


Este exemplar corresponde à redação final da
Tese/Dissertação devidamente corrigida e defendida
por: Dário Vieira Conceição
e aprovada pela Banca Examinadora.
Campinas, 14 de outubro de 99

COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Consistência de Dados em um Ambiente
de Computação Móvel

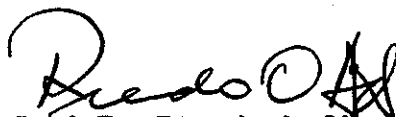
Dário Vieira Conceição

Dissertação de Mestrado

Consistência de Dados em um Ambiente de Computação Móvel

Este exemplar corresponde à redação final da
Dissertação devidamente corrigida e defendi-
da por Dário Vieira Conceição e aprovada pela
Banca Examinadora.

Campinas, 24 de Agosto de 1999.



Prof. Dr. Ricardo de Oliveira Anido
(Orientador)

Dissertação apresentada ao Instituto de Com-
putação, UNICAMP, como requisito parcial para
a obtenção do título de Mestre em Ciência da
Computação.

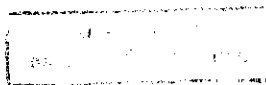
Consistência de Dados em um Ambiente de Computação Móvel

Dário Vieira Conceição

Agosto de 1999

Banca Examinadora:

- Prof. Dr. Ricardo de Oliveira Anido (Orientador)
IC-UNICAMP
- Profa. Dra. Regina Helena Carlucci Santana
ICMC-USP-São Carlos
- Profa. Dra. Maria Beatriz Felgar de Toledo
IC-UNICAMP
- Prof. Dr. Célio Cardoso Guimarães (suplente)
IC-UNICAMP



UNIDADE	BC
N.º CHAMADA:	
V.	EX
CLASSE	80,39 277
PROJ.	229/99
	<input type="checkbox"/> <input checked="" type="checkbox"/>
VALOR	R\$ 11,00
DATA	28/10/99
N.º OPD	

CM-00136605-B

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Conceição, Dário Vieira

C744c Consistência de dados em um ambiente de computação móvel /
Dário Vieira Conceição -- Campinas, [S.P. :s.n.], 1999.

Orientador : Ricardo de Oliveira Anido

Dissertação (mestrado) - Universidade Estadual de Campinas,
Instituto de Computação.

1. Sistemas operacionais distribuídos (Computadores). 2.
Programa de computação (Validação). 3. Tolerância a falha
(Computação). 4. Processamento paralelo (Computadores) - Protocolos .
I. Anido, Ricardo de Oliveira. II. Universidade Estadual de Campinas.
Instituto de Computação. III. Título.

*“Minha alma engrandece ao Senhor
Meu espírito se alegra em Deus, meu Salvador
Pois com poder tem feito grandes coisas
E com misericórdia, demonstrado amor.”*

(Asaph Borba)

Ao meu Deus, provedor da vida e
de todo conhecimento.

Aos meus queridos e amados pais.

A minha avó, Constancia Maria Senna,
(*in memoriam*).

Agradecimentos

*“Ouse sonhar e lutar
pelo que você acredita;
insista em mostrar seu rosto,
ocupar seu posto no mundo.”*
(Fátima Machado)

Traz tão grande paz, reconhecer, Senhor, o Teu cuidado. Ao olhar pra trás e Te perceber nas cores do passado, tenho a plena convicção de que foi Tua mão que sustentou o sonho acalentado. Por isto, posso dizer que valeu demais... o tempo e o contratempo, as noites mal dormidas... o tempo de esperar o Teu querer. Como posso agradecer pelo bem que tens feito a mim? “Aquele que leva a preciosa semente andando e chorando, voltará com júbilo, trazendo consigo os seus feixes.” E é com júbilo que agradeço a Ti, ó Deus, pelo grande amor que tens demonstrado, pois com efeito grandes coisas tens feito por mim!

