para ambientes móveis fracamente (parcialmente) conectados. Deno utiliza um mecanismo de controle de concorrência baseado em votação ponderada em um sistema de gerenciamento de dados *peer-to-peer*. O propagação das atualizações entre os servidores é realizada por um algoritmo epidêmico, baseado em pares. Deno proporciona um modelo de consistência fraca no qual as transações de atualização são serializadas e as consultas sempre acessam valores consistentes, entretanto não há garantias de que as transações de atualização (com escritas) sejam serializadas com as transações de leituras. A consistência fraca aumenta a disponibilidade dos dados e alcança uma taxa de *commit* mais alta, em ambientes altamente dinâmicos. Na arquitetura proposta, dois ou mais clientes podem se conectar a um determinado servidor. Os servidores se comunicam através da troca de informações entre pares. Não é necessário que os servidores estejam sempre conectados. Além disso, nenhum servidor precisa ter conhecimento de todos os membros do grupo (de servidores), além disso, um determinado servidor somente precisa estar em contato intermitente com um outro servidor, para que possa participar do processo de votação e *commit*. Os votos são disseminados entre os servidores de forma assíncrona através de uma propagação baseada em pares. Não existe um servidor principal que seja o proprietário dos itens de dados, ou responsável por serializar as atualizações.

3.2.3. Replicação em Redes Ad Hoc

Os recentes avanços ocorridos na computação e nas tecnologias de comunicação sem fio tem levado a um grande interesse pelas redes *ad hoc*, as quais são compostas exclusivamente por dispositivos móveis. Em uma rede *ad hoc*, cada dispositivo móvel funciona como um roteador, e se interliga com outros computadores móveis. Assim, a comunicação entre dois *hosts* é possível mesmo que o *host* origem e o destino não estejam na área de cobertura um do outro, neste caso, os pacotes de dados são encaminhados através de outros *hosts* que estejam no caminho até que cheguem ao destino.

Em redes *ad hoc*, uma vez que os *hosts* se movem livremente e que as desconexões ocorrem com freqüência, o que causa a freqüente divisão (partição) da rede, as diferenças tecnológicas com as redes fixas e redes móveis estruturadas são grandes.

Quando acontece a divisão da rede, o que ocorre com freqüência em redes *ad hoc*, um dispositivo que está em uma das partições não consegue acessar os itens de dados mantidos por dispositivos que estejam em outra partição. Assim, a disponibilidade de dados é bem menor que em redes móveis infra-estruturadas. Com a finalidade de reduzir a deteriorização da disponibilidade de dados uma das estratégias possíveis é a replicação de dados. Porém, os métodos tradicionais são impraticáveis devido ao alto dinamismo das redes *ad hoc*. Além disso, outra diferença fundamental é que em redes *ad hoc*, mesmo que um cliente móvel não possa acessar um servidor (provedor) de dados diretamente, (ou seja, mesmo que não esteja na mesma área de cobertura do servidor), ele poderá fazê-lo através de *hosts* intermediários.

Desta forma, novos mecanismos para gerência de dados replicados foram propostos especificamente para redes *ad hoc*, tentando tirar proveito e se adequar às suas características particulares.

3.2.3.1. MÉTODOS PARA ALOCAÇÃO DE RÉPLICAS

A replicação de dados é muito eficiente para prover (aumentar) a disponibilidade de dados. Porém, como os dispositivos móveis geralmente possuem recursos limitados, é impossível manter uma réplica de todos os itens. Em [34], Hara propõe três métodos para alocação de réplicas nos dispositivos móveis, com a finalidade de aumentar a disponibilidade dos dados. Os métodos propostos levam em consideração a freqüência com que os *hosts* móveis acessam cada item de dados, o *status* da conexão de rede e o tempo entre duas atualizações consecutivas para cada item de dado.

3.2.3.2. O MÉTODO PAN

Em [37], Luo et al. propõem um mecanismo que consiste em uma coleção de protocolos para o armazenamento e gerenciamento confiável de dados replicados, acessados concorrentemente (leitura e escrita), em redes *ad hoc*, que garanta alta disponibilidade de dados. A abordagem proposta comporta-se de uma maneira previsível devido a um mecanismo *multicast* de difusão baseado em boatos aplicado a um quorum de acesso. O *overhead* é diminuído adotando-se a construção assimétrica do quorum.

3.2.4. SBD's Múltiplos em CM

Durante muitos anos a utilização dos bancos de dados centralizados foi a abordagem dominante para o gerenciamento dos dados corporativos. Entretanto, as freqüentes fusões e aquisições de diferentes companhias tem levado as organizações à uma nova realidade, trazendo a necessidade de gerenciar uma variedade de sistemas de bancos de dados, os quais são muitas vezes heterogêneos, autônomos e distribuídos geograficamente. Com a finalidade de tornar estes diferentes bancos de dados inter-operáveis a abordagem de bancos de dados múltiplos tem sido exaustivamente estudada. Um banco de dados múltiplo consiste em um coleção de bancos de dados pré-existentes e autônomos, os quais são chamados de bancos de dados locais (*local databases – LDBs*). Os sistemas concebidos para gerenciar os bancos de dados múltiplos são denominados sistemas de bancos de dados múltiplos (*Multidatabase Systems – MDBSs*).

Com o rápido avanço nas tecnologias de comunicação sem fio surge a necessidade de estender os serviços dos MDBSs a usuários de dispositivos portáteis (móveis). Este novo cenário é denominado de sistemas de bancos de dados múltiplos móveis (*Mobile Multidatabase Systems – MMDBSs*). Uma possível arquitetura é mostrada na figura 3.3.

O processamento de transações em sistemas de bancos de dados múltiplos tem sido foco de diversas pesquisas [46] e muitos protocolos para controle de concorrência foram propostos. Entretanto, estas técnicas não são adequadas para sistemas de bancos de dados múltiplos com suporte a usuários móveis. O conceito de mobilidade, onde os usuários acessam os dados através de uma conexão sem fio, introduz complexidades e restrições adicionais, as quais incluem: 1) redução da capacidade de comunicação, devido à limitada largura de banda dos canais (*links*) sem fio, 2) recursos limitados para processamento e armazenamento e 3) freqüente desconexão dos dispositivos portáteis. Assim, novas abordagens têm sido propostas na tentativa de solucionar este relevante problema.

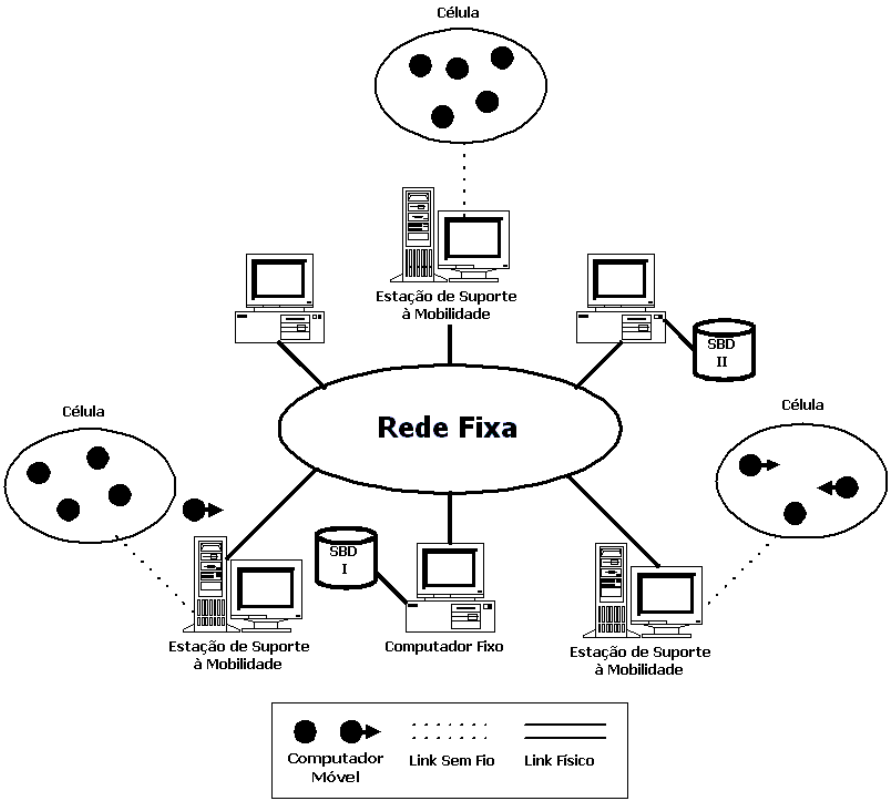

Figura 3.3. Uma possível arquitetura para um MMDBS

3.2.4.1. KANGAROO

O Kangaroo [26] é um modelo de transações que enfatiza o aspecto da confiabilidade que deve possuir um MMDBS em lidar com a mobilidade física dos dispositivos portáteis durante o processamento de transações móveis. Ele é um modelo concebido para MMDBSs (múltiplos e heterogêneos) que baseia-se nas transações dos tipos *global/open-nested* e *split*. A execução de uma transação móvel é descrita a seguir.

Quando um dispositivo móvel requisita a execução de uma transação, a estação de suporte à mobilidade (ESM) correspondente cria uma transação global chamada *kangaroo transaction* (KT) e uma sub-transação chamada *joey transaction* (JT), que é a unidade de execução em cada ESM participante da transação global. Cada JT cria transações locais para acesso aos dados gerenciados pela ESM local, e sub-transações, para acesso aos dados gerenciados por outras ESMs ou servidores fixos. Havendo movimento do dispositivo móvel para outra célula (migração ou *hand-off*), uma nova JT é criada na ESM que estiver cobrindo a nova célula. Após a conclusão das transações locais e globais já iniciadas na JT anterior, serão criadas transações locais e sub-transações, visando executar o restante do código da KT ainda não executado. A atividade de gerenciamento da KT em execução acompanha a migração dos dispositivos móveis.

O *Kangaroo* não garante a execução correta de transações globais concorrentes, de forma que poderão surgir inconsistências nos bancos de dados, causadas por históricos (*schedules*) de execução não serializáveis, embora a atomicidade global seja garantida através de transações de compensação. A origem, o processamento, o gerenciamento de transações móveis e o aceso aos dados são tarefas que dependem das ESMs, o que reduz a autonomia dos computadores móveis. O transporte do contexto de execução das transações de acordo com a migração dos *hosts* móveis pode gerar sobrecarga na rede, pois o restante da transação ainda não executado deve ser totalmente enviado a uma nova ESM, a cada mudança de célula. A fragmentação e a dispersão de informações sobre as transações em execução, através de várias ESMs, podem tornar o processo de recuperação mais complexo.

