PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 21/06

Um Mecanismo para a Consistência de Dados Replicados em Computação Móvel

José Maria Monteiro

Markus Endler

Sérgio Lifschitz

Ângelo Brayner

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 21/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Julho, 2006

Um Mecanismo para a Consistência de Dados Replicados em Computação Móvel *

José Maria Monteiro, Markus Endler, Sérgio Lifschitz, Ângelo Brayner[1]

[1]Mestrado em Informática Aplicada – Universidade de Fortaleza (UNIFOR)

monteiro@inf.puc-rio.br, endler@inf.puc-rio.br, sergio@inf.puc-rio.br, brayner@unifor.br

Abstract. The recent advances in portable computer together with the developments in wireless communication technology are allowing users of portable equipment to maintain the network connection while they move about freely, having access to shared resources, services and information. This paradigm is called mobile computing. Mobile computing allows for the development of new and sophisticated database applications. Such applications require the reading of current and consistent data. In order to improve the data availability, increases performance and maximize throughput, data replication is used. However, due to inherent limitations in mobile environments, where there is typically great limitations for communications between mobile equipment and prolonged lack of connection between these pieces of equipment, making necessary changes in the concurrency control and replica control mechanisms be implemented. This paper proposes a new protocol that guarantees the consistency of replicated data in a mobile computing environment.

Keywords: Concurrency Control, Mobile Computing, Data Replicated, Mobile Transaction, Context Based Services.

Resumo. As recentes evoluções ocorridas nos computadores portáteis em conjunto com os avanços nas tecnologias de comunicação sem fio estão 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. Para melhorar a disponibilidade dos dados, obter ganhos de performance e maximizar o throughput, a replicação de dados é utilizada. Entretanto, devido às limitações inerentes aos ambientes móveis, 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, torna-se necessário que mudanças nos mecanismos de controle de concorrência e controle de réplicas sejam implementadas. Este trabalho propõe um novo protocolo que garante a consistência de bancos de dados replicados em ambientes de computação móvel.

Palavras-chave: Controle de Concorrência, Computação Móvel, Dados Replicados, Transações Móveis, Serviços Baseados em Contexto.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

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) [4].

A computação móvel possibilita o desenvolvimento de novas e sofisticadas aplicações em banco de dados. Aplicações de comércio eletrônico, tais como leilões, reserva de ingressos, pedido e compra de produtos; sistemas de gerenciamento de tráfego, editoração compartilhada de documentos [4], são exemplos de aplicações de bancos de dados que requerem suporte da tecnologia de computação móvel. Logicamente, estas aplicações necessitam acessar dados atuais e consistentes, apesar do acesso concorrente aos dados compartilhados.

Para melhorar a disponibilidade dos dados, obter ganhos de performance e maximizar o *throughput*, a replicação de dados é utilizada. A idéia básica consiste em armazenar múltiplas cópias dos itens de dados em diferentes servidores, distribuídos através de uma rede de comunicação. Assim, as aplicações podem acessar os dados de qualquer uma das réplicas, podendo inclusive continuar sua execução, mesmo que alguns dos servidores falhem. Entretanto, estes benefícios trazem a necessidade de atualizar todas as cópias de um item quando este for alterado, ou seja, torna-se necessário manter a coerência entre os itens replicados nos diversos servidores. A replicação de dados não é uma idéia nova, mas sua utilização tem, recentemente, crescido bastante, devido a massificação dos ambientes e das tecnologias de computação móvel. Esta técnica tem se mostrado de extrema importância tanto em redes móveis quanto em redes fracamente conectadas, pois possibilita o compartilhamento eficiente dos dados apesar 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 [7].

Um banco de dados replicado móvel é um banco de dados distribuído através de uma rede sem fio, no qual múltiplas cópias dos itens de dados são armazenadas em diferentes servidores. Aplicações que executam em um ambiente de banco de dados replicado móvel necessitam acessar dados atuais e consistentes. Desta forma, o SGBD deve garantir a consistência dos dados, ou seja, 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. Além disso, o SGBD necessita assegurar que o estado das diversas réplicas irão eventualmente convergir para um único estado final consistente, apesar das operações terem sido executadas nas diferentes cópias de forma independente, se os usuários pararem de submeter novas operações, o que é denominado de consistência eventual. Nos tradicionais SGBD's replicados este controle é realizado através de um protocolo de controle de réplica (*replica control protocol*), os quais podem ser classificados em três grupos: *Primary-Copy Methods* (também denominado de *pessimistic replication*), *Quorum-Consensus Methods* e *Available-Copies Methods* (também conhecido como *optmistic replication*) [6]. Entretanto, estes protocolos têm se mostrado impróprios para ambientes de computação móvel [7].

Com o objetivo de solucionar eficientemente o problema da consistência de dados replicados, em ambientes de computação móvel, algumas propostas têm tentado relaxar o conceito de seriabilidade, buscando formas mais fracas de garantir a qualidade dos dados acessados pelos clientes. Tais métodos têm sido chamados de *weaker replica consistency*. Porém, a maioria dessas propostas requer intensa troca de mensagens [8,9,12], ou ainda, que as operações que compões as transações, e que são armazenadas em *log*, sejam re-executadas diversas vezes. Por exemplo, em [1] 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. Além disso, para permitir um meio de adaptabilidade, algumas propostas possibilitam que o grau de consistência seja adaptado aos vários níveis de conectividade. Em [1] isto é realizado selecionando-se individualmente que garantias de sessão deseja-se utilizar. Entretanto, tais garantias não possuem uma semântica clara, do ponto de vista da consistência de dados, ou seja, não é estabelecida nenhuma relação entre estas garantias e os níveis de isolamento padronizados pela ANSI/ISO SQL 92 [3], e nem são especificados que fenômenos são evitados em cada uma das garantias de sessão.

Este trabalho propõe um novo mecanismo que garante a consistência de dados replicados (*one-copy serializability*) e a convergência das diversas réplicas para um único estado final consistente (consistência eventual), em ambientes de computação móvel. Esse protocolo explora informações temporais referentes ao momento em que um objeto do banco de dados replicado foi lido ou atualizado. No protocolo proposto, as operações que compõem as diversas transações são executadas uma única vez. Os servidores não necessitam armazenar um *log* com todas as operações realizadas. O mecanismo utiliza um critério de correção já consolidado, a seriabilidade, e relaxa a propriedade de isolamento, permitindo uma maior disponibilidade dos dados. Além disso, nossa abordagem reduz o numero de mensagens trocadas entre os servidores. Dessa forma, nosso protocolo procura garantir uma economia no custo da utilização dos canais de comunicação, na limitada capacidade de energia dos computadores portáteis e nos seus escassos recursos de memória.

Este artigo está organizado da seguinte forma: a seção 2 discute as características dos ambientes de computação móvel. A seção 3 caracteriza um ambiente de computação móvel com servidores de bancos de dados replicados. A seção 4 descreve o modelo transacional que utilizaremos neste trabalho. Na seção 5 será descrito e analisado o protocolo proposto para controle da consistência. Na seção 6 serão discutidos os trabalhos relacionados. A seção 7 conclui este trabalho e aponta direções para trabalhos futuros.

2 Ambientes de Computação Móvel

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 [13]. 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.