“Há homens que lutam um dia e são bons. Há outros que lutam um ano e são melhores. Há aqueles que lutam muitos anos e são muito bons. Mas há os que lutam a vida inteira. Esses são os imprescindíveis”, como meus pais, Samuel Vieira Senna e Maria de Lourdes Vieira. Obrigado pelo apoio, confiança, dedicação, orações, pelo grande amor e, sobretudo, pelos pais e amigos que têm sido para mim! Agradeço a Deus pela bênção de tê-los como meus pais!

Aos meus amados e queridos irmãos, irmãs, sobrinhos, sobrinhas e demais familiares, pela amizade, pelo incentivo, força, apoio, orações e confiança depositada.

A Elisângela de Araújo Rodrigues, pelo amor demonstrado, pela aprazível companhia, pelo incentivo e carinho constantes, pela amizade, orações e apoio fundamental, sobretudo, na etapa final desta dissertação. Que Deus possa te recompensar pela enorme contribuição!

“Professores ideais são os que se fazem de ponte, que convidam os alunos a atravessarem e depois, tendo facilitado a travessia, desmoronam-se com prazer, encorajando-os a criarem suas próprias pontes”. Agradeço ao meu orientador, Ricardo de Oliveira Anido, por se fazer de ponte, e encorajar-me a criar a minha.

A Elibia Maria de Souza, pelo carinho, amizade, orações, as palavras de estímulo, incentivo constante, apoio irrestrito e fundamental, e enorme contribuição para realização de mais uma vitória. De certo, tua participação foi imprescindível e, somente Deus pode recompensar-te por tudo que fizeste por mim!

Ao Cláudio Marcio da Silveira e Silvio Jamil F. Guimarães. É amigos... amigo é coisa pra se guardar do lado esquerdo do peito, mesmo que o tempo e a distância digam não... Obrigado sobretudo pela amizade, pelo incentivo, apoio, pela agradável companhia, por tornar nossa república um ambiente aprazível e pelas discussões sobre o projeto.

A Igreja Assembléia de Deus de Ilhéus (sobretudo às senhoras do Círculo de Oração), ao Centro Evangélico de Missões, a Comunidade Cristã Koinonia, a Editora Ultimato e, principalmente, a Igreja Presbiteriana de Viçosa, pelas orações, apoio, amizade e carinho!

Ao grupo de trabalho do LSD pela companhia, amizade e pelas diversas discussões que só levam ao crescimento pessoal e profissional.

A Delano, Luciane, Nadia, Rodrigo, Wanderley, Pablo, Roberto, Marta, César, Erlon, Mauro, aos colegas de graduação, ao Departamento de Informática da Universidade Federal de Viçosa, a toda turma do Msc97, e demais amigos de Campinas pela amizade e companhia.

Ao Instituto de Computação, pelo apoio acadêmico, e aos seus funcionários pela colaboração e amizade.

A Fapesp e ao CNPq, pelo apoio financeiro.

Resumo

A rápida expansão da tecnologia de comunicação sem fio e dos serviços de satélite proporcionaram o aparecimento de um novo paradigma no campo da ciência da computação, chamado *computação móvel*. Nos sistemas móveis as soluções para o problema de distribuição diferem das existentes em computação distribuída convencional, devido a características intrínsecas do ambiente móvel, tais como: desconexão, mobilidade, estreita banda passante, alta latência, baixa capacidade de armazenamento e consumo de energia. Assim, embora as soluções básicas de problemas distribuídos sejam bem entendidas, uma análise de algumas destas em relação ao ambiente de computação móvel pode ser de vital importância na busca de novas soluções, adequadas a este ambiente. Esta dissertação analisa aspectos de um problema específico em sistemas distribuídos - consistência de dados - no contexto de computação móvel. São apresentados um novo modelo de execução de transação e um novo protocolo de gravação e recuperação de estados globais consistentes, ambos formando um arcabouço de consistência de dados adequado ao ambiente de computação móvel.