3.2.4.2. PSTM

O PSTM (*pre-serialization transaction management*) [25] é um modelo de gerenciamento global de transações que visa: i) garantir a correta e eficiente execução de transações móveis através da serialização global de transações e da confirmação independente de sub-transações compensáveis; ii) evitar *aborts* desnecessários provocados pela perda provisória de conexão dos dispositivos móveis com a rede de computação móvel. O modelo compõe-se de dois módulos: o STM (*site transaction manager*), situado em servidores de dados fixos e o GTC (*global transaction coordinator*), situado nas ESMs. Descreve-se, a seguir, o processamento de uma transação móvel.

Um nó qualquer da rede, seja fixo ou móvel, requisita o início de uma transação móvel. A efetiva criação da transação e seu gerenciamento são realizadas pelo GTC. Este gera sub-transações e as submete, juntamente com as transações de compensação, aos SBDs referenciados, com intermediação dos STMs correspondentes. Ao final da execução da fase vital (fase que contém operações que podem gerar conflitos) de uma transação móvel, o GTC verifica se houve violação das propriedades de atomicidade e isolamento, através da construção de um grafo de serialização global, a partir de informações propagadas pelos STMs participantes da transação. Se tiver havido, a transação será abortada. Caso contrário, a ordem de serialização será estabelecida, antes mesmo do término da transação. Ao final da execução de cada sub-transação, o STM respectivo envia uma operação de término para o SBD local. Em caso de o dispositivo móvel que originou a transação perder a conexão com a rede, o GTC suspende a execução temporariamente, aguardando pela retomada da transação, abortando-a somente se for estritamente necessário. A atividade de gerenciamento das transações móveis migra para novas ESMs, de acordo com o movimento dos *hosts* móveis.

O PTSM apresenta como desvantagens: presença de uma camada extra de software (STM) nos dispositivos móveis destinos, o que pode levar a uma perda de desempenho global; violação da autonomia dos SBDs locais, pois os STMs forçam conflitos no sentido de garantirem a serialização global.

3.2.4.3. V-LOCK

O modelo descrito em [31] foi projetado para integrar-se ao SSM (modelo sumário de esquemas), o qual visa dar acesso aos dados de um ambiente múltiplo, hierárquico e distribuído de SBDs, tendo como foco a solução das dificuldades causadas pelas desconexões freqüentes dos dispositivos portáteis com a rede móvel. Este modelo compõe-se de um mecanismo de controle de concorrência (*V-lock*) cujo critério de correção é a serialização de transações globais, que é realizada através de bloqueios que seguem as regras do protocolo tradicional 2PL. Os bloqueios são realizados com base em informações contidas em árvores hierárquicas de esquemas globais de dados, objetivando produzir históricos globais serializáveis, e detectar/desfazer os bloqueios cíclicos (*deadlocks*). O modelo também apresenta um mecanismo de replicação de dados nos dispositivos móveis (*p-caching*), visando aumentar a disponibilidade dos dados. A seguir, descreve-se o processamento das transações.

Ao receber uma transação móvel, em um *host* móvel, o *V-lock* localiza, através do SSM, o nó fixo que contém os esquemas de dados referenciados na transação. Este nó torna-se o coordenador da transação e realiza os seguintes procedimentos: gera as sub-transações, atualiza a tabela local de bloqueios e o grafo local de espera, com base nos dados a serem acessados, e envia estas informações para os nós fixos da rede onde deverão ser processadas as sub-transações. Nestes últimos nós, serão atualizadas a tabela de bloqueios e o grafo de espera locais.

Este modelo apresenta as seguintes desvantagens: dependência de servidores fixos para armazenar dados e para realizar o controle de concorrência; possibilidade de conflitos indiretos provocados por transações globais gerenciadas por distintos nós fixos; violação parcial da autonomia de SBDs pré-existentes devido ao uso do protocolo tradicional de confirmação (2PC).

3.2.5. Comunidades de Bancos de Dados

Tradicionalmente, os ambientes de computação móvel têm sido desenvolvidos como uma combinação de computadores fixos e móveis, onde os computadores fixos estão interconectados através de uma rede de alta velocidade, e, dotados de uma interface de comunicação sem fio, agem como estações de suporte à mobilidade, controlando o tráfego entre os dispositivos móveis e os computadores da rede fixa [54]. Entretanto, com a crescente evolução na performance e na capacidade de comunicação dos dispositivos portáteis, alguns destes dispositivos adquiriram a capacidade de se comunicar sem a participação de qualquer componente da rede fixa, podendo formar estruturas altamente dinâmicas que se formam de maneira espontânea. Estas estruturas são chamadas redes *ad hoc* [55]. Em contraste com os ambientes tradicionais de computação móvel, onde a localização dos servidores de dados e as conexões de rede são relativamente estáveis, nas redes *ad hoc* a migração dos *hosts* e a instabilidade das conexões são a norma.

Naturalmente, os usuários destes ambientes desejam compartilhar informações entre si. Este compartilhamento pode ser realizado através da criação de federações dinâmicas de bancos de dados. Tais federações são denominadas comunidades de bancos de dados móveis (*Mobile Database Communities - MDbC's*). Uma MDbC é um ambiente dinamicamente configurável composto de múltiplos bancos de dados heterogêneos, autônomos, distribuídos e móveis, no qual cada usuário pode acessar o banco de dados dos demais usuários através de uma infra-estrutura de comunicação sem fio (figura 3.4). Cada banco de dados neste ambiente é chamado de membro ou participante da comunidade. Assim, uma MDbC é um sistema de bancos de dados múltiplos onde os computadores que armazenam bancos de dados locais podem mover-se livremente, e assim, apresentam uma conexão intermitente. Desta forma, os mecanismos propostos para o controle de concorrência em sistemas de bancos de dados múltiplos e móveis (MMDBSs) não são adequados para MDbC's. Logo, novas abordagens estão sendo estudadas.

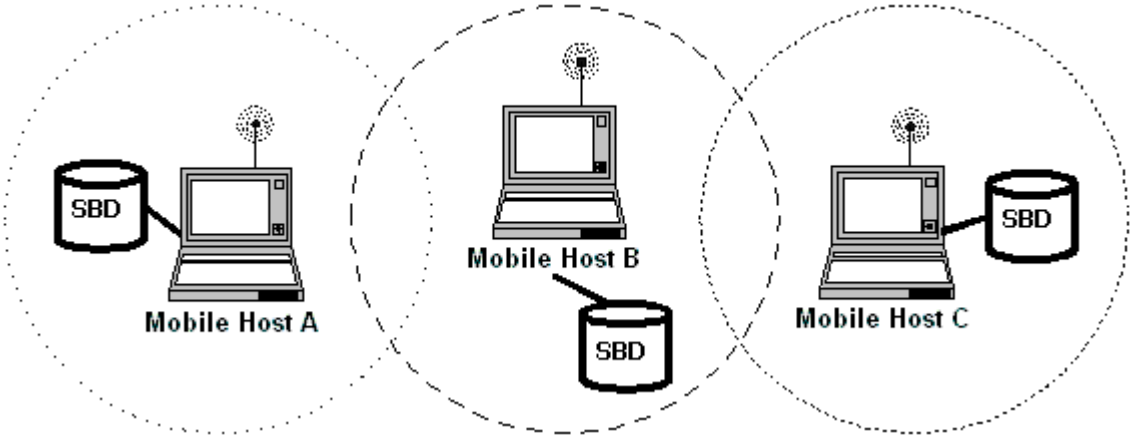

Figura 3.4. Uma possível arquitetura para uma MDbC

3.2.5.1. SESAMO

O SESAMO (acrônimo de SErializabilidade Semântica Aplicada à MObilidade) é um mecanismo distribuído de controle de concorrência em SBDs móveis proposto para a arquitetura AMDB [56,57] e que adota a Serializabilidade Semântica [46,48] como modelo de transações, o qual flexibiliza o conceito de serialização global. No SESAMO, uma transação móvel é originada em um dispositivo de computação móvel (DCM origem), sendo dividida em sub-transações independentes que são enviadas para execução em SBDs móveis distintos, localizados remotamente (DCMs destinos). O SESAMO garante a autonomia de SBDs móveis, DCMs origens e DCMs destinos na execução e gerenciamento de transações locais, transações móveis e sub-transações, respectivamente. A proposta do SESAMO é descentralizar o controle de concorrência para transações móveis, de forma que cada DCM gerencie, de forma independente de qualquer outra unidade de computação, o acesso de transações móveis a múltiplos SBDs móveis. O SESAMO utiliza o conceito de Bloqueio Semântico, proposto em [46].

3.2.6. AMBIENTES DE BROADCAST

A computação móvel está proporcionando o surgimento de novas e sofisticadas aplicações em banco de dados. Muitas delas apresentam como características um grande número de clientes móveis, um pequeno número de servidores e um banco de dados relativamente pequeno. Como exemplo podemos citar as aplicações de comércio eletrônico, tais como leilões e propostas eletrônicas, sistemas de controle de tráfego e automação industrial [52]. Enquanto tradicionalmente os dados são enviados dos servidores para os clientes sob demanda (*pull-based*), estas aplicações se beneficiam do modo de disseminação de dados baseado em difusão (*push-based*). Nele, o servidor repetidamente difunde (*broadcast*) os itens de dados para uma população de clientes sem que haja uma requisição específica por partes dos clientes. Cada período do *broadcast* é chamado de *broadcast cycle* ou *bcycle*, enquanto que o conteúdo do broadcast é chamado de *bcast*. Os clientes monitoram o canal de *broadcast* e retiram os itens de dados que necessitam [51]. Enquanto estes itens são enviados por *broadcast*, as transações que estão sendo executadas no servidor podem atualizar os valores dos itens de dados. Desta forma, distinguimos dois tipos de transações: as transações do servidor e as transações dos clientes, as quais chamamos de transações móveis.