Nesta seção, descrevemos e analisamos as principais propriedades e características de um ambiente de computação móvel. Primeiramente apresentaremos uma arquitetura referência para um ambiente de computação móvel, em seguida discutiremos as características, limitações e problemas referentes a estes ambientes, e, finalmente, discutiremos os benefícios e os desafios da utilização de bancos de dados replicados.

2.1 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) [13].

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 [13].

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.

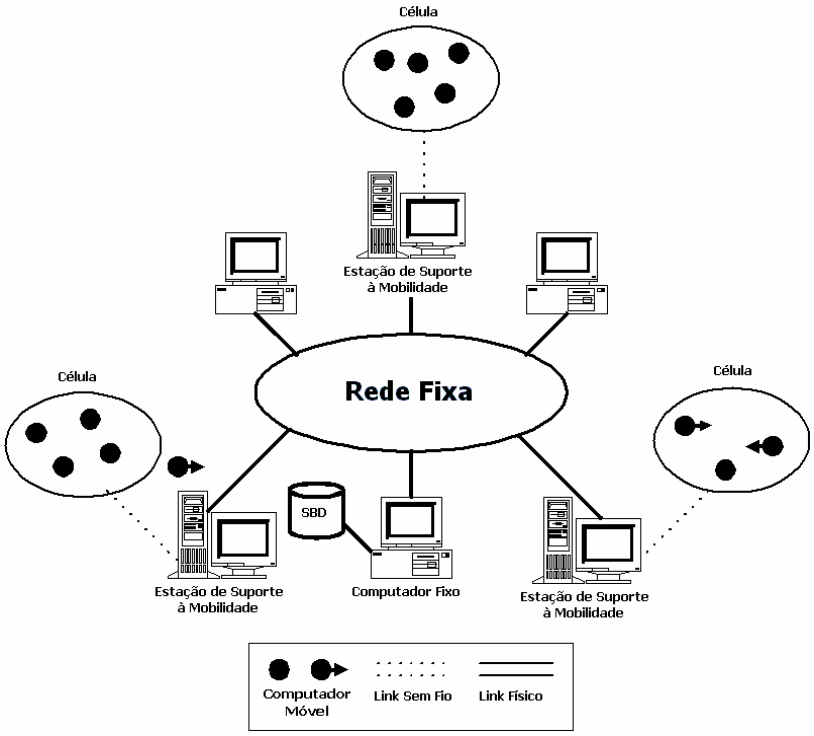


Figura 2.1. Arquitetura Clássica para um Ambiente de Computação Móvel

2.2 Características de um Ambiente de Computação Móvel

Como afirmado anteriormente, um ambiente de computação móvel apresenta características bastante particulares. Dentre elas, destacam-se [13, 14]:

(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 [7]. 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*).

2.3 Replicação de Dados em Computação Móvel

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 2.2). A replicação de dados traz as seguintes vantagens [6]:

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.

Os tradicionais algoritmos para replicação pessimista, como por exemplo, o popular algoritmo *primary-copy*, fornecem uma semântica conhecida como *single-copy consistency*, na qual os usuários têm a ilusão de estarem utilizando uma única, e altamente disponível, cópia do banco de dados. Em geral, estes algoritmos bloqueiam o acesso aos dados replicados desatualizados até que estes estejam atualizados. Estas técnicas funcionam relativamente bem em redes locais, onde as latências são pequenas e as falhas não muito comuns. Mas, não apresentam boa performance em redes sem fio devido às altas latências da rede, a intermitência da conexão dos dispositivos móveis e à baixa escalabilidade. Já os mecanismos baseados em quorum (*Quorum-Consensus Methods*) requerem elevado tráfego de mensagens, devido aos processos de votação. Os algoritmos de replicação otimista, por sua vez, permitem que o usuário acesse dados desatualizados, de uma forma controlada, aumentando a disponibilidade dos dados, o que é particularmente importante para ambientes sem fio. Porém, os testes para validação e reconciliação requerem elevado tráfego de mensagens, o que compromete a eficiência destes métodos.

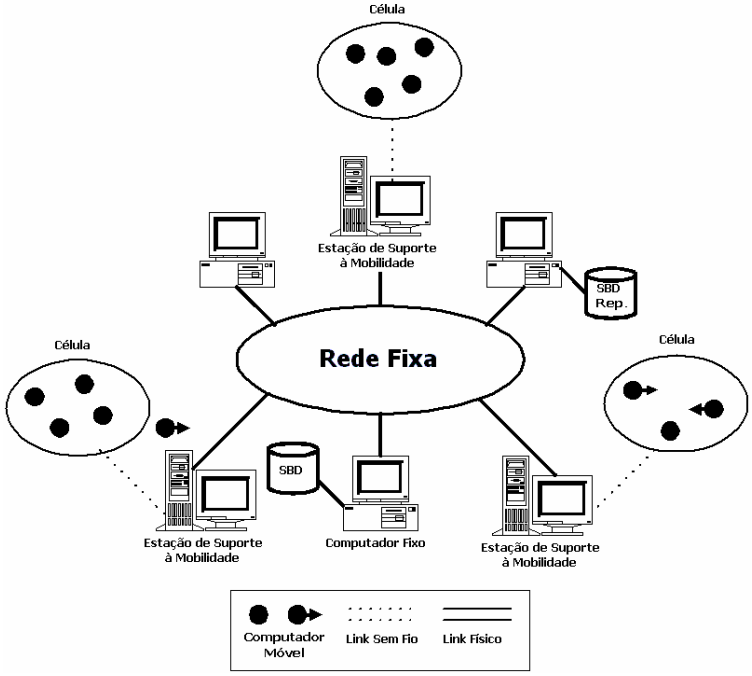


Figura 2.2.a. Uma possível arquitetura de CM com servidores replicados.

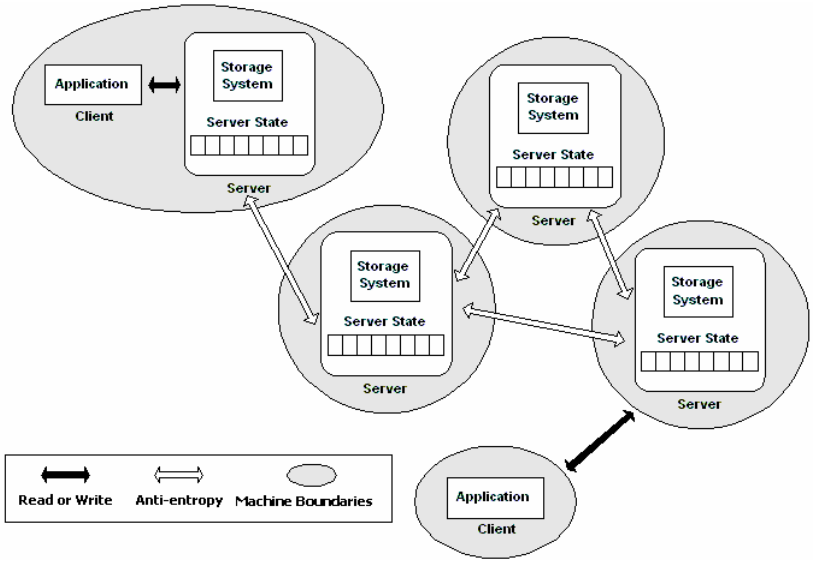


Figura 2.2.b. Uma possível arquitetura de CM com servidores replicados em ambientes parcialmente conectados (fraca conectividade).