Abstract

The fast advance in wireless communication technologies and satellites service have enabled the appearance of a new paradigm in computer science, called *mobile computing*. Solutions to problems in mobile computing are different from those to traditional distributed systems, due to a set of inherent characteristics of mobile environment, like: mobility, frequent disconnection, low bandwidth, high latency, reduced storage capacity, limited battery life and small screen sizes. Therefore, although the basic issues in the traditional distributed environment are well understood, a review of some of these issues with respect to the mobile environment are useful to comprehend those characteristics. In this dissertation it is proposed a new transaction model and a new checkpoint protocol for the mobile environment. The transaction model and checkpoint protocol form an adequate framework for data consistency to mobile computing environment.

Conteúdo

Agradecimentos	vii
Resumo	ix
Abstract	x
Lista de Definições	xiv
Lista de Tabelas	xv
Lista de Figuras	xvi
1 Introdução	1
2 Características do ambiente de computação móvel	4
2.1 Redes de comunicação sem fio	4
2.1.1 Protocolos	6
2.2 Largura de banda	6
2.3 Desconexões	7
2.4 Mobilidade	7
2.5 Heterogenidade	8
2.6 Portabilidade	9
2.7 Baixa capacidade de armazenamento	9
2.8 Restrição de energia	10
2.9 Resumo do capítulo	11
3 Conceitos Básicos	13
3.1 O modelo do sistema	13
3.2 Relógios lógicos	15
3.2.1 Tempo escalar	16
3.2.2 Tempo vetorial	17

3.2.3	Tempo matricial	19
3.3	Armazenamento estável	20
3.4	Ações atômicas	20
3.4.1	Ações atômicas em sistemas distribuídos	22
3.4.2	Gerenciador de <i>commit</i>	25
3.4.3	Controle de concorrência	28
3.4.4	Transações aninhadas	32
3.5	Modelo de ações atômicas no ambiente móvel	32
3.5.1	O modelo de Chrysanthis	32
3.5.2	O modelo de Narasayya	33
3.5.3	O modelo de Qi Lu e Satyanarayanan	33
3.5.4	O modelo de Evangelgia e Bharat	33
3.5.5	O modelo de Jing, Bukhres e Elmagarmid	34
3.5.6	O modelo de Walborn e Chrysanthis	34
3.6	Pontos de recuperação	34
3.6.1	Estado consistente do sistema	35
3.6.2	Efeito dominó	37
3.6.3	Coleta de lixo	37
3.6.4	Recuperação por retrocesso baseada no estabelecimento de pontos de recuperação	38
3.7	Algoritmos de pontos de recuperação para o ambiente móvel	39
3.7.1	O algoritmo de Neves e Fuchs	39
3.7.2	O algoritmo de Prakash e Sihghal	40
3.7.3	O algoritmo de Acharya e Badrinath	40
3.7.4	O algoritmo de Pradhan, Krishna e Vaidya	41
3.8	Resumo do capítulo	41
4	Consistência de dados usando ações atômicas	45
4.1	Introdução	45
4.2	Problemas com os esquemas atuais	46
4.2.1	Desconexão	46
4.3	O modelo proposto	47
4.3.1	Unidade móvel conectada	48
4.3.2	Unidade móvel vai desconectar-se	48
4.3.3	Unidade móvel desconectada	49
4.3.4	Transações aninhadas	49
4.4	Resumo do capítulo	52

5	Consistência de dados usando pontos de recuperação	53
5.1	Introdução	53
5.2	Pontos de recuperação no ambiente móvel	54
5.2.1	Estado global consistente	55
5.3	O Algoritmo proposto	56
5.3.1	Reduzindo o tamanho das mensagens	57
5.3.2	Estabelecimento de pontos de recuperação	58
5.3.3	Protocolo de gravação de estado	59
5.3.4	Protocolo de reconstrução de estados globais	60
5.4	Resumo do capítulo	61
6	Conclusões e trabalhos futuros	63
	Bibliografia	65