Os ambientes de *broadcast* apresenta diversas vantagens. O servidor na rede fixa não fica sobrecarregado com pedidos de requisições e não envia várias mensagens individuais que teriam que ser transmitidas em sistemas *pull-based*. Além disso, os dados podem ser acessados concorrentemente por qualquer número de clientes sem nenhuma degradação de performance. Entretanto, o acesso aos dados é estritamente seqüencial, ou seja, os clientes necessitam esperar que os dados de seu interesse apareçam no canal. Isto aumenta a latência de acesso à informação, a qual é proporcional à quantidade de informações a serem transmitidas em um *bcast*.

O conceito de disseminação de dados por *broadcast* não é novo. Trabalhos anteriores incluem o projeto *datacycle* e o sistema de informações da comunidade de Boston (BCIS). No projeto *datacycle*, um banco de dados circula sobre uma rede de alta velocidade (140 Mbps). Os usuários consultam o banco de dados filtrando as informações relevantes através de um *hardware* especial (*special massively parallel transceiver*). Herman [58] discute um suporte transacional na arquitetura *datacycle*. Ele utiliza um mecanismo de controle de concorrência baseado em seriabilidade, os quais já mostramos serem muito dispendiosos, restritivos e desnecessários em tais ambientes. No sistema BCIS, notícias e informações são enviadas por *broadcast*, através de um canal de FM, para clientes com computadores pessoais equipados com receptores de rádio.

Em [60], os autores discutem a relação entre dados correntes (atuais) e problemas de performance quando alguns itens de dados enviados por broadcast são atualizados por processos executados no servidor. Entretanto, as atualizações não estão associadas a uma semântica transacional. As atualizações são realizadas somente por processos executados no servidor, enquanto que os processos nos clientes são somente de leitura.

A disseminação de dados baseada em *broadcast* está se transformando em um dos principais modos de transferência de informações em ambientes de computação móvel e comunicação sem fio [63, 59]. Muitos destes sistemas têm sido propostos [65,64] e já existem alguns produtos comerciais para disseminação de informações em redes de comunicação sem fio como *AirMedia* (*www.airmedia.com*) que envia regularmente notícias (manchetes e resumos) da CNN para usuários de computadores móveis, e o *DirectPC* (*www.directpc.com*) que busca informações em servidores Web, envia para uma rede de satélites e, em seguida, difunde as mensagens para computadores pessoais em velocidades de até 400 kbps.

3.2.6.1. ARQUITETURA REFERÊNCIA

Os componentes da arquitetura básica para um ambiente de disseminação de dados baseado em *broadcast* são apresentados na figura 3.5. O banco de dados consiste em uma coleção de itens de dados inter-relacionados. O servidor de banco de dados (SGBD) é responsável por armazenar e gerenciar as informações de um banco de dados. O servidor de *broadcast* periodicamente difunde os itens de dados. Os clientes representam computadores móveis nos quais estão sendo executadas aplicações que realizam operações de leitura e escrita sobre os itens de dados.

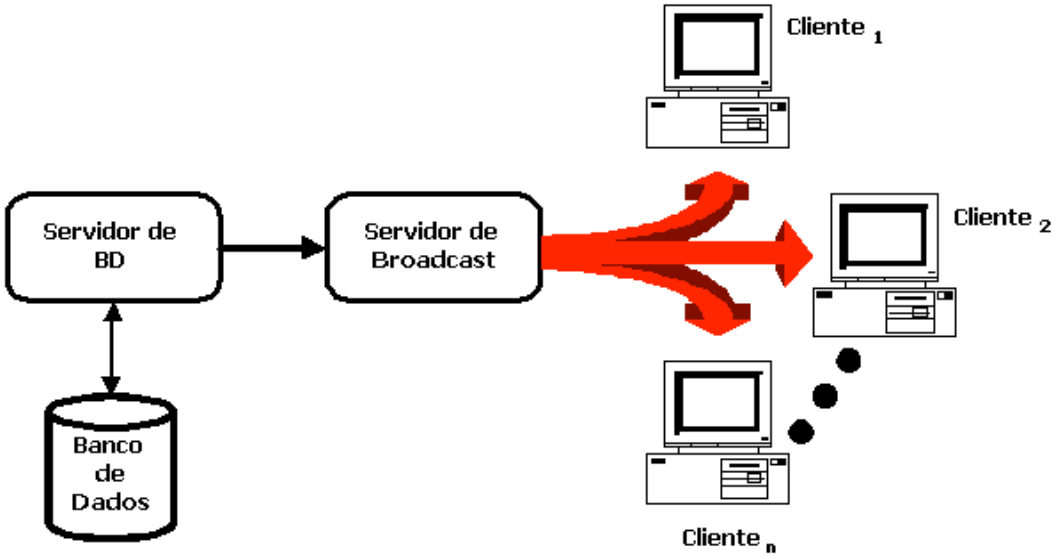

Figura 3.5. Disseminação de dados baseada em broadcast

Recentemente, o modo de disseminação de dados baseado em difusão tem recebido uma considerável atenção na área de computação móvel por causa do suporte físico para o *broadcast* existente nas redes celulares e via satélites.

As aplicações, em ambientes de *broadcast*, necessitam ler dados atuais e consistentes apesar das atualizações que podem ocorrer no servidor ou até mesmo nos clientes móveis. Por isso, as seguintes propriedades devem ser satisfeitas:

(1) Consistência Mútua: Os dados mantidos no servidor e os dados lidos pelos clientes devem ser mutuamente consistentes.

(2) Dados Correntes: Os dados lidos pelos clientes devem ser atuais.

Todavia, devido às limitações dos ambientes móveis, garantir a consistência dos dados de maneira eficiente é um problema complexo.

3.2.6.2. ESTRUTURA DO BROADCAST

A partir do momento que um cliente necessita de um item de dado, o computador móvel deve ficar escutando o meio de comunicação até receber a informação desejada. Este é um processo que consome energia e só pode ser executado com o computador móvel no modo ativo. Além disso, é comum que os clientes móveis queiram acessar somente alguns itens específicos de dados transmitidos via difusão. Logo, é importante organizar os dados transmitidos via difusão para que os clientes escutem o meio de comunicação apenas pelo período de tempo necessário para recuperar os dados de seu interesse. Desta forma, será minimizado o consumo de energia, pois o cliente não terá que permanecer ativo durante todo o processo de difusão.

Assim, os clientes não necessitam escutar continuamente o canal de *broadcast*. Em vez disso, eles podem sintonizar apenas para ler determinados itens. Entretanto, para que esta seletividade seja possível os clientes devem ter, a priori, conhecimento sobre a estrutura do *broadcast*, a fim de determinar quando um item do seu interesse aparecerá no canal. Uma outra alternativa consiste em que o *broadcast* seja auto-descritivo, ou seja, de alguma forma as informações sobre a estrutura do *broadcast* é enviada junto com os dados. Neste caso, o cliente primeiramente lê as informações sobre o *broadcast* e em seguida usa estas informações nas leituras dos itens de seu interesse. Algumas técnicas para enviar informações de índices junto com os dados também podem ser utilizadas[50].

Uma possível estrutura para a organização do *broadcast* é mostrada na figura 3.6 [62]. Nesta estrutura, cada *bcast* é constituído por segmentos de índices e por blocos de dados. Os seguimentos de índices descrevem a organização e a ordem das informações transmitidas. A utilização de índices é de extrema importância nas situações em que um cliente está interessado em parte dos dados transmitidos, permitindo um acesso seletivo aos itens desejados. Isto possibilita que os clientes economizem energia, ficando no modo *standby* a maior parte do tempo e entrando no estado ativo apenas para recuperar os itens de seu real interesse. Os blocos de dados são formados por unidades lógicas chamadas *buckets*. O conceito de *bucket* é análogo ao de bloco em sistemas de arquivos. Cada *bucket* tem um cabeçalho que inclui informações úteis, como por exemplo, o número do ciclo corrente (atual) ou os identificadores dos itens atualizados durante o últumo ciclo. O conteúdo exato do cabeçalho pode variar, dependendo da implementação do *broadcast*. As informações no cabeçalho geralmente incluem a posição do *bucket* no *bcast*, um *offset* para o início do *bcast* e um *offset* para o início do próximo *bcast*. O *offset* para o início do próximo *bcast* pode ser usado pelo cliente para determinar o início do próximo *bcast* quando o tamanho do *bcast* não é fixo. Os itens correspondem, por exemplo, a tuplas no banco de dados. Os usuários acessam os dados através do valor de um dos atributo do registro, a chave de busca.

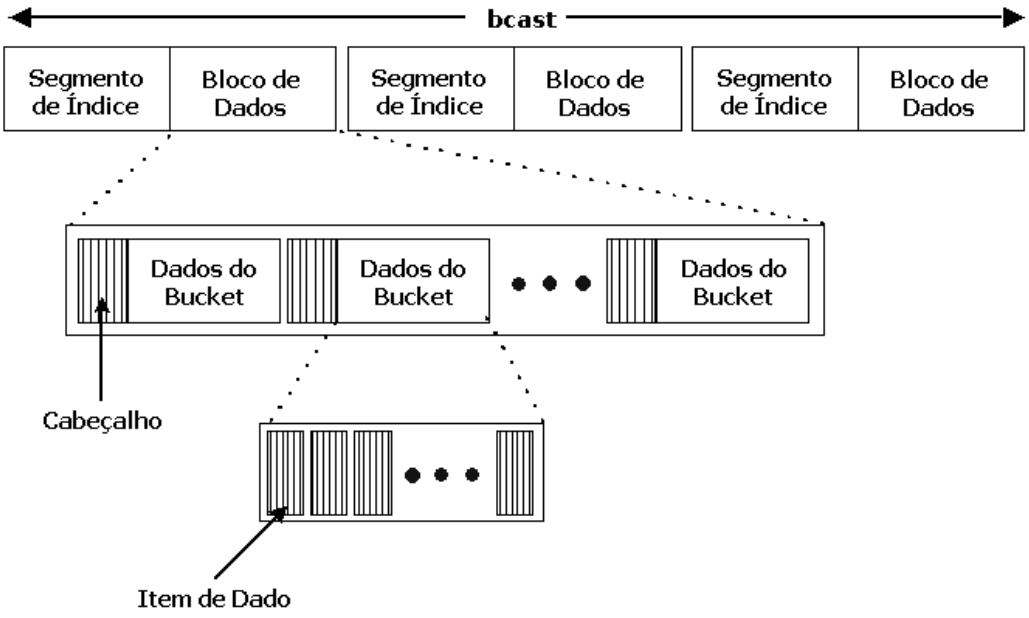

Figura 3.6. Uma possível estrutura para a organização do *broadcast*

3.2.6.3. Considerações Sobre Consistência de Dados em Ambientes de Broadcast

Os principais mecanismos para o controle de concorrência em ambientes de broadcast consideram que as transações nos clientes são somente de leitura, ou seja, todas as atualizações são executadas no servidor e disseminadas a partir deste para a população de clientes. Esta consideração é razoável, pois a maioria das transações em sistemas de difusão são somente de leitura. Além disso, mesmo que as transações de atualização fossem permitidas nos clientes, seria mais eficiente processá-las com algoritmos especiais.