3 Ambiente de Computação Móvel com Replicação de Dados

Um banco de dados replicado móvel é um banco de dados distribuído através de uma rede sem fio, no qual múltiplas cópias dos itens de dados são armazenadas em diferentes servidores. Os servidores permitem acesso (leitura e escrita) aos dados replicados mesmo quando estão desconectados do restante do sistema de comunicação. As aplicações podem acessar os dados de qualquer uma das réplicas, podendo inclusive continuar sua execução, mesmo que alguns dos servidores falhem. Desta forma, pode-se obter ganhos de performance, aumentar a disponibilidade dos dados e maximizar a vazão (*throughput*). Tais propriedades são particularmente importantes para ambientes de comunicação sem fio, onde os clientes móveis podem se desconectar por longos períodos de tempo, fazendo com que as transações que executam nestes dispositivos sejam de longa duração.

Entretanto, estes benefícios trazem a necessidade de atualizar todas as cópias de um item quando este for alterado, ou seja, torna-se necessário manter a coerência entre os itens replicados nos diversos servidores. Conflitos entre atualizações executadas em diferentes réplicas poderão ocorrer, o que poderá conduzir um mesmo a objeto a estados distintos, em um mesmo instante de tempo. Logicamente, a divergência entre os estados das diferentes réplicas, deve ser evitada ou minimizada. Para garantir a consistência dos dados, mediante o acesso concorrente nas diversas réplicas, e a eventual convergência das réplicas para um mesmo estado final consistente (consistência eventual), um protocolo de controle de replica deve ser utilizado.

4 Modelo Transacional

Um banco de dados consiste de uma coleção de objetos que representam entidades do mundo real. O conjunto de valores de todos os objetos armazenados em um banco de dados num dado instante de tempo é chamado estado do banco de dados. O mundo real impõe certas restrições sobre as suas entidades. Logo, os bancos de dados devem capturar tais restrições, chamadas restrições de consistência. Se os objetos do banco de dados, em um dado instante, satisfazem todas as restrições de consistência dizemos que o estado do banco de dados é consistente. Os valores destes objetos podem ser lidos e modificados pelas transações. Uma transação é uma abstração que representa uma seqüência finita de operações de leitura e escrita sobre os objetos do banco de dados. A notação $r_i(x)$ e $w_i(x)$ será utilizada para representar operações de leitura e escrita executadas por uma transação T_i sobre um objeto x. Através de $OP(T_i)$ será representado o conjunto de todas as operações executadas pela transação T_i. Vamos assumir que a execução de uma transação preserva a consistência do banco de dados, caso essa transação seja executada por completo e de forma isolada das demais transações.

A execução concorrente de um conjunto de transações é realizada através do entrelaçamento das operações que compõem as várias transações. Esse entrelaçamento pode levar o banco de dados a um estado inconsistente. Logo, torna-se necessário definir quando uma execução concorrente de transações está corretamente entrelaçada, produzindo, assim, um estado consistente do banco de dados. Daqui em diante chamaremos uma execução corretamente entrelaçada de execução correta. Uma história ou *schedule* global indica a ordem na qual as operações de um conjunto de transações são executadas com relação umas com as outras. Duas operações de diferentes transações conflitam (ou estão em conflito) se e somente se elas acessam o mesmo objeto do banco de dados e pelo menos uma delas é uma operação de escrita. A notação $p <_S q$ indica que a operação p foi executada antes da operação q no *schedule* global S.

Seja S um *schedule* global sobre um conjunto de transações $T=\{T_1, T_2, ... , T_n\}$. O grafo de serialização para S, representado por GS(S), é definido como o grafo direcionado GS(S) = (N,A) no qual cada nó em N corresponde a uma transação em T. O conjunto A contém as arestas na forma $T_i \rightarrow T_j$, se e somente se $T_i, T_j \in N$ e existem duas operações $p \in OP(T_i)$, $q \in OP(T_j)$, onde p conflita com q e $p <_S q$. Um *schedule* global S é serializável por conflito se e somente se o grafo de serialização para S é acíclico. Um *schedule* S é correto se ele for serial ou serializável por conflito [5].

Um banco de dados replicado móvel é um banco de dados distribuído através de uma rede sem fio, no qual múltiplas cópias dos itens de dados são armazenadas em diferentes servidores. A menor unidade de replicação é denominada de objeto. Uma réplica é uma cópia de um objeto, armazenada em um determinado *host* (servidor). Um servidor pode armazenar réplicas de múltiplos objetos, porém, as vezes utilizamos o termo réplica como sinônimo de servidor (*host*). Os servidores (*hosts*) que podem atualizar suas réplicas são denominados de servidores *master* (*master sites*), para diferenciar daqueles cujas replicas (dados replicados) são somente de leitura. O símbolo "N" é utilizado para denotar o número total de réplicas e "M" para denotar o número de réplicas *master*, para um dado objeto. Comumente temos: M=1 (*single-master systems*) e M=N (*multi-master systems*). O mecanismo proposto neste trabalho utiliza o esquema *multi-master*, ou seja, *ready-any/ write-any*. Os servidores permitem o acesso (leitura e escrita) aos dados replicados mesmo quando estão desconectados.

Para atualizar um determinado objeto, uma aplicação cliente submete uma operação (escrita) a um dos servidores replicados. Cada servidor replicado utiliza o nível de isolamento *read uncommitted*. Ao receber uma operação de escrita, o servidor executa a operação em sua cópia local, e o cliente continua sua execução normalmente, considerando a atualização como realizada. As atualizações realizadas em um determinado servidor replicado não são imediatamente enviadas para os demais servidores. Tais atualizações são consideradas uma tentativa (*tentative write*). Conflitos entre atualizações executadas nas diferentes réplicas poderão ocorrer, o que poderá conduzir um mesmo objeto a estados distintos, em um mesmo instante de tempo. Para garantir a consistência dos dados, mediante o acesso concorrente nas diversas réplicas, e a eventual convergência das réplicas para um mesmo estado final consistente (consistência eventual), um servidor primário (*primary copy*) é utilizado. Uma forma de tratar este problema é detectar as operações que estão em conflito e resolvê-los, abortando uma das transações envolvidas, por exemplo. Desta forma, os servidores replicados necessitam comunicar, em background, as operações realizadas em suas cópias locais, para o servidor primário. O termo *commitment* refere-se ao algoritmo utilizado para assegurar a convergência das réplicas para um mesmo estado comum.

Neste trabalho, assumimos que o ambiente de computação móvel a ser considerado consiste em um sistema assíncrono, no qual os clientes móveis apresentam uma conexão intermitente e os servidores podem, eventualmente, falhar. Partições na rede podem ocorrer, restringindo a conectividade entre os servidores localizados em diferentes partições. Neste caso, somente os servidores na mesma partição do servidor primário poderiam efetivar suas transações. As transações dos demais servidores seriam consideradas transações de tentativa, sendo sincronizadas quando a conexão com o servidor primário for re-estabelecida. Considera-se que o servidor primário pode, eventualmente, falhar. Assim, um algoritmo de eleição deve ser utilizado para a escolha de um novo servidor primário. Entretanto, assumimos que o conjunto dos servidores funciona como um serviço de um *middleware* distribuído, onde, mesmo que falhas eventuais ocorram, a recuperação se dará com uma performance razoável.