Lista de Definições

Armazenamento estável	20
Ação atômica	20
Transação	20
Escalonador	24
Protocolos de <i>commit</i>	25
Protocolos de controle de concorrência	28
<i>Lock</i>	28
<i>Timestamp</i>	31
Transações aninhadas	32
Pontos de recuperação	35
Linha de recuperação	35
Recuperação por retrocesso	35
Efeito Dominó	37
Coleta de lixo	38
Estabelecimento de pontos de recuperação de forma assíncrona	38
Estabelecimento de pontos de recuperação de forma síncrona	38
Estabelecimento de pontos de recuperação de forma quase síncrona	39

Lista de Tabelas

2.1	Principais diferenças entre o ambiente estático e móvel.	11
2.2	Tabela comparativa de unidades móveis e estáticas.	12
3.1	Estabelecimento de pontos de recuperação no ambiente móvel	44
3.2	Comparação entre os algoritmos de gravação e recuperação de estados globais no ambiente móvel	44

Lista de Figuras

3.1	Modelo para algoritmos distribuídos	14
3.2	Exemplo de uma computação com tempo escalar	17
3.3	Exemplos de transação	21
3.4	Exemplo de concorrência entre transações	22
3.5	Exemplo do modelo de ações atômicas em sistemas distribuídos	23
3.6	Possíveis ações tomadas por um escalonador	24
3.7	Autômato de estados finitos do protocolo de <i>commit</i> de duas fases	27
3.8	Autômato de estados finitos do protocolo de <i>commit</i> de três fases	27
3.9	Exemplo de <i>lock</i> binário	29
3.10	Exemplo de <i>locks</i> múltiplos	30
3.11	(a) Corte consistente; (b) Corte inconsistente	36
3.12	Linha de recuperação e efeito dominó	37
3.13	Possíveis estados de uma transação	43
4.1	Modelo proposto de execução de transação	48
5.1	Gravação de estados locais assincronamente	56
5.2	Exemplo de uma computação com a técnica de Jard-Jourdan	59
5.3	Exemplo de uma computação com o protocolo proposto	62

Capítulo 1

Introdução

*“Bem-aventurado o homem que acha sabedoria,
e o homem que adquire conhecimento.”*

(Pv 3.13)

A rápida expansão da tecnologia de comunicação sem fio e dos serviços de satélite proporcionaram o aparecimento de um novo paradigma no campo da ciência da computação, chamado *computação móvel*. Este novo paradigma está revolucionando a maneira como os computadores são usados, possibilitando, aos usuários dos sistemas móveis, acesso às informações independentemente de suas localizações.

Computação móvel representa uma extensão do conceito de computação distribuída. Um sistema distribuído composto por unidades móveis consiste de uma parte fixa, onde existem alguns sistemas que se comunicam com as unidades móveis, e de uma parte móvel representada por uma área ou célula onde existe a comunicação sem fio. No entanto, as soluções em computação móvel diferem daquelas propostas para a computação distribuída convencional, devido às características únicas dos computadores móveis e da tecnologia de comunicação sem fio. Embora as soluções básicas relacionadas a sistemas distribuídos sejam bem entendidas, uma análise de algumas destas em relação ao ambiente de computação móvel pode ser de vital importância para a melhoria de desempenho e busca de novas soluções. O objetivo principal da computação móvel é fornecer aos usuários um ambiente composto de um conjunto de serviços comparáveis aos existentes na computação distribuída convencional.

O conjunto de características intrínsecas ao ambiente de computação móvel introduz uma série de novos problemas interessantes nos campos da ciência da computação e telecomunicações. Estes englobam rede de computadores, sistemas operacionais, otimização, sistemas de informação, banco de dados, sistemas distribuídos, etc. O objetivo deste trabalho foi estudar as soluções existentes na literatura para algumas classes de problemas

em sistemas estáticos que envolvam consistência de dados (e.g., difusão confiável, manutenção de consistência de réplicas, entrega confiável de mensagens, validação de transação) e analisar como estas soluções se comportam em ambientes de computação móvel. Diante destes estudos e análises, está sendo proposto um novo protocolo para gravação e recuperação de estados globais consistentes e um novo modelo de execução de transação, ambos adequados ao ambiente móvel.