Em um ambiente de computação móvel com disseminação de dados por *broadcast* o servidor periodicamente difunde o conteúdo do banco de dados para a população de clientes móveis. Um banco de dados consiste de um conjunto finito de itens de dados. Enquanto os itens de dados são enviados por *broadcast*, as transações que estão sendo são executadas no servidor podem atualizar os valores dos itens que foram enviados em *broadcast*. Vamos assumir que os valores dos itens de dados enviados em *broadcast* durante cada ciclo correspondem ao estado do banco de dados até o início do broadcast, ou seja, correspondem aos valores produzidos por todas as transações que executaram operações de *commit* até o início do ciclo. Iremos denominar essas transações de "*committed*". Desta forma, garantimos que o conteúdo do *broadcast* a cada ciclo é consistente.

Portanto, uma transação somente de leitura que lê todos os seus dados em um único ciclo pode ser executada sem problemas, pois todos os valores lidos estão consistentes.

Um critério de corretude que pode ser usado no caso de transações somente leitura é que cada transação deve ler dados que correspondem a um mesmo estado do banco de dados, ou seja, a um único *bcast*.

Visto que o conjunto de itens a serem lidos por uma transação não é conhecido a priori e o acesso aos dados é seqüencial, as transações podem ler itens de dados de diferentes *bcasts*, ou seja, valores de diferentes estados do banco de dados. Esta é a principal desvantagem desta abordagem.

Consideremos, por exemplo, a transação T que corresponde ao seguinte programa:

if a > 0 then read b else read c

Suponha que os itens "b" e "c" precedem o item "a" no *broadcast*. Logo, a transação cliente tem que primeiramente ler o item "a" e esperar o próximo ciclo para ler os itens "b" e "c". Caso as transações clientes leiam itens de dados de diferentes ciclos, não é garantido que os valores lidos sejam consistentes.

Esta abordagem é utilizada como critério de corretude por várias das propostas que discutiremos a seguir.

3.2.6.4. Principais Abordagens para Consistência de Dados em Ambientes de Broadcast

Com o objetivo de solucionar eficientemente a questão do controle de concorrência em ambientes de broadcast, várias abordagens têm sido apresentadas. Nesta seção descreveremos as principais propostas existentes para a solução deste complexo problema.

3.2.6.4.1. Relatório de Invalidação

Neste mecanismo, proposto em [51], cada *bcast* é precedido por um relatório de invalidação na forma de uma lista que inclui todos os itens de dados que foram atualizados no servidor, por transações *committed*, durante o ciclo de *broadcast* anterior. Para cada transação somente leitura ativa R, o cliente guarda um conjunto *Read_Set*(R) de todos os itens de dados lidos por R. Vamos definir *readset* de uma transação T, denotado por *Read_Set*(T), o conjunto dos itens lidos pela transação T. Em particular, *Read_Set*(T) é um conjunto de pares ordenados na forma (item lido, valor lido). O *readset* de cada transação somente leitura deve ser um subconjunto de um estado consistente do banco de dados. No início de cada *bcast*, o cliente sintoniza e lê o relatório de invalidação. A

transação somente leitura R é abortada se um item x ∈ *Read_Set*(R) foi atualizado, isto é, se x aparece no relatório de invalidação.

[51] apresenta o seguinte teorema: O relatório de invalidação produz transações somente leitura corretas.

Uma transação somente leitura R lê os valores mais recentes, isto é, os valores produzidos por todas as transações *committed* até o início do ciclo onde R executa o *commit*. Desta forma, todos os valores lidos pela transação R correspondem a um único *bcast*, ou seja, a um único estado do banco de dados.

Esta abordagem requer que cada cliente leia todos os relatórios de invalidação, que são enviados antes de cada ciclo. Desta forma, o cliente não pode desconectar-se por muito tempo. Para tentar resolver este problema, os relatórios de invalidação podem ser enviados em uma freqüência menor. Neste caso, antes que uma transação somente de leitura execute o *commit* ela deve ler o próximo relatório de invalidação que aparecer no canal de *broadcast*.

Além de ser uma abordagem bastante simples, o cliente não necessita contactar o servidor e as informações de controle transmitidas são de tamanho relativamente pequeno. Por outro lado, ela descarta várias histórias serializáveis por conflito. Além disso, o cliente não pode desconectar-se por muito tempo, pois precisa ler todos os relatórios de invalidação, independentemente de sua freqüência, o que torna esta proposta inviável.

3.2.6.4.2. Invalidação Baseada em Versão

Neste método, descrito em [51], um marcador de tempo ou um número de versão é enviado junto com cada item de dado. Este número de versão corresponde ao número do ciclo onde o item foi atualizado pela última vez. Para cada transação somente de leitura ativa R, o cliente guarda um conjunto *Read_Set*(R) de pares ordenados formados pelos itens de dados lidos até o momento e seus respectivos números de versão. Por exemplo, o par ordenado (x, C_i) indica que o item de dado x foi atualizado pela última vez durante o ciclo C_i. Seja C_o o número do ciclo no qual uma transação R executa sua primeira

operação de leitura. A cada nova leitura, o seguinte teste deve ser realizado: se um item $x \in Read\_Set(R)$ foi atualizado, isto é, se $C_x > C_o$ então a transação R deve ser abortada.

O seguinte teorema é demonstrado em [51]: A invalidação baseada em versão produz transações somente leitura corretas.

Uma transação somente leitura R lê os valores produzidos por todas as transações *committed* até a execução da primeira operação de leitura pela transação R . Desta forma, todos os valores lidos pela transação R correspondem a um único *bcast*, ou seja, a um único estado do banco de dados.

Como no relatório de invalidação, cliente não necessita contactar o servidor. Contudo, o cliente não necessita ler todos os *bcasts*, podendo desconectar-se por qualquer período de tempo. Como desvantagem desta proposta podemos citar: descarta várias histórias serializáveis por conflito; além disso, a cada nova leitura deve-se ler também o número de versão de todos os itens lidos anteriormente.

3.2.6.4.3. MÚLTIPLAS VERSÕES

A idéia básica desta proposta, apresentada em [51], consiste em manter temporariamente versões anteriores dos itens de dados a fim de diminuir o número de *aborts* nas transações somente leitura. Em uma abordagem particular, *S-Multiversion*, para cada item de dado são armazenados os seus S valores anteriores, ou seja, os valores do item durante os S *bcycles* anteriores, onde S é o máximo *span* entre todas as transações somente leitura. Vamos definir *span* de uma transação T, *span*(T), como sendo o número máximo de ciclos de *broadcast* onde a transação T lê itens de dados. Então, para uma transação T qualquer, se *span*(T) = 1, T é correta. Para implementar este esquema, o servidor, além de difundir o último valor *committed* para cada item de dado, mantém e difunde múltiplas versões para cada item de dado. Pelo menos um valor, a versão corrente, é difundido para cada item de dado. A cada *bcycle k*, o servidor descarta as *k-S* versões do *bcast*.

Seja C_0 o número do ciclo de *broadcast* durante o qual a transação cliente R executada sua primeira operação de leitura. Durante C_0, a transação R lê o valor mais atual de cada item de dado. Nas leituras posteriores, R lê o valor com o maior número de versão C_n , tal que $C_n \leq C_0$.

Em [51] encontramos a demonstração do seguinte teorema: *S-Multiversion* produz transações somente leitura corretas.

Desta forma, uma transação somente leitura R lê os valores produzidos por todas as transações *committed* até o início do *bcycle* C_0. Assim, todos os valores lidos pela transação R correspondem a um único *bcast*, ou seja, a um único estado do banco de dados.

Basicamente, existem três meios para o armazenamento das versões anteriores. Um meio de armazenamento em potencial é o ar, neste caso, as versões anteriores são difundidas junto com os valores atuais dos itens de dados. A Segunda possibilidade é manter as versões anteriores em um *cache* no cliente. Neste caso, é possível termos um coletor de lixo para as versões antigas, desde que existam informações sobre as transações ativas no cliente. Por último, podemos manter parte do banco de dados nos clientes na forma de visões materializadas.

O uso de múltiplas versões proporciona um aumento no grau de concorrência. Além disso o cliente não necessita contactar o servidor para executar as operações de leitura. Por outro lado, esta abordagem descarta várias histórias serializáveis por conflito, provoca um *overhead* na leitura e manutenção das múltiplas versões e requer que o cliente não fique desconectado por muito tempo.

Caso as múltiplas versões sejam armazenadas em *caches* nos clientes, relatórios de invalidação são enviados, a cada *bcycle*, para garantir a leitura de dados correntes. Desta forma, o cliente necessita escutar o canal de *broadcast* a cada *bcycle*. Para tentar resolver este problema, os relatórios de invalidação podem ser enviados em uma freqüência menor. Neste caso, antes que uma transação somente de leitura execute o *commit* ela deve ler o próximo relatório de invalidação que aparecer no canal de *broadcast*.

3.2.6.4.4. TESTE DO GRAFO DE SERIALIZAÇÃO (SGT)

Tanto os métodos baseados em invalidação quanto o de múltiplas versões garantem que as transações somente leitura lêem valores consistentes, isto é, valores produzidos por uma execução serializável, pois os valores lidos por estas transações correspondem a um único *bcast*, ou seja, a um único estado do banco de dados. No caso do métodos do relatório de invalidação, os valores lidos correspondem ao *bcast* do final da transação, enquanto no método de múltiplas versões os valores lidos correspondem ao *bcast* do início da transação. Entretanto, é suficiente que as transações leiam valores que correspondam a qualquer estado consistente, não necessariamente um estado que tenha sido enviado por *broadcast*. Neste fato baseia-se o método do teste do grafo de serialização, proposto em [*31*].