5 Um Novo Mecanismo para a Consistência de Dados Replicados

Nesta seção, nós iremos definir um protocolo de controle de concorrência que garante o acesso a dados consistentes e atuais por parte de aplicações em ambientes de computação móvel com bancos de dados replicados. Em seguida, ilustraremos, a partir de um exemplo, o funcionamento do mecanismo proposto.

5.1. Princípios Gerais

O protocolo proposto foi projetado para ambientes de computação móvel formados por um conjunto de servidores de bancos de dados replicados e por dispositivos portáteis fracamente conectados. Na arquitetura em questão, um cliente e um servidor podem co-existir em um determinado *host* (fixo ou móvel), ou podem executar de forma isolada. Nestes ambientes os canais de comunicação são instáveis e nem sempre um servidor está disponível ou alcançável. Assim, tanto os servidores quanto os dispositivos móveis podem ser visualizados como um ponto em um modelo ponto a ponto (*peer-to-peer*), onde servidores e clientes estão conectados de forma intermitente uns com os outros.

Para melhorar a disponibilidade dos dados, obter ganhos de performance e maximizar a vazão, a replicação de dados é utilizada. Assim, cada banco de dados é inteiramente replicado em um conjunto de servidores. Desta forma, as aplicações não necessitam se restringir a utilizar um único servidor, mas podem interagir com qualquer servidor com o qual consigam se comunicar, ou seja, os clientes podem executar tanto leituras quanto escritas em qualquer um dos servidores, com os quais possa estabelecer uma comunicação. Para isso, é utilizado um esquema de replicação *read-any/write-any*. Este esquema proporciona um meio de adaptabilidade, uma vez que o cliente pode, dependendo de sua conectividade, utilizar, por exemplo, o servidor que estiver mais próximo (economizando energia). 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, em um servidor A, e posteriormente ao executar esta operação novamente, em um servidor B, pode obter um valor anterior ao valor lido inicialmente. Outro problema sério ocorre mesmo quando um único usuário ou aplicação está atualizando os dados. Suponha que um usuário móvel executou uma operação de escrita em um servidor A, e mais tarde tentou ler o mesmo item de um outro servidor B, caso o ultimo servidor ainda não tenha sido informado da atualização ocorrida em A, o cliente irá obter um valor inconsistente. Para evitar ou reduzir as inconsistências observadas pelos clientes quando acessam diferentes servidores, um protocolo de controle de concorrência e de controle de replicas deve ser utilizado.

Réplicas de diferentes servidores podem apresentar conteúdos distintos. Isto é, $DB(S_1,t)$, ou seja o estado do banco de dados no servidor replicado S_1 no instante de tempo t, não é necessariamente equivalente a $DB(S_2,t)$ para quaisquer dois servidores S_1 e S_2. Entretanto, na prática, os sistemas usualmente desejam atingir uma consistência eventual na qual os servidores convergem para uma cópia idêntica.

A fim de aumentar a taxa com que as transações são efetivadas (*throughput*) e diminuir o tempo necessário para a convergência das diversas replicas, uma adaptação do esquema denominado *primary-commit*, proposto em [1], é utilizada. Assim, um servidor designado como primário tem a responsabilidade de sincronizar e executar o *commit* das atualizações, ou seja, torná-las permanente. As escritas já consolidadas (*committed writes*) geram novas versões dos itens de dados, que devem ser propagadas para os demais servidores.

O protocolo não inclui a noção de operação desconectada (onde o cliente acessa os dados locais e, quando a conexão é re-estabelecida, executa um procedimento de reintegração ou sincronização) uma vez que vários níveis de conectividade são possíveis. Sendo necessário apenas que ocasionalmente a comunicação entre cada servidor e o servidor primário (*primary-commit*), ou provedor de serviço, aconteça. Assim, o sistema trata a conectividade intermitente da rede. Por exemplo, um determinado servidor pode se desconectar do restante do sistema e ainda assim continuar sendo utilizado pelos clientes.

O protocolo proposto, é baseado em uma estratégia similar a utilizada pelo mecanismo de teste do grafo de serialização temporal proposto em [5]: o monitoramento e gerenciamento dinâmico de um grafo que deve ser sempre acíclico. Adicionalmente, este protocolo explora informações temporais referentes ao momento em que um item de dado foi lido ou atualizado.

No protocolo proposto, as funções para o controle de concorrência estão distribuídas entre os clientes, servidores e o servidor primário (provedor do serviço de consistência). Portanto, assumimos que o servidor primário, os demais servidores (réplicas) e os clientes apresentam funcionalidades específicas para o gerenciamento das transações. A seguir descreveremos estas funcionalidades.

Após o *commit* de uma determinada transação T_i, o servidor primário propaga os novos valores dos itens de dados atualizados por T_i, juntamente com os marcadores de tempo correspondentes (numero de versão), para todas as replicas com as quais tiver comunicação. Estes marcadores são definidos da seguinte forma:

- Cada item de dado (x) em $DB(S_i)$, onde $0<i\leq N$, é associado a um marcador de tempo (*timestamp*) C(x);
- Cada operação (escrita ou leitura) de uma transação T_i, $P_i(x)$, também está associada a um marcador de tempo (*timestamp*) representado por $C(P_i(x))$;
- Um marcador de tempo consiste em um par ordenado (z,y). A primeira parte (z) será denominada versão, já a segunda parte (y) será chamada de sub-versão;
- Inicialmente $C(x) = (0,0)\ \forall\ x \in BD$, em toda réplica R_j, onde $0<j\leq N$;
- A versão de C(x), ou seja, z, é incrementada a cada *commit* de uma transação qualquer T_i que executa uma operação de escrita sobre x (nova versão de x);
- A cada *commit* de uma transação que produz uma nova versão de x, com $C(x) = (v,0)$, onde $v>0$, esta nova versão, juntamente com o seu novo marcador de tempo, é enviada pelo servidor primário para todas as réplicas R_j (com as quais tiver comunicação);
- Diferentes estratégias podem ser utilizadas para o envio dos novos valores dos itens de dados e das novas versões, do servidor primário (copia principal) para as demais réplicas, possibilitando um meio de adaptabilidade, onde as informações de contexto, como por exemplo: qualidade do sinal, nível de energia e memória disponível, podem ser consideradas. Estas estratégias serão discutidas posteriormente;
- A sub-versão de C(x), ou seja, y, é incrementada a cada escrita de uma transação ativa (não efetivada, ou seja, *uncommitted*) qualquer T_i, que escreveu em x, em qualquer réplica R_j;

A cada operação de leitura $r_i(x) \in T_i$ (que pode ser executada em qualquer uma das réplicas R_j), o servidor replicado utilizado (R_j) guarda o valor e o identificador do item lido, juntamente com o marcador de tempo correspondente, $C(r_i(x))$, o qual corresponde ao valor corrente do marcador de tempo do item lido, x, na sua cópia local, ou seja, $C(x)_{Rj}$, e associa estes valores à operação executada $r_i(x)$.

A cada operação de escrita $w_i(x) \in T_i$ (que pode ser executada em qualquer uma das réplicas R_j), o servidor replicado utilizado (R_j) incrementa o valor da sub-versão (y) do marca-

dor de tempo do item lido (x), em sua cópia local (R_j). Em seguida, o servidor R_j, guarda o valor e o identificador do item atualizado, juntamente com o marcador de tempo correspondente, $C(w_i(x))$, o qual corresponde ao valor corrente do marcador de tempo do item lido, x, na sua cópia local, ou seja, $C(x)_{Rj}$, e associa estes valores à operação efetuada ($w_i(x)$).