Transação é um bom mecanismo para se garantir correção e consistência de dados em um ambiente distribuído composto por unidades móveis. No entanto, os modelos tradicionais de execução de transações em um ambiente móvel são seriamente afetados principalmente devido a desconexões e o meio de comunicação sem fio. Devido a desconexão, uma computação móvel pode ser interrompida ou suspensa, enquanto que o meio de comunicação sem fio possui atrasos consideráveis, acarretando um alto custo tanto de energia quanto financeiro. No capítulo 4, apresenta-se um novo modelo de execução de transações, adequado às características do ambiente móvel, visto que fornece às unidades móveis autonomia durante a execução de suas aplicações, reduz o uso da estreita banda passante (diminuindo o custo financeiro e o consumo de energia) e proporciona um alto grau de concorrência.

Um outro bom mecanismo para se garantir consistência de dados em ambiente distribuído, composto por unidade móveis, é o uso de gravação e recuperação de estados globais consistentes. Este mecanismo é útil para garantir que apesar da desconexão, perda ou falha de uma das unidades móveis a computação móvel possa ser restaurada a partir do último estado global salvo. No capítulo 5, apresenta-se um novo protocolo para o problema de gravação e recuperação de estado global consistente em um ambiente móvel. Neste protocolo, o tamanho das mensagens enviadas é minimizado, amenizando desta forma os problemas de espaço de armazenamento (tanto do ponto de vista de capacidade como de segurança), de consumo de energia e do uso da estreita banda passante.

O protocolo proposto e o modelo de execução de transações formam um arcabouço de consistência de dados para o ambiente de computação móvel. Embora seja necessário reconhecer que é preciso verificar a adequação através de protótipos e experimentos, acredita-se que as soluções propostas sejam adequadas ao ambiente móvel visto que, lidam de maneira eficiente com os problemas relacionados a este ambiente.

Esta dissertação está estruturada da seguinte forma: no capítulo 2, são descritas as principais características do ambiente de computação móvel. No capítulo 3, são introduzidos os principais conceitos necessários ao bom entendimento desta dissertação, além dos principais resultados encontrados na literatura sobre gravação e recuperação de estados globais consistentes e ações atômicas no ambiente de computação móvel. Nos capítulos 4 e 5, são apresentados, respectivamente, o modelo de execução de transações, e o protocolo de pontos de recuperação (*checkpoint*) propostos. Finalmente no capítulo 6, apresenta-

se as principais contribuições e conclusões desta dissertação, bem como propostas para trabalhos futuros. No final de cada capítulo tem-se ainda um resumo com os principais tópicos tratados.

Capítulo 2

Características do ambiente de computação móvel

*“A sabedoria é a coisa principal;
adquire, pois, a sabedoria;
sim, com tudo o que possuis,
adquire o conhecimento”*
(Pv 4.7)

Os fatores que diferenciam as aplicações distribuídas em computação móvel das aplicações nos sistemas distribuídos convencionais são as características intrínsecas do ambiente de computação móvel. Nas próximas subseções analisa-se tais características com maiores detalhes.

2.1 Redes de comunicação sem fio

Conexão através de dispositivos sem fio é um fator decisivo para se conseguir mobilidade, embora as limitações na largura de banda imponham severas restrições no volume de dados que podem ser transferidos no enlace de comunicação sem fio [IB94]. A restrição da largura de banda limita, ainda, o número de usuários simultâneos, a natureza da aplicação que pode ser executada, ou ambos [Duc92].

A comunicação sem fio é mais difícil de ser obtida do que a que se dá através dos métodos convencionais. Isto se deve à interferência do meio interagindo com o sinal, além da introdução de ruídos e ecos. Consequentemente, essas comunicações têm uma qualidade mais baixa do que as tradicionais: estreita largura de banda, alta taxa de erro e freqüentes desconexões. Estes fatores podem aumentar a latência devido a retransmissões,

erro no processamento do protocolo de controle e desconexões. Além disso, há o problema de interceptação de transmissão, comprometendo a segurança e privacidade dos usuários dos sistemas. Portanto, existe a necessidade de uma medida de segurança que possa ser obtida via métodos de criptografia e autenticação implementados em *software* ou em *hardware*.