O grafo de serialização de uma história H, denotado por SG(H), é um grafo direcionado onde os nós são as transações *committed* em H e as arestas são da forma $T_i \rightarrow T_j$ $(i \neq j)$ tal que uma operação de T_i precede e conflita com uma operação de T_j em H. De acordo com o teorema da seriabilidade, uma história H é serializável se e somente se SG(H) é acíclico.

O método SGT trabalha da seguinte forma. Cada cliente mantém localmente uma cópia do grafo de serialização. O grafo de serialização no servidor inclui somente as transações *committed* no servidor, enquanto, o grafo mantido nos clientes incluem as transações *committed* no servidor e as transações somente leitura ativas neste cliente. A cada ciclo, o servidor difunde qualquer atualização no grafo de serialização. Após receber as atualizações, o cliente integra estas atualizações em sua cópia local do grafo. Uma operação de leitura no cliente é executada somente se ela não cria um ciclo no grafo de serialização local. O grafo de serialização no servidor não é necessariamente usado para o controle de concorrência no servidor, um método mais prático, como o de bloqueio em duas fases pode ser utilizado.

Neste método, o servidor difunde, no início de cada *bcast*, as seguintes informações de controle:

- As alterações no grafo de serialização

  Em particular, o servidor difunde para cada transação T_i que foi *committed* durante o ciclo anterior, uma lista das transações com a qual ela conflita, isto é, as transações com a qual T_i está diretamente conectada por uma aresta.

- Um relatório de invalidação

  Este relatório inclui todos os itens de dados escritos durante o *bcycle* anterior juntamente com o identificador da transação que primeiramente escreveu cada item.

Estas informações de controle são utilizadas pelos clientes na montagem e atualização do grafo de serialização. Assim, cada cliente sintoniza no início do *broadcast* para obter as informações de controle. Em seguida, o cliente atualiza a sua cópia local do grafo de serialização da seguinte forma: No início de cada $bcycle_{i+1}$, o cliente adiciona arestas para todas as transações somente leitura ativas da seguinte forma: Seja R uma transação ativa e $RS_i(R)$ o conjunto dos itens lidos por R até o $bcycle_i$. Para cada item x presente no relatório de invalidação tal que $x \in RS(R)$, o cliente adiciona uma aresta $R \rightarrow T_f$, onde T_f é a primeira transação que escreveu em x durante o $bcycle_i$. Desta forma R conflita com todas as transações que escreveram em x durante o $bcycle_i$.

O teorema a seguir é apresentado em [51]: O método SGT produz transações somente leitura corretas.

Nesta abordagem, o cliente não necessita contactar o servidor para executar as operações de leitura. Além disso, algumas histórias que não seriam aceitas na serialização tradicional são consideradas corretas. Por outro lado, o cliente não pode desconectar-se por muito tempo, pois precisa ler as informações de controle enviadas no início de cada *bcycle*. Para ler estas informações, o cliente necessita escutar o canal de *broadcast* durante um período de tempo maior, o que se torna problemático devido às limitações no fornecimento de energia pelas baterias dos computadores portáteis. O cliente também precisa manter e testar o grafo de serialização. Entretanto, os computadores portáteis têm, em geral, uma pequena capacidade de memória.

3.2.6.4.5. CONSISTÊNCIA DE ATUALIZAÇÃO

Esta abordagem, proposta em [52], utiliza como critério de corretude uma adaptação do critério proposto no contexto do controle de concorrência em sistemas de múltiplas versões, chamado consistência de atualização. Nele uma execução é dita consistente em atualização se e somente se as duas condições a seguir forem satisfeitas:

(1) Todas as transações de atualização são serializáveis;

(2) Cada transação somente de leitura R é serializável com relação ao conjunto das transações que, direta ou indiretamente, atualizaram valores que foram lidos pela transação R;

Determinar se uma história é consistente em atualização é um problema NP-Completo [52]. Por isso, este critério foi adaptado e um algoritmo de verificação, aproximativo e de tempo polinomial, APPROX, foi desenvolvido para determinar de forma eficiente histórias legais [52].

A seguir descrevemos com maiores detalhes o critério de corretude adaptado.

Seja T uma transação que executa em uma história (*schedule*) H.

Seja $LIVE_h$ (T) o conjunto mínimo fechado onde:

(a) $T \in LIVE_h$ (T)

(b) Se $T' \in LIVE_h$ (T), para toda transação T'' tal que T' lê um valor de um item de dado escrito por T'', $T'' \in LIVE_h$ (T)

H_{UPDATE} é a projeção da história H que inclui apenas as operações de todas as transações que executam alguma operação de escrita.

Uma determinada história S é legal se e somente:

(1) H_{UPDATE} é serializável por conflito;

(2) Pata toda transação somente de leitura R em H, S_H (R) é acíclico. Onde S_H (R) é o grafo de serialização consistindo somente das transações em $LIVE_h$ (R)

Uma implementação do algoritmo APPROX, chamada F_MATRIX, foi apresentada em [52]. Nesta implementação, o servidor difunde, durante cada ciclo, o último valor *committed* dos itens de dados juntamente com uma matriz de controle, a qual será usada pelos clientes para determinar se as transações somente leitura lêem valores consistentes.

As informações de controle são enviadas na forma de uma matriz C, de ordem n X n. Onde n é o número de itens do banco de dados.

Considerando que os itens de dados possuem identificadores eles podem ser representados da seguinte forma; O_{b1} .... O_{bn}.

Cada entrada da matriz de controle C, C(i,j), terá como valor o número de um determinado ciclo do *broadcast*.

Seja H a história das transações de atualização *committed* no servidor e T_j a última transação *committed* que escreve em O_{bj}.

A matriz de controle C é montada da seguinte forma:

Vamos considerar que uma transação hipotética T_0 escreve todos os itens no ciclo 0. Então,

C(i,j) max $_{T' \in LIVEH\ (Tj)\ \wedge\ T'escreve\ em\ Obi}$ ($C_{T'}$).

Onde $C_{T'}$ é o número do ciclo onde T' executa o *commit*.

No cliente, antes de uma operação de leitura ser executada sobre um item de dado, durante um determinado ciclo do *broadcast*, as informações de controle transmitidas durante este ciclo são consultadas para determinar se a operação de leitura pode proceder. Caso a operação de leitura não possa proceder a transação é abortada. Abaixo, mostramos como é feita esta validação.

Para uma determinada transação somente de leitura R em um cliente, o seguinte protocolo é seguido antes de cada operação de leitura:

Seja RS(R) o conjunto, (O_{bi}, *ciclo*), das leituras executadas anteriormente pela transação T, ou seja, T leu o último valor *committed* de O_{bi} até o início do ciclo *ciclo*.

Então, a leitura sobre um item O*bj* é permitida se e somente se:

$\forall$ (O_{bi}, *ciclo*) $\in$ RS(R) (C(i,j) < *ciclo*)

Se esta condição falhar então a transação é abortada.

O seguinte teorema é apresentado em [52]: Uma transação somente leitura R pode executar o *commit* seguindo o protocolo acima se e somente se SG(R) é acíclico.

Nesta abordagem, o cliente não necessita contactar o servidor para executar as operações de leitura, nem escutar continuamente o canal de *broadcast*. Além disso, algumas histórias que não seriam aceitas na serialização tradicional são consideradas corretas. Por outro lado, o cliente precisa ler a matriz de controle, enviada no início de cada *bcycle*. Para ler estas informações, o cliente necessita escutar o canal de *broadcast* durante um período de tempo maior, o que se torna problemático devido às limitações no fornecimento de energia pelas baterias dos computadores portáteis e ao alto custo da transmissão. O cliente também precisa armazenar uma matriz de ordem n X n (onde n é o número de itens do banco de dados), entretanto, os computadores portáteis têm, em geral, uma pequena capacidade de memória. Uma outra desvantagem consiste no fato de que determinar se uma história é consistente em atualização é um problema NP-Completo. Por isso, o algoritmo aproximativo polinomial APPROX é usado. Entretanto, este algoritmo descarta algumas histórias corretas segundo o critério da consistência de atualização. Outro fato importante é que o servidor precisa manter uma estrutura de dados $LIVE_h$ (T) para cada transação de atualização T, o que pode degradar a performance das transações submetidas no servidor.

3.2.6.4.6. Teste do Grafo de Serialização Temporal (TGST)

O protocolo, denominado teste do grafo de serialização temporal (TGST), é baseado em uma estratégia similar a utilizada pelo mecanismo de teste do grafo de serialização proposto em [61]: o monitoramento e gerenciamento dinâmico de um grafo que deve ser sempre acíclico. Em contraste com o protocolo do teste do grafo de serialização, o protocolo TGST explora informações temporais referentes ao momento em que um item de dado foi lido ou atualizado.

No TGST as funções para o controle de concorrência estão divididas entre os clientes e o servidor. Portanto, assumimos que o servidor e os clientes apresentam funcionalidades específicas para o gerenciamento das transações. A seguir descreveremos estas funcionalidades [49].

Durante cada ciclo de *broadcast*, o servidor difunde os valores dos itens de dados juntamente com os marcadores de tempo correspondentes. Estes marcadores são definidos da seguinte forma: o número do ciclo que a transação T_i executou sua operação de *commit*, se $w_i(x) \in OP(T_i)$ e T_i é a última transação que executou uma operação de escrita sobre x. Quando uma transação móvel T_k executar uma operação pk(x), esse marcador de tempo será associado a $p_k(x)$. Para as operações das transações executadas no servidor o marcador de tempo continua sendo o número do ciclo onde a operação é realizada. Os marcadores de tempo podem ser enviados no cabeçalho da mensagem ou junto com cada item de dado. Vamos assumir que os valores dos itens de dados enviados em *broadcast* durante cada ciclo correspondem ao estado do banco de dados até o início do *broadcast*, ou seja, correspondem aos valores produzidos por todas as transações que executaram operações de *commit* até o início do ciclo. Essas transações são denominadas "*committed*". Desta forma, o servidor mantém duas versões de cada item: o último valor *committed* e o último valor escrito. O servidor monta e gerencia o grafo de serialização temporal para o *schedule* global, envolvendo tanto as transações do servidor quanto as dos clientes móveis.