Periodicamente, cada servidor replicado (R_j) deve enviar um pacote (mensagem) ao servidor primário contendo as operações de leitura e escrita executadas até o momento em sua cópia local, juntamente com os seus respectivos marcadores de tempo, além do seu próprio identificador (R_j). As operações já informadas não precisam ser enviadas novamente. Estas informações são recebidas por um escalonador (*scheduler*), no servidor primário, e são utilizadas para sincronizar as operações das diversas transações, gerando um entrelaçamento correto, ou seja, que preserva (segundo algum critério de correção) a consistência dos dados replicados.

Quando um cliente recebe um pedido de *commit* ou *abort* para uma transação móvel T_i, ele deve submeter este pedido a qualquer um dos servidores replicados (R_j), o qual irá repassar esta requisição ao servidor primário. No caso da operação de *commit* deve ser enviando também o número de operações da transação T_i. Adicionalmente o cliente C_i pode enviar a mensagem contendo a solicitação de *commit* de uma transação T_i diretamente para o servidor primário. O cliente deve esperar a resposta para efetuar a operação de *commit* ou de *abort*. Caso esgote-se um *timeout* e o cliente não receba a resposta para o seu pedido de *commit*, algumas alternativas podem ser escolhidas: realizar o *abort* da transação T_i ou definir T_i como uma transação de tentativa, ou seja, uma transação concluída cujas escritas podem não ser efetivadas. Adicionalmente, se a transação Ti é uma transação somente de leitura (*read only*), o cliente pode-se decidir pela conclusão da mesma, ficando ciente de que esta pode ter lido valores inconsistentes. Com esta finalidade, informações de contexto podem ser utilizadas como um meio de adaptabilidade. Por exemplo, se o nível de energia está baixo e a qualidade do sinal não é boa, a desconexão é eminente. Neste caso, o cliente pode optar por concluir sua transação como uma tentativa, ou seja, tendo o conhecimento de que suas escritas podem não ser efetivadas, ou seja, podem ser desfeitas, e que suas leituras podem ter acessado valores inconsistentes (temporários). Caso contrário, o cliente pode esperar um pouco mais, pois, provavelmente, a conexão com o servidor R_j, ou mesmo com o servidor primário, será re-estabelecida.

A abordagem proposta para assegurar a consistência dos dados replicados baseia-se na comparação dos marcadores de tempo associados a cada operação de leitura e escrita, pertencentes às diversas transações, e que são executadas sobre diferentes réplicas. A comparação entre dois marcadores de tempo $C(q_j(x))$ e $C(p_i(x))$ é realizada da seguinte forma: Suponha, sem perda de generalidade, que $C(q_j(x)) = a.b$ e $C(p_i(x)) = c.d$. Se $a > c$ então $C(q_j(x)) > C(p_i(x))$. Agora, se $a = c$, deve-se verificar o valor da sub-versão (segunda parte do valor referente ao marcador de tempo). Desta forma, se $a = c$ e $b > d$, então, $C(q_j(x)) > C(p_i(x))$. No caso em que $a = c$ e $b = d$, $C(q_j(x)) = C(p_i(x))$.

5.2 Protocolo de Controle de Réplica

Um escalonador (*scheduler*) implementando o protocolo proposto 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, da seguinte forma:

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

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

Senão

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

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

Senão // $C(q_j(x)) = C(p_i(x))$

Se $p_i(x)$ e $q_j(x)$ foram executadas na mesma réplica (cópia)

Se $p_i(x)$ é uma operação de escrita e $q_j(x)$ é uma leitura

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

Se $p_i(x)$ é uma operação de leitura e $q_j(x)$ é uma escrita

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

Se $p_i(x)$ e $q_j(x)$ foram executadas em réplicas distintas

Se $p_i(x)$ é uma operação de escrita e $q_j(x)$ é uma leitura

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

Se $p_i(x)$ é uma operação de leitura e $q_j(x)$ é uma escrita

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

Se $p_i(x)$ e $q_j(x)$ são operações de escrita

o escalonador insere uma aresta da forma $T_j \rightarrow T_i$ no grafo.

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 , remove a aresta inserida e avisa ao cliente C_i sobre o *abort* da transação T_i. Caso contrário, $p_i(x)$ é aceita e escalonada.

Passo 3. Ao receber um pedido de *commit*, de uma transação T_i, juntamente com o número de operações de T_i, o escalonador verifica se já recebeu e escalonou todas as operações da transação T_i e se esta ainda é uma transação ativa. Neste caso o *commit* será concedido e a transação efetivada. Caso o escalonador não tenha recebido ainda todas as operações de T_i , ele deve retardar a resposta. Pode acontecer ainda que a transação T_i já tenha sido anteriormente abortada e que a informação do *abort* não tenha chegado na aplicação cliente. Neste caso *o abort* é re-enviado ao cliente. Após efetivar a transação T_i, o escalonador incrementa a versão (marcador de tempo) de todos os itens de dados atualizados por T_i, da seguinte maneira: $\forall$ x atualizado por T_i faça C(x) = (z,y) receber (z+1, 0). Em seguida, os novos valores dos itens de dados atualizados por T_i, juntamente com os novos marcadores de tempo, devem ser propagados para todas as replicas com as quais o servidor primário tiver comunicação. Caso algum servidor replicado R_j esteja fora da área de comunicação, o seu identificador deve ser armazenado em *log*, para que posteriormente o escalonador tente o reenvio destas informações.

Passo 4. Ao receber um pedido de *abort*, de uma transação T_i, o escalonador, desfaz o efeito das operações de T_i, remove a aresta referente a esta transação e envia a confirmação do *abort* ao cliente C_i. ■

5.3. Exemplo de Aplicação

Para ilustrar a aplicabilidade e uso de nossa proposta, vamos tomar como exemplo uma aplicação de comércio eletrônico onde os diversos produtos são disponibilizados para venda mediante a realização de um leilão eletrônico. Considere que para melhorar a disponibilidade dos dados, obter ganhos de performance e maximizar a vazão, o banco de dados principal (consolidado), BD_{PC}, foi replicado em dois outros *hosts* (R_1 e R_2), ou seja, existem dois servi-

dores replicados com uma cópia (total ou parcial) do banco de dados (BD_{R1} e BD_{R2}). Esta configuração é mostrada na figura 5.1. Considere ainda o seguinte conjunto de transações, as quais lêem e atualizam (efetuam lances) os valores dos produtos à venda:

T_1: $r_1(CEL)r_1(PALM)w_1(CEL, CEL+5)C_1$;
T_2: $r_2(CEL)w_2(CEL, CEL+7)C_2$;
T_3: $r_3(PALM)r_3(CEL)w_3(CEL,CEL+10)C_3$;

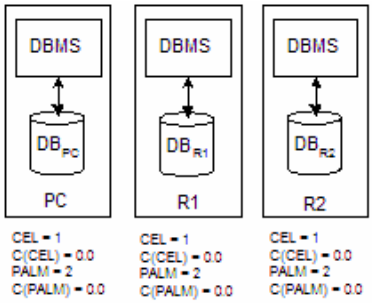


Figura 5.1. Ambiente Replicado.