A cada operação de leitura, o cliente guarda o valor e o identificador do item lido, juntamente com o marcador de tempo correspondente. Periodicamente, o cliente deve enviar um pacote (mensagem) ao servidor contendo os itens lidos até o momento, juntamente com os seus marcadores de tempo. As leituras já informadas não precisam ser enviadas novamente. Quando um cliente recebe um pedido de *commit* ou *abort* para uma transação móvel T_i, este deve enviar uma mensagem ao servidor contendo essa solicitação para T*i*. O cliente deve esperar a resposta para efetuar a operação de *commit* ou de *abort*.

Um escalonador (scheduler) implementando o protocolo TGST funciona como descrito a seguir. Quando um escalonador inicia sua execução o GST é criado como um grafo vazio. Com base no marcador de tempo descrito no parágrafo anterior, o grafo de serialização temporal é construído como descrito abaixo:

Passo 1. Para cada operação $p_i(x) \in OP(T_i)$ que o escalonador recebe, ele checa se existe uma operação $q_j(x) \in OP(T_j)$ que conflita com $p_i(x)$ e que já foi escalonada. Caso $q_j(x)$ exista, uma aresta será inserida entre T_i e T_j. Para que esta aresta seja incluída corretamente deveremos considerar dois casos distintos, os quais descrevemos a seguir:

Caso 1. T_i é uma transação executada no servidor. Neste caso, será executada uma verificação temporal da seguinte forma:

Se $C(q_j(x)) \leq C(p_i(x))$

Então o escalonador insere uma aresta da forma $T_j \rightarrow T_i$.

Se não

uma aresta da forma $T_i \rightarrow T_j$ será inserida no grafo.

Caso 2. T_i é uma transação móvel. Neste caso, será executada uma verificação temporal da seguinte forma:

Se $C(q_j(x)) < C(p_i(x))$

Então o escalonador insere uma aresta da forma $T_j \rightarrow T_i$

Se não

Se $C(q_j(x)) > C(p_i(x))$

Então o escalonador insere uma aresta da forma $T_i \rightarrow T_j$

Se não

Se T_j já executou o *commit*

Então o escalonador insere uma aresta da forma $T_j \rightarrow T_i$

Se não o escalonador insere uma aresta da forma $T_i \rightarrow T_j$

Passo 2. O escalonador verifica se a nova aresta introduz um ciclo no grafo de serialização temporal. Em caso afirmativo, o escalonador rejeita a operação $p_i(x)$, desfaz o efeito das operações de T_i e remove a aresta inserida. Caso contrário, $p_i(x)$ é aceita e escalonada.

Em [49] encontramos a demonstração do seguinte teorema: Seja TGST o conjunto de *schedules* sobre um conjunto T de transações produzido pelo protocolo TGST e CSR o conjunto de todos os *schedules* serializáveis por conflito sobre T. Então, TGST = CSR.

Além de utilizar um critério de corretude já consolidado, essa abordagem ainda apresenta as seguintes vantagens. O cliente não necessita contactar o servidor para solicitar as operações de leitura, nem escutar continuamente o canal de *broadcast*. Diferentemente das propostas apresentadas em [51] e [52], as informações enviadas do servidor para os clientes a cada ciclo consistem apenas dos valores dos itens de dados e seus marcadores de tempo. Essa característica garante que a utilização do canal de *broadcast* é minimizada, o que proporciona uma economia no custo de transmissão e da limitada capacidade de energia dos computadores portáteis. Além disso, os clientes só precisam armazenar os valores dos itens de dados de seu real interesse, o que economiza

os seus parcos recursos de memória. Essa última propriedade não é garantida pelas propostas [51] e [52], que foram discutidas anteriormente. Outra grande vantagem é que as transações móveis podem executar operações de escrita, ou seja, não são somente de leitura.

Por outro lado, o servidor precisa manter um grafo de serialização de todas as transações *committed* e ativas, incluindo tanto as transações do servidor quanto as transações dos clientes móveis, o que pode comprometer a escalabilidade. Outra desvantagem consiste no fato dos clientes terem que comunicar ao servidor os itens lidos por suas transações.

Em [49], o protocolo TGST é estendido a fim de utilizar como critério de corretude a serializabilidade semântica proposta inicialmente em [46]. Este critério foi examinado com a finalidade de introduzir um maior grau de concorrência entre as transações, diminuindo o número de transações abortadas e aumentando a eficiência do protocolo TGST.

A serializabilidade semântica, proposta em [46], baseia-se na utilização de informações semânticas sobre os objetos do banco de dados (e não sobre as transações). A idéia principal consiste em proporcionar diferentes visões de atomicidade para cada transação e, por esta razão, permitir um maior entrelaçamento entre as transações. Isto é alcançado através da aceitação de *schedules* que não são serializáveis, mas que preservam a consistência do banco de dados.

Desta forma, os autores acreditam que o protocolo TGST terá ganhos em eficiência com a utilização da serializabilidade semântica, pois, uma vez que o grau de entrelaçamento entre as transações aumenta o número de transações abortadas diminui, o que é de grande importância se considerarmos que as transações em ambientes de computação móvel são de longa duração.

4. Estudo de Caso: O Sistema Bayou

4.1 Introdução

Bayou [40,41,42,43] é um sistema de armazenamento replicado, que utiliza um nível fraco de consistência e que foi projetado para ambientes de computação móvel que inclui dispositivos portáteis fracamente conectados. O sistema Bayou provê uma infra-estrutura para o desenvolvimento de uma variedade de aplicações colaborativas, que não tenham requisitos de tempo real, tais como calendários, e-mail e edição de documentos para grupos desconectados.

4.2 Arquitetura do Sistema

No sistema Bayou, cada banco de dados é inteiramente replicado em um conjunto de servidores. As aplicações executam como clientes e interagem com os servidores através de uma API específica, denominada *Bayou Application Programming Interface*, a qual suporta duas operações básicas: leitura e escrita. Os clientes podem executar tanto leituras quanto escritas em qualquer um dos servidores, com os quais possa se comunicar. O cliente e o servidor podem co-existir em um determinado *host*, ou podem executar de forma isolada. Assim, em contraste com o sistema Coda que faz uma distinção clara entre unidades móveis executando como clientes e *hosts* fixos fortemente conectados a redes fixas executando como servidores, a arquitetura Bayou trata os dispositivos móveis como um ponto em um modelo ponto a ponto (*peer-to-peer*) onde os servidores estão conectados de forma intermitente uns com os outros.

O sistema Bayou não provê suporte à replicação transparente para as aplicações que executam na camada superior. As aplicações têm o conhecimento de que podem ler dados inconsistentes e que suas escritas são uma tentativa.

Entretanto, o conhecimento específico do domínio da aplicação é explorado com a finalidade de detectar e solucionar conflitos de forma automática.

4.3 Conectividade Fraca

O sistema Bayou suporta um esquema de replicação *read-any*/*write-any*. Quando um cliente submete uma operação de escrita a um determinado servidor, este executa a escrita imediatamente em sua réplica, sem qualquer processo de detecção de conflitos. As escritas aceitas localmente são chamadas de tentativas (*tentative*). O valor resultante de uma tentativa de escrita (*tentative write*) é imediatamente disponibilizado para leitura. Os servidores propagam as escritas entre si através de um processo baseado na comunicação entre pares denominado sessão de anti-entropia (*anti-entropy)*. Durante uma sessão, os dois servidores envolvidos trocam suas escritas, e então, ao final do processo, os servidores entram em um consenso sobre que escritas devem executar e em que ordem. O sistema garante que todos os servidores irão eventualmente receber todas as escritas e que dois servidores que receberam o mesmo conjunto de escritas irão apresentar o mesmo estado (conteúdo) do banco de dados. Para alcançar este objetivo, é necessário que as escritas sejam executadas na mesma ordem global em todos os servidores e que os procedimentos para detecção e resolução de conflitos sejam determinísticos.

Uma vez que os servidores podem receber as operações de escrita dos clientes e dos outros servidores em uma ordem diferente da ordem global, os servidores podem necessitar desfazer os efeitos de uma tentativa de escrita (*tentative write*) previamente executada e executá-la novamente em uma outra ordem. Assim, uma dada escrita pode ser executada diversas vezes em um mesmo servidor. Uma escrita se torna estável quando o servidor recebe e executa todas as escritas que a precedem. A fim de aumentar a taxa com que as escritas se tornam estáveis, Bayou usa um esquema denominada *primary-commit*: um servidor designado como primário tem a responsabilidade de executar o *commit* das atualizações, ou seja, torná-las permanente. As escritas já consolidadas (*committed writes*) são ordenadas de acordo com o momento em que elas são consolidadas no servidor primário, sendo ordenadas antes das

tentativas de escrita. A informação de que escritas foram consolidadas e em que ordem são propagadas para os demais servidores durante o processo de anti-entropia. Cada servidor matem duas visões do banco de dados: uma cópia que reflete somente os dados consolidados (*committed*) e outra cópia que reflete as tentativas de escrita conhecidas até o momento pelo servidor.

O sistema Bayou provê um método automático baseado em validação de dependências para detectar conflitos e procedimentos para a resolução automática de conflitos. As validações de dependências e os procedimentos para resolução de conflitos são incluídos juntos a cada operação de escrita. A validação de dependência consiste de uma consulta e o resultado esperado. Antes da operação de escrita ser executada no servidor, a consulta que compõe a validação de dependência é executada sobre a cópia corrente do banco de dados. Um conflito é detectado se a consulta não recupera o resultado esperado. Neste caso, a atualização requisitada não é executada e o servidor invoca o procedimento para resolver o conflito detectado. Este procedimento é um programa escrito em uma linguagem interpretada de alto nível, o qual é responsável por solucionar conflitos e produzir uma atualização modificada que será executada. Quando a resolução automática não é possível, o procedimento de resolução de conflito deve armazenar esta informação em um log, o qual será utilizado para a resolução manual de conflitos. Bayou permite que as réplicas permaneçam acessíveis mesmo quando conflitos são detectados e ainda não resolvidos. Isto permite que os clientes continuem suas operações, mas pode levar a criação de conflitos em cascata.