Considere agora o *schedule* S_1 apresentado na figura 5.2.

$$S_1 = r_1(CEL)_{R1}\ r_1(PALM)_{R1}\ w_1(CEL, CEL+5)_{R1}\ r_2(CEL)_{R1}\ w_2(CEL, CEL+7)_{R2}\ r_3(PALM)_{R1}\ r_3(CEL)_{R1}\ w_3(CEL, CEL+10)_{R2}\ C_1\ C_2\ C_3$$

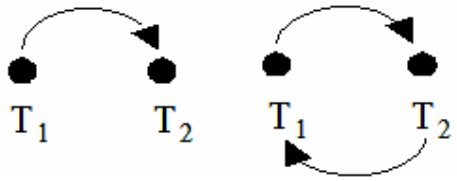
Figura 5.2. Schedule S_1.

Vamos assumir que no cenário de execução apresentado na figura 5.2 as operações foram temporalmente executadas na ordem apresentada. O grafo de serialização para o *schedule* S_1 é ilustrado na figura 5.4 (a). Observe que esse grafo não apresenta um ciclo. Portanto, o *schedule* S_1 poderia ser considerado serializável por conflito (correto). Entretanto, se observarmos o valor final do item 'CEL', veremos que na réplica BD_{R1} este item tem valor igual a 6, já na réplica BD_{R2} e na cópia primária, este tem valor 16. Entretanto, ambos os valores são inconsistentes, uma vez que não seriam gerados por nenhuma execução serial das transações T_1, T_2 e T_3. Desta forma, torna-se necessário re-visitar tanto os protocolos de controle de concorrência quanto os mecanismos para controle de réplicas.

Observe que esta inconsistência decorre do fato de que o valor lido pela operação $r_3(CEL)_{R1}$ não corresponde ao valor escrito por $w_2(CEL, CEL+7)_{R2}$. Portanto, o valor lido pela operação $r_3(CEL)_{R1}$ é anterior ao valor gerado pela operação $w_2(CEL, CEL+7)_{R2}$. O *schedule* temporalmente correto (S_1') é mostrado na figura 5.3, enquanto o grafo de serialização temporal correto para o *schedule* S_1' é apresentado na figura 5.4 (b). Assim, um *schedule* não serializável por conflito (incorreto) foi considerado correto. Naturalmente, este fenômeno deve ser evitado, pois provocaria inconsistências no banco de dados. Pelo protocolo proposto, o grafo de serialização temporal correto seria gerado, um ciclo identificado e a transação T_3 desfeita. Neste caso o valor final do item 'CEL' na réplica BD_{R1} seria igual a 6, já na réplica BD_{R2} e na cópia primária, este item teria valor 16. O valor 16 seria então propagado pela cópia primária para as demais réplicas. Porém, este é um valor consistente, pois seria gerado por ambas as execuções seriais das transações T_1 e T_2.

$$S_1' = r_1(CEL)_{R1}\ r_1(PALM)_{R1}\ w_1(CEL, CEL+5)_{R1}\ r_2(CEL)_{R1}\ r_3(PALM)_{R1}\ \boxed{r_3(CEL)_{R1}\ w_2(CEL, CEL+7)_{R2}}\ w_3(CEL, CEL+10)_{R2}\ C_1\ C_2\ C_3$$

Figura 5.3. Schedule S_1'.



(a) Grafo de Serialização para S_1. (b) Grafo de Serialização Temporal para S_1'.

Figura 5.4. Grafos de Serialização para os Schedules S_1 e S_1'.

Vale a pena ressaltar que mesmo se os pacotes com as informações das leituras e escritas executadas pelas transações nas diversas réplicas chegarem ao servidor primário em uma ordem diferente da ordem em que as operações foram executadas, o protocolo proposto continua gerando o grafo de serialização correto e identificando o ciclo.

Vamos observar agora um segundo exemplo. Considere a mesma configuração anterior. Considere agora o seguinte conjunto de transações, as quais lêem e atualizam (efetuam lances) os valores dos produtos à venda:

T_1: $r_1(CEL)r_1(PALM)w_1(CEL, CEL+5)C_1$;

T_2: $r_2(PALM)r_2(CEL)w_2(CEL, CEL+7)C_2$;

Considere agora o *schedule* S_2 apresentado na figura 5.5.

$$S_2 = r_1(CEL)_{R1}\ r_1(PALM)_{R1}\ w_1(CEL, CEL+5)_{R1}\ r_2(PALM)_{R1}\ r_2(CEL)_{R2}\ w_2(CEL, CEL+7)_{R2}\ C_1\ C_2$$

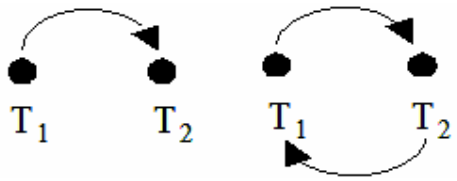
Figura 5.5. Schedule S_2.

Vamos assumir que no cenário de execução apresentado na figura 5.5 as operações foram temporalmente executadas na ordem apresentada. O grafo de serialização para o *schedule* S_2 é ilustrado na figura 5.7 (a). Observe que esse grafo não apresenta um ciclo. Portanto, o *schedule* S_2 poderia ser considerado serializável por conflito (correto). Entretanto, se observarmos o valor final do item 'CEL', veremos que na réplica BD_{R1} este item tem valor igual a 6, já na réplica BD_{R2} e na cópia primária, este tem valor 8. Entretanto, ambos os valores são inconsistentes, uma vez que não seriam gerados por nenhuma execução serial das transações T_1 e T_2.

Observe que esta inconsistência decorre do fato de que o valor lido pela operação $r_2(CEL)_{R2}$ não corresponde ao valor escrito por $w_1(CEL, CEL+5)_{R1}$. Portanto, o valor lido pela operação $r_2(CEL)_{R2}$ é anterior ao valor gerado pela operação $w_1(CEL, CEL+5)_{R1}$. O *schedule* temporalmente correto (S_2') é mostrado na figura 5.6, enquanto o grafo de serialização temporal correto para o *schedule* S_2' é apresentado na figura 5.7 (b). Assim, um *schedule* não serializável por conflito (incorreto) foi considerado correto. Naturalmente, este fenômeno deve ser evitado, pois provocaria inconsistências no banco de dados. Pelo protocolo proposto, o grafo de serialização temporal correto seria gerado, um ciclo identificado e a transação T_2 desfeita. Neste caso o valor final do item 'CEL' na réplica BD_{R1} e na cópia primária seria igual a 6, já na réplica BD_{R2} este item teria valor 1. O valor 6 seria então propagado pela cópia primária para as demais réplicas. Porém, este é um valor consistente, pois foi gerado pela execução de uma única transação (T_1).

$S_2' = r_1(CEL)_{R1}\ r_1(PALM)_{R1}\ r_2(PALM)_{R1}\ \boxed{r_2(CEL)_{R2}\ w_1(CEL, CEL+5)_{R1}}\ w_2(CEL, CEL+7)_{R2}\ C_1\ C_2$

Figura 5.6. Schedule S_2'.



(a) Grafo de Serialização para S_2. (b) Grafo de Serialização Temporal para S_2'.

Figura 5.7. Grafos de Serialização para os Schedules S_2 e S_2'.