4.4 Mobilidade e Garantias de Sessão

As aplicações no sistema Bayou não necessitam se restringir a utilizar um único servidor, mas podem interagir com qualquer servidor com o qual consigam se comunicar. Para isso, é utilizado um esquema de replicação *read-any/write-any*. Além disso, o Bayou utiliza um mecanismo de consistência fraca, os quais são interessantes pois proporcionam alta disponibilidade, boa escalabilidade, e simplicidade de projeto. Entretanto, é possível que os clientes obtenham valores inconsistentes quando lêem dados de diferentes réplicas. Por exemplo, um

usuário pode ler o valor de um determinado item de dado e posteriormente ao executar esta operação novamente pode obter um valor anterior ao valor lido inicialmente. Um problema sério ocorre nos sistemas com consistência fraca mesmo quando um único usuário ou aplicação está atualizando os dados. Suponha que um usuário móvel executou uma operação de escrita em um determinado servidor, e mais tarde tentou ler o mesmo item de um outro servidor, mas como os servidores ainda não se sincronizaram, o cliente irá obter um valor inconsistente. Para reduzir as inconsistências observadas pelos clientes quando acessam diferentes servidores, Bayou provê garantias de sessão (*session guarantees*). Uma garantia de sessão é uma abstração para uma seqüência de operações de leitura e escrita que são submetidas durante a execução de uma aplicação. Não se tem a intenção de que as sessões sejam correspondentes ao conceito de transação atômica, que garante atomicidade e serializabilidade. Ao contrário, a intenção é proporcionar às aplicações individualmente uma visão do banco de dados que seja consistente com as suas próprias ações, mesmo que estas leiam e escrevam a partir de vários servidores, potencialmente inconsistentes. Quatro garantias são propostas. Estas garantias podem ser aplicadas independentemente nas diversas operações que pertencem à sessão. A seguir, descrevemos estas garantias:

1. **Read Your Writes**: As operações de leitura devem refletir as escritas executas anteriormente (pela mesma sessão).

2. **Monotonic Reads**: Leituras sucessivas refletem o resultado de um conjunto de escritas não decrescente.

3. **Writes Follow Reads**: As operações de escrita são propagadas após as operações de leitura das quais dependem.

4. **Monotonic Writes**: As operações de escrita são propagadas após as escritas que as precedem logicamente.

Essas propriedades são garantias no sentido que o sistema de armazenamento as assegura para cada leitura ou escrita pertencente a uma sessão, ou caso não consiga assegurá-las informa a aplicação que a garantia não pode ser alcançada.

As garantias de sessão proporcionam um meio de adaptabilidade. O grau de consistência pode ser adaptado aos vários níveis de conectividade, selecionando-se individualmente que garantias de sessão deseja-se utilizar.

Para entendermos como podemos implementar estas garantias algumas considerações necessitam ser feitas.

Cada operação de escrita tem um identificador globalmente único, chamado "WID". O primeiro servidor que receber a operação de escrita tem a responsabilidade de atribuir o WID.

Vamos definir DB(S,t) como sendo a seqüência de escritas que foram recebidas pelo servidor S até o tempo t. Se t é o tempo corrente, então ele pode ser omitido, assim, utilizamos DB(S) o conteúdo corrente do banco de dados (réplica) do servidor S. A ordem das escritas em DB(S) não corresponde necessariamente a ordem na qual S inicialmente recebeu as escritas (e sim a uma ordem resultante do processo de sincronização, como discutiremos a seguir).

A consistência fraca permite que as réplicas de diferentes servidores tenham conteúdos distintos. Isto é, DB(S1,t) não é necessariamente equivalente a DB(S2,t) para quaisquer dois servidores S1 e S2. Entretanto, na prática, os sistemas usualmente desejam atingir uma consistência eventual na qual os servidores convergem para uma cópia idêntica, após uma período sem atualizações. A consistência eventual traz duas propriedades: propagação total e ordem consistente. A seguir, discutiremos os mecanismos para se alcançar estas propriedades.

As escritas são propagadas entre os servidores através de um processo chamado anti-entropia, também conhecido como "*rumor mongering*", "*lazy propagation*" ou "*update dissemination*". A anti-entropia garante que cada escrita é eventualmente recebida por todos os servidores. Em outras palavras, para cara escrita w existe um tempo t tal que $w \in DB(S,t)$ para todo servidor S.

Adicionalmente, todos os servidores devem executar as escritas não comutativas na mesma ordem. Seja *WriteOrder*(W1,W2) um predicado booleano indicando que a escrita W1 deve ser ordenada antes da escrita W2. O sistema garante que se WriteOrder(W1,W2) então W1 é ordenado antes de W2 em DB(S) para cada servidor S que tenha recebido W1 e W2.

Vamos definir *RelevantWrites*(S,t,R) como uma função que retorna o menor conjunto de escritas suficiente para determinar o valor da leitura R.

Agora, podemos definir mais formalmente as garantias citadas acima.

RYW (Read Your Writes): Se uma leitura R segue uma escrita W em uma determinada sessão e R é executada em um servidor S no tempo t, então W ∈ DB(S,t).

MR (Monotonic Reads): Se R1 ocorre antes de R2 em uma determinada sessão, e R1 acessa o servidor S1 no tempo t1 e R2 acessa o servidor S2 no tempo t2, então RelevanteWrites(S1,t1,R1) é um subconjunto de DB(S2,t2).

WFR (Writes Follow Reads): Se uma leitura R1 precede uma escrita W2 em uma determinada sessão, e R1 é executada no servidor S1 no tempo t1, então, para qualquer servidor S2, se W2 ∈ DB(S2) então todo W1 ∈ *RelevantWrites*(S1,t1,R1) também pertence a DB(S2) e, além disso, *WriteOrder*(W1,W2) é verdadeiro.

MW (Monotonic Writes): Se uma escrita W1 precede uma escrita W2 em uma determinada sessão, então, para qualquer servidor S2, se W2 ∈ DB(S2) então W1 também pertence a DB(S2) e *WriteOrder*(W1,W2) é verdadeiro.

Para cada sessão são gerenciados dos conjuntos de WIDs:

read-set = conjunto dos WIDs das escritas que são relevantes para as leituras da sessão.

wrte-set = conjunto dos WIDs das escritas executadas (pertencentes) na sessão.

A implementação da garantia RYW envolve dois passos básicos. Quando uma escrita é aceita por um servidor, o WID assinalado pelo servidor à esta operação é adicionado ao conjunto *write-set* da sessão. Antes de uma leitura ser executa em um servidor S no tempo t, deve-se checar se *write-set* é um subconjunto de DB(S,t). Esta checagem é geralmente feita no servidor, embora possa ser feita no cliente, passando-se o *write-set* para o servidor. Caso o resultado da checagem seja positivo a leitura é executada, caso contrário o cliente deve procurar executar a operação em um outro servidor. Caso não seja encontrado nenhum servidor onde o resultado da avaliação é positivo, a garantia não pode ser atendida e este fato deve ser reportado à aplicação.

A implementação da garantia MR é feita de forma similar, antes de uma leitura ser executa em um servidor S no tempo t, deve-se checar se *read-set* é um subconjunto de DB(S,t). Adicionalmente, após a execução de uma leitura no servidor S, o WID de cada escrita em RelevantWrites(S,t,R) deve ser adicionado ao *read-set* da sessão. Presume-se que o servidor consiga computar as escritas relevantes e retornar essa informação juntamente com o valor do item lido.

A implementação das garantias WFR e MW requer que duas restrições adicionais sejam consideradas.

C1. Quando um servidor S aceita uma nova escrita W2 no tempo t, garante-se que *WriteOrder*(W1,W2) é verdadeiro para toda escrita W1 já presente em DB(S,t). Isto é, novas escritas são ordenadas após já recebidas anteriormente pelo servidor.

C2. A anti-entropia é executada de tal forma que se W2 é propagada do servidor S1 para o servidor S2 no tempo t, então todo W1 em DB(S1,t) tal que *WriteOrder*(W1,W2) já foi propagado para S2.

A implementação da garantia WFR pode ser conseguida da seguinte forma, cada leitura R executada no servidor S no tempo t faz com que *RelevantWrites*(S,t,R) seja adicionada no *read-set* da sessão. Antes de cada escrita ser executa no servidor S no tempo t, checa-se se *read-set* é um subconjunto de DB(S,t).

Implementar a garantia MW envolve dois passos. Para que um servidor S aceite uma escrita W no tempo t, DB(S,t) deve incluir o *write-set* da sessão. Além disso, quando a escrita é aceita pelo servidor, o seu WID é adicionado ao *write-set* da sessão.

A utilização de vetores de versão (*version vectors*) pode facilitar a implementação das garantias de sessão. Um vetor de versão é uma seqüência de pares <servidor, clock>, um para cada servidor. O primeiro elemento do par corresponde ao identificador de uma réplica (servidor) em particular. Já o segundo corresponde a um relógio lógico monotonicamente crescente gerenciado pelo servidor. O valor do relógio lógico é incrementado a cada escrita executada pelo servidor. Esse relógio lógico pode ser um relógio de Lamport ou simplesmente um contador. Um par <servidor, clock> é utilizado como o WID de uma escrita aceita por um servidor.

Cada servidor mantém o seu próprio vetor de versões, o qual mantém o seguinte invariante: se um servidor tem <S,c> no seu vetor de versões, então ele já recebeu todas as escritas cujo WIDs foram assinaladas pelo servidor S, ou seja, primeiramente aceitas pelo servidor S, antes do tempo lógico c no relógio de S. Assim, os servidores, durante o processo de anti-entropia, podem transferir escritas na ordem em que os WID foram assinalados. O vetor de versões de um determinado servidor é atualizado durante o processo de anti-entropia.

Para obter um vetor de versões que provê uma representação compacta para um conjunto de WIDs, Ws, fazemos V[S] = ao valor do maior WID assinalado pelo servidor S em Ws (ou 0 se nenhuma escrita de S foi recebida).

Para obter um vetor de versões V que representa a união de dois conjuntos de WIDs, Ws1 e Ws2, primeiro obtemos V1 de Ws1 e V2 de Ws2, como descrito acima. Em seguida, fazemos V[S] = MAX(V1[S],V2[S]) para todo S.

Para verificar se um conjunto de WIDs, Ws1, é um subconjunto de outro, Ws2, primeiro obtemos V1 de Ws1 e V2 de Ws2, como acima. Então, verificamos se V2 "domina" V1, onde dominar significa que um vetor é maior ou igual a outro em todas as posições.

Os servidores devem retornar um vetor de versões juntamente com o valor de uma leitura para indicar as escritas relevantes para a leitura efetuada. Neste caso, o vetor retornado é uma estimativa grosseira das escritas relevantes para uma consulta. Entretanto, esta estratégia simplifica a implementação das garantias.