5.4. Correção do Protocolo

A seguir demonstraremos que o algoritmo proposto assegura duas propriedades fundamentais: a) 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, e b) todas as réplicas, eventualmente, irão convergir para um mesmo estado consistente.

Primeiramente, demonstraremos que todo *schedule* produzido pelo protocolo proposto é serializável por conflito, ou seja garante a consistência do banco de dados em todos os servidores.

Teorema 1. Seja $TGST_R$ o conjunto de *schedules* sobre um conjunto $\Im=\{T_1, T_2, \ldots, T_N\}$ de transações produzido pelo protocolo proposto e CSR o conjunto de todos os *schedules* serializáveis por conflito sobre $\Im$. Então $TGST_R$ =CSR.

Prova (esboço). É fácil mostrar que $TGST_R \subset CSR$. Nós precisamos apenas observar que todo *schedule* S produzido pelo protocolo proposto apresenta uma grafo de serialização temporal (GST) acíclico. Por definição, grafo de serialização temporal de S representa o grafo de serialização convencional para S, adicionado de informações temporais para capturar a ordem correta de execução das operações em S. Em outras palavras, se o GST para S é acíclico, o grafo de serialização convencional também é. Portanto, $S \in CSR$, conseqüentemente $TGST_R \subset CSR$. Para provar que $TGST_R \supset CSR$, precisamos mostrar que todo *schedule* $S \in CSR$ pode ser produzido pelo protocolo proposto. Nós podemos mostrar isso por indução no tamanho de S que toda operação p em S não pode originar um ciclo no GST. Por esse motivo a operação p pode ser executada. Como já mencionado anteriormente, o GST representa o grafo de serialização convencional para S, adicionado de informações temporais. ■

Agora iremos demonstrar que todas as réplicas, eventualmente, irão convergir para um mesmo estado consistente.

Teorema 2. Seja $R=\{R_1, R_2, \ldots, R_N\}$ o conjunto dos servidores replicados, e $DB(R_1,t)$ o estado do banco de dados no servidor replicado R_1, no instante de tempo t. Então, existirá um determinado tempo t onde, $\forall$ i,j (com $i \neq j$, $0<i,j\leq n$) $DB(R_i,t) = DB(R_j,t)$ e $DB(R_i,t)$ é consistente.

Prova (esboço). Pelo passo 3 do protocolo, observamos que após a efetivação de uma transação T_i, o escalonador propaga os novos valores dos itens de dados atualizados por T_i, para todas as replicas com as quais o servidor primário tiver comunicação. Caso algum servidor replicado R_k esteja fora da área de comunicação, o seu identificador deve ser armazenado em l*og*, para que posteriormente o escalonador tente o re-envio destas informações. Certamente, haverá um instante de tempo t, no qual o sistema estará na ausência de atualizações. Neste instante, $DB(R_j,t)$, onde R_j é o servidor primário, é consistente e estável, e, adicionalmente, todos os servidores replicados R_k terão mantido uma conexão, mesmo temporária, com o servidor primário R_j, recebendo as atualizações pendentes e atualizando o estado de sua réplica local, fazendo $DB(R_k,t) = DB(R_j,t)$. Logo, pelo Teorema 1, $DB(R_k,t)$ é consistente ($\forall$ k com $0<k\leq n$). ■

O mecanismo para controle de concorrência que estamos propondo utiliza serializabilidade para garantir a consistência dos dados. Vale destacar que serializabilidade já se tornou um padrão para controle de concorrência em bancos de dados, sendo implementada em praticamente todo SGBD existente no mercado. Além de utilizar um critério de correção já consolidado, o protocolo proposto relaxa a propriedade de isolamento. Esta propriedade prevê que as transações são isoladas, ou seja, que seus resultados só podem ser vistos ao final de sua execução, tenha ela terminado ou não com sucesso. O protocolo apresentado relaxa o isolamento e adota a abordagem *ready-any/write-any,* permitindo uma maior disponibilidade dos dados. Além disso, as operações que compões as diversas transações são executadas uma única vez e os servidores não necessitam armazenar um *log* com todas as operações realizadas. Essas duas últimas propriedades não são garantidas pela proposta apresentada em [1]. Além disso, nossa abordagem reduz o numero de mensagens trocadas entre os servidores. Dessa forma, nosso protocolo procurar garantir uma economia no custo da utilização dos canais de comunicação, na limitada capacidade de energia dos computadores portáteis e nos seus escassos recursos de memória.

Por outro lado, o servidor precisa manter um grafo de serialização de todas as transações ativas, o que pode comprometer a escalabilidade. Todavia, a estratégia de coleta de lixo apresentada em [4] pode ser utilizada para reduzir o tamanho do grafo. Outra desvantagem consiste no fato dos servidores replicados terem que comunicar ao servidor primário os itens lidos e/ou escritos por suas transações. Entretanto, a informação a ser enviado consiste apenas do identificador do item, da transação, do servidor e um *timestamp*. Semelhantemente à abordagem mostrada em [1], um servidor primário é utilizado e a coerência entre as replicas é garantida apenas eventualmente. Porém, no nosso protocolo, a detecção de conflitos é realizada através da comparação entre *timestamps*, os quais consistem em pares de valores numéricos, o que proporciona um custo pequeno. Já em [1], é utilizado um método onde cada escrita está associada a um conjunto de regras de validação e a um procedimento, escrito em uma linguagem interpretada de alto nível, para a resolução de conflitos, ou seja, a tarefa de detecção e resolução de conflitos é transferida aos programadores. Além disso, a suposição assumida no Teorema 2 de que haverá um instante de tempo t no qual o sistema estará na ausência de atualizações, apesar de parecer forte, é comum em outros mecanismos de replicação, como por exemplo, a abordagem apresentada em [1, 2].

6 Trabalhos Relacionados

O problema da consistência de dados replicados em ambientes de computação móvel tem sido tratado por diversos trabalhos, os quais buscam como principal objetivo possibilitar uma alta disponibilidade dos dados. Muitas das soluções propostas buscam assegurar a consistência eventual em direção a um estado consistente. Golding[11] propôs que cada servidor, individualmente, consolide uma atualização quando este estiver certo de esta atualização já foi recebida por todas as demais replicas. A principal limitação desta abordagem e que a indisponibilidade de uma única replica inviabiliza o processo de *commit* como um todo (*commitment process*). No projeto Bayou[1], cada servidor armazena um *log* contendo todas as escritas de que tem conhecimento. Além disso, no Bayou, uma mesma operação pode ser re-executada diversas vezes, o que pode gerar *overhead*. As abordagens baseadas em votação permitem utilizar uma grande variedade de quoruns para se decidir pelo *commit* de uma atualização Em particular, Deno [9,10] utiliza um protocolo de votação epidêmico a fim de suportar a replicação de objetos em um *framework* transacional para ambientes fracamente conectados. Deno requer que um round (etapa) de votação seja executado por completo para cada atualização. Isto pode ser aceitável quando as aplicações estão interessadas em conhecer o andamento do processo de *commit* para cada tentativa de atualização sobre um determinado item, antes de executar uma outra operação sobre este mesmo item e dado. Entretanto, nos cenários considerados por este trabalho, usuários e aplicações estarão muitas vezes interessados em acessar itens de dados atualizados por operações de escrita ainda não efetivadas. Em tais situações, a espera pelo *commit* imposta elo protocolo de votação do Deno se torna inaceitavelmente tão alta quanto no protocolo *primary commit*.