A seguir mostramos um algoritmo para se obter as garantias de sessão.

```
Read(R,S) = {
        if MR then
                check S.vector dominates read-vector
        if RYW then
                check S.vector dominates write-vector
        [result, relevant-write-vector] := read R from S
        read-vector := MAX(read-vector, relevant-write-vector)
        return result
        }
Write(W,S) = {
        if WFR then
                check S.vector dominates write-vector
        wid := write W to S
        write-vector[S] := wid.clock
        }
```

4.5 Operações Desconectadas

O projeto Bayou não inclui a noção de operação desconectada, uma vez que vários níveis de conectividade são possíveis. Sendo necessário apenas que ocasionalmente a comunicação entre pares de servidores aconteça. Assim, o sistema trata a arbitrária conectividade da rede. Por exemplo, grupos de servidores podem se desconectar do restante do sistema e ainda continuarem conectados entre si.

5. CONCLUSÕES

A importância deste estudo se dá em razão de um número razoável de abordagens existentes para garantir a consistência e a disponibilidade de dados em ambientes de computação móvel. Assim, procuramos elaborar um survey a fim de oferecer um guia de referência para os interessados em desenvolver aplicações e pesquisas que envolvam a manipulação de dados em ambientes móveis.

Logicamente, não temos a pretensão de esgotar a discussão sobre a consistência de dados em computação móvel, ao contrário, desejamos despertar a atenção dos interessados no assunto, procurando abordar os pontos que julgamos importantes. Além disso, oferecemos uma relação bibliográfica bastante vasta, que inclui diversas soluções para as mais variadas arquiteturas encontradas em computação móvel.

Outras contribuições importantes deste trabalho foram a elaboração de uma taxonomia que classifica e organiza as principais propostas para a consistência de dados em ambientes de computação móvel, e a descrição detalhada do sistema Bayou.

6. REFERÊNCIAS BIBLIGRÁFICAS

[1] Walborn G., e Chrysanthis P. **Supporting Semantics-Based Transaction Processing in Mobile Database Applications.** 1995.

[2] Gray J., Helland P., O'Neil P., e Shasha D. **The Dangers of Replication and a Solution.** 1996.

[3] Barbará-Millá D., e Garcia-Molina H. **Replicated Data Management in Mobile Environments: Anything new under the sun?** 1994.

[4] Ionitou C., e Andersen B. **Replicated Objects with Lazy Consistency.** 1995.

[5] Lu Q., e Satyanaranyanan M. **Isolation-Only Transactions for Mobile Computing.** 1994.

[6] Huang Y., Sistla P., e Wolfson O. **Data Replication for Mobile Computers.** 1994.

[7] Kanitkar V., e Delis A. **Two-Stage Transaction Processing in Client-Server DBMSs.** 1998.

[8] Yee W., Donahoo M., Omiecinski E., e Navathe S. **Scaling Replica Maintenance in Intermittently Synchronized Mobile Database**. 2001.

[9] B. R. Badrinath and T. Imielinski. **Replication and Mobility.** 1992.

[10] Lu Q., e Satyanaranyanan M. **Improving Data Consistency in Mobile Computing Using Isolation-Only transaction.**

[11] Pitoura E., e Bhargava B. **Data Consistency in Intermittently Connected Distributed Systems.** 1999.

[12] Loh y., Hara T., Tsukamoto M, e Nishio S. **A Hybrid Method for Concurrent Updates on Disconnected Databases in Mobile Computing Environments.** 2000.

[13] Cao H. **Adaptive Power-Aware Cache Management for Mobile Computing Systems.** 2002.

[14] Lin H., Chen C., e Zhou B. **A Probability-Based Approach of Transaction Consistency in Mobile Environments.** 2001.

[15] Barbará D., e Imielínski T. **Sleepers and Workaholics: caching Strategies in Mobile Environments. 1994**.

[16] Su C., e Tassiulas L. **Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery.** 1998.

[17] Jing J., e Elmargarmid A. **Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments.** 1997.

[18] Franklin M., Carey M., e Livny M. **Transactional Client-Server Cache Consistency: Alternatives and Performance.** 1997.

[19] Mazumdar S., Pietrzyk M., e Chrysanthis P. **Caching Constrained Mobile Data.** 2001

[20] Machado J. **Um Mecanismo de Gerência de Cache para um Servidor de Dados Móveis em Redes Ad Hoc.** 2004.

[21] Lee S., Hwang C., e Yu H. **Supporting Transactional Cache Consistency in Mobile Database Systems.** 1999.

[22] Yuen J., Chan E., Lam K., e Leung H. **Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data.** 2000.

[23] Berkenbrok C., Dantas M., Berkenbrok G. e Filho P. **Caching Strategies Using Propagation and Invalidation Update in Mobile Computing Environments**.

[24] Phatak S., e Badrinath. **Transaction-Centric Reconciliation in Disconnected Database.** 2004.

[25] Dirckze R., e Gruenwald L. **A Pré-Serialization Transaction Management Techniques for Mobile Multidatabases.** 2000.

[26] Dunham M., Helal A., e Balakrishnan S. **A Mobile Transaction Model that Captures Both the Data and Movement Behavior.** 1997.

[27] Weijun X., Zhengding L., Bing L. e Sarem M. **Transaction Management in Mobile Multidatabase Systems.** 2001.

[28] Dirckze P., e Gruenwald L. **A Toggle Transaction Management Technique for Mobile Multidatabase.** 1998.

[29] Pitoura E., e Bhargava B. **A Framework for Providing Consistency and Recoverable Agent-Based Access to Heterogeneous Mobile Databases.** 1995.

[30] Faiz M., e Zaslavsky A. **Database Replica Management Strategies in Multidatabase Systems with Mobile Hosts.** 1995.

[31] Lim J., e Hurson A. **Transaction Processing in Mobile, Heterogeneous Database System.** 2002.

[32] Holliday J., Agrawal D., e Abbadi A. **Planned Disconnections for Mobile Databases.** 2000.

[33] Holliday J., Agrawal D., e Abbadi A. **Disconnected Modes for Mobile Database.** 2002.

[34] Hara T. **Replica Allocation Methods in Ad Hoc Networks with Data Update.** 2003.

[35] Frank S., e Brayner A. **SESAMO: Uma Estratégia de Controle de Concorrência em Sistemas Múltiplos de Bancos de Dados.** 2004.

[36] Alonso R., Bárbara D., e Garcia-Molina H. **Data Caching Issues in a Information Retrieval System.** 1990.

[37] Luo J., Hubaux J., e Eugster P. **PAN: Providing Reliable Storage in Móbile Ad Hoc Networks with Probabilistic Quorum Systems**. 2003.

[38] Hou W., Su M., Zhang H., e Wang H. **An Optimal Construction of Invalidation Reports for Móbile Databases.** 2001.

[39] Barbará D. **Mobile Computing and Databases – A Survey.** 1998.

[40] Terry D., Demers A., Petersen K., Spreitzer M., Theimer M. e Welch B. **Session Guarantees for Weakly Consistent Replicated Data.** 1994.

[41] Terry D., Demers A., Petersen K., Spreitzer M., Theimer M., Welch B. e Hauser C. **Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. 1995.**

[42] Terry D., Demers A., Petersen K., Spreitzer M., Theimer M. e Welch B. **The Bayou Architecture: Support for Data Sharing among Mobile Users.** 1994.

[43] Terry D., Demers A., Petersen K., Spreitzer M. e Theimer M. **Flexible Update Propagation for Weakly Consistent Replication.** 1997.

[44] Rodrig M. e Lamarca A. **Decentralized Weighted Voting for P2P Data Management.** 2003.

[45] Cetintemel U., Keleher P. e Franklin M. **Support for Speculative Update Propagation and Mobility in Deno.** 1999.

[46] Brayner A. **Transaction Management in Multidatabase System.** 1999.

[47] Brayner A. e Häder T. **Global Semantic Serializability: Na Approach to Increase Concurrency Control in Multidatabase Systems.** 2001

[48] Brayner A., Häder T. e Ritter N. **Semantic Serializability: A Correctness Criterion for Processing Transactions in Advanced Database Applications.** 1999.

[49] Brayner A. e Monteiro J. **Temporal Serialization Graph Testing: An Approach to Control Concurrency in Broadcast Environments.** 2000.

[50] Mateus G. e Loureiro A. **Introdução à Computação Móvel.** 1998.

[51] Pitoura E. e Chrysanthis P. **Scalable Processing of Read-Only Transactions in Broadcast Push.** 1999.

[52] Shanmugasundaram J., Nithrakashyap A., Sivasankaran R. e Ramamritham K. **Efficient Concurrent Control for Broadcast Environments.** 1999.

[53] Sheng S., Chandrasekaran A. e Broderson R. **A Portable Multimedia Terminal for Personal Communications.** 1992

[54] J. Jing, A. Helal and A. Elmagarmid. **Client-Server Computing in Mobile Environments.** 1999.

[55] J. P. Macker and M. S. Corson. **Mobile Ad Hoc Networking and the IETF", *Internet Engineering Task Force MANET Working Group***. Available online at http://www.ietf.org/html.charter/manet-charter.html.

[56] Brayner A. e Aguiar L. **Increasing Mobile Transaction Concurrency in Móbile Database Communities.** 2003.

[57] Brayner A. e Aguiar J. **Sharing Mobile Database in Dynamically Configurable Environments.** 2003.

[58] Herman G. **The Datacycle Architecture for Very Hight Throughput Database Systems.** 1987.

[59] S. Acharya, R. Alonso, M. Franklin, e S. Zdonik. **Broadcast Disks: Data Management for Asymmetric Communications Environments.** 1995.

[60] S. Acharya, M. Franklin, e S. Zdonik. **Disseminating Updates on Broadcast Disks.** 1996.

[61] M. A. Casanova. **The Concurrency Problem of Database Systems.** 1981.

[62] A. Datta, D. E. Vandermeer, A. Celik, e V. Kumar. **Broadcast Protocols to Support Eficcient Retrieval from Databases by Mobile Users.** 1999.

[63] T. Imielinski and B. R. Badrinath. **Mobile Wireless Computing: Challenges in Data Management.** 1994.

[64] B. Oki. **The Information Bus – A Architecture for Extensible Distributed Systems.** 1993.

[65] S. Shekar and D. Liu. **Genesis and Advanced Traler Informations Systems (ATIS): Killer Applications for Mobile Computing.** 1994.