7 Conclusões e Trabalhos Futuros

Neste trabalho apresentamos um novo mecanismo que garante a consistência de dados replicados (*one-copy serializability*) e a convergência das diversas réplicas para um único estado final consistente (consistência eventual), em ambientes de computação móvel. Esse protocolo explora informações temporais referentes ao momento em que um objeto do banco de dados replicado foi lido ou atualizado e utiliza serializabilidade como critério de correção. Vale salientar que serializabilidade já se tornou um padrão, e, por isso, praticamente todo SGBD existente no mercado implementa este modelo. Além de utilizar um critério de correção já consolidado, o protocolo proposto relaxa a propriedade de isolamento e adota a abordagem *ready-any/write-any,* permitindo uma maior disponibilidade dos dados. Neste mecanismo, as operações que compões as diversas transações são executadas uma única vez e os servidores não necessitam armazenar um *log* com todas as operações realizadas. Além disso, nossa abordagem reduz o numero de mensagens trocadas entre os servidores. Tudo isso irá garantir uma economia no custo da utilização dos canais de comunicação, na limitada capacidade de energia dos computadores portáteis e nos seus escassos recursos de memória.

Como trabalhos futuros pretendemos estender o protocolo proposto a fim de permitir que o nível de isolamento utilizado seja adaptado às condições do ambiente, através da escolha de um dentre os quatro níveis de isolamento padronizados pela ANSI/ISO SQL 92 [3], e possibilitar a propagação dos itens atualizados de maneira ponto-a-ponto (P2P).

Referências Bibliográficas

1. Terry, D., Demers, A., Petersen, K., Spreitzer, M., Theimer, M., Welch, B. e Hauser, C. (1995) "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System".
2. Terry, D., Demers, A., Petersen, K., Spreitzer, M. e Theimer, M. (1997) "Flexible Update Propagation for Weakly Consistent Replication".
3. Adya, A., Liskov, B. e O'Neil, P. (2000) "Generalized Isolation Level Definitions". In proceedings of the IEEE International Conference on Data Engineering, San Diego, CA, March 2000.
4. Monteiro, J. e Brayner, A. (2005) "Controlling Concurrency in Mobile Computing Environments with Broadcast-based Data Dissemination". Proceedings of the EUROPAR, Lisboa, Portugal, 2005.
5. Casanova, M. A. (1981) "The Concurrency Problem of Database Systems". In Lectures Notes in Computer Science, 116, 1981.
6. Bernstein, P. A., Hadzilacos, V. And Goodman, N. Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987.
7. Saito, Y. e Shapiro, M. (2005) "Optimistic Replication". ACM Computing Surveys, Vol. V, No. N, 3 2005.
8. Barreto, J. e Ferreira, P. (2005) "An Efficient and Fault-Tolerant Update Commitment Protocol for Weakly Connected Replicas". Proceedings of the EUROPAR, Lisboa, Portugal, 2005.
9. Cetintemel, U., Keleher, P. e Franklin, M. (2001) "Support for Speculative Update Propagation and Mobility in Deno". Proceedings of the The 21st IEEE International Conference on Distributed Computing Systems (ICDCS'01).
10. Keleher, P. (1999) "Decentralized replicated-object protocols". In: Proc. of the 18th Annual ACM Symp. on Principles of Distributed Computing (PODC'99).
11. Golding, R., Long, D. "Modeling replica divergence in a weak-consistency protocol for global scale dirstibuted data bases". Technical Report UCSC-CRL-93-09 (1993).
12. Maya, R. e Anthony, L. (2003) "Decentralized Weighted Voting for P2P Data Management". Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access, San Diego, CA, USA, 2003 (MobiDe '03).
13. Mateus G. e Loureiro A. Introdução à Computação Móvel. 1998.
14. Monteiro, J. e Brayner, A. (2000). "Temporal Serialization Graph Testing: An Approach to Control Concurrency in Broadcast Environments". Proceedings of the15th Brazilian Symposium on Databases - SBBD'2000, João Pessoa, Brazil, 2000.
15. Brayner, A. R. A., Aguiar, J. M. F. (2006) "On Mobile Transaction Processing in Dynamically Configurable Mobile Database Communities". International Journal Of Parallel Emergent And Distributed Systems, Inglaterra, v. 21, n. 3, p. 199-213, 2006.
16. Hara T. Replica Allocation Methods in Ad Hoc Networks with Data Update. MONET 8(4): 343-354 (2003) .
17. Angelo Brayner, Frank Stefan Alencar: A Semantic-Serializability Based Fully-Distributed Concurrency Control Mechanism for Mobile Multi-Database Systems. DEXA Workshops 2005: 1085-1089.
18. Luo J., Hubaux J., e Eugster P. PAN: Providing Reliable Storage in Mobile Ad Hoc Networks with Probabilistic Quorum Systems. ACM MobiHoc 2003.
19. Rodrig M. e Lamarca A. Decentralized Weighted Voting for P2P Data Management. MobiDE'03. San Diego, CA, September 19, 2003.

20. Brayner A. Transaction Management in Multidatabase System. 1999.
21. Angelo Brayner, Theo Härder: Global Semantic Serializability: An Approach to Increase Concurrency in Multidatabase Systems. CoopIS 2001: 301-315 22.
22. Angelo Brayner, José de Aguiar M. Filho: Increasing Mobile Transaction Concurrency in Dynamically Configurable Environments. ICDCS Workshops 2005: 637-641
23. Angelo Brayner, José de Aguiar M. Filho: Sharing Mobile Databases in Dynamically Configurable Environments. CAiSE 2003: 724-737.
24. Takahiro Hara: Location Management of Replicas Considering Data Update in Ad Hoc Networks. AINA (1) 2006: 753-758.
25. Hideki Hayashi, Takahiro Hara, Shojiro Nishio: On Updated Data Dissemination Exploiting an Epidemic Model in Ad Hoc Networks. BioADIT 2006: 306-321.
26. Yohei Sawai, Masako Shinohara, Akimitsu Kanzaki, Takahiro Hara, Shojiro Nishio: Consistency Management among Replicas Using a Quorum System in Ad Hoc Networks. MDM 2006: 128.
27. Hideki Hayashi, Takahiro Hara, Shojiro Nishio: Update Log Dissemination in Mobile Ad Hoc Networks. MDM 2006: 71
28. Masako Shinohara, Hideki Hayashi, Takahiro Hara, Shojiro Nishio: Replica Allocation Considering Power Consumption in Mobile Ad Hoc Networks. PerCom Workshops 2006: 463-467.
29. Takahiro Hara: Data Replication Issues in Mobile Ad Hoc Networks. DEXA Workshops 2005: 753-757.
30. Takahiro Hara, Sanjay Kumar Madria: Consistency Management among Replicas in Peer-to-Peer Mobile Ad Hoc Networks. SRDS 2005: 3-12
31. Takahiro Hara: Location management of data items in mobile ad hoc networks. SAC 2005: 1174-1175.
32. Takahiro Hara, Sanjay Kumar Madria: Dynamic Data Replication Using Aperiodic Updates in Mobile Adhoc Networks. DASFAA 2004: 869-881
33. Hideki Hayashi, Takahiro Hara, Shojiro Nishio: Updated Data Dissemination in Ad Hoc Networks. UMICS 2004: 28-42.