As atuais redes de comunicação sem fio disponíveis podem ser classificadas como a seguir:

- **Celular:** as redes existentes de telefonia celular fornecem serviços de voz e dados para seus usuários. Este tipo de rede cobre uma área pequena, e tem problemas com escalabilidade (para acomodar um número grande de usuários), transmissão de dados, e estreita largura de banda;
- **LAN sem fio:** LAN sem fio pode ser conectada a uma unidade móvel ou a uma rede estática via interface sem fio, e utiliza uma antena para estabelecer a comunicação entre as unidades. Este tipo de sistema cobre somente uma área local, por exemplo, o interior de um prédio, e não fornece suporte à mobilidade;
- **WAN sem fio:** este tipo de rede fornece serviços de correio eletrônico ou de acesso a partir de uma unidade móvel a uma aplicação sendo executada em uma unidade estática. Além disso, cobre grandes áreas, mas tem estreita largura de banda e pode ter problemas de escalabilidade;
- **Paging:** redes *paging* fornecem uma cobertura ilimitada com estreita largura de banda, mas transferência de dados somente em uma direção.

A taxa de dados para redes no espectro infra-vermelho vai de 19.2 kbps a 1 Mbps, e para redes via radio é de 19.2 kbps. LAN sem fio tem uma taxa de dados de 1 a 2 Mbps e pode estender-se para 100 Mbps, enquanto que nas redes tradicionais LAN a taxa de dados chega a 100 Mbps. Fica claro, portanto, que os atuais tipos de redes não são capazes de fornecerem suporte adequado para um ambiente de computação móvel.

Objetivando lidar com o problema de enlace de dados neste ambiente, técnicas de compressão de dados podem ser usadas para minimizar o problema de largura de banda e, de buscas antecipadas para facilitar e otimizar o fluxo de dados para as unidades móveis. Protocolos têm sido propostos para compensar a baixa velocidade de alguns enlaces e reduzir o custo de comunicação. Por exemplo, esquemas de agregação de banda passante têm sido sugeridos combinando diversos sinais, com o objetivo de fornecer uma alta banda passante por um certo período de tempo.

2.1.1 Protocolos

A fim de lidar explicitamente com o problema de mobilidade, novos protocolos são necessários. Por exemplo, os atuais protocolos TCP/IP para redes tradicionais não são adequados para as dos sistemas móveis. Vários protocolos têm sido propostos, com a finalidade de tornar a operação e o desempenho de uma unidade móvel indistinguíveis das fixas. Para isto, dois requisitos devem ser satisfeitos: transparência operacional e de desempenho. Transparência operacional significa não ter que reinicializar o sistema ou aplicações individuais quando uma mudança de localização ocorrer. Transparência de desempenho significa otimizar o roteamento de pacotes para e a partir de uma unidade móvel. Outros problemas que esses protocolos devem tratar vão desde privacidade, segurança a correção de pacotes.

Uma solução bastante utilizada envolve o conceito de estação base - *home base*. Nesta, cada unidade móvel possui uma estação base responsável pela entrega e envio dos seus pacotes. Toda vez que uma unidade móvel mudar sua localização ela deve informar à sua estação base sua nova localização - esse processo é conhecido como *handoff*. Esta solução tem sido usada pelos protocolos IP objetivando adaptá-los ao ambiente móvel.

2.2 Largura de banda

Um outro fator que é de vital importância no projeto de protocolos e algoritmos distribuídos, para o ambiente de computação móvel, é a largura de banda. Uma vez que a largura de banda nas redes de computação móvel é menor do que nas redes tradicionais, as soluções para o ambiente móvel tendem a se concentrar mais no problema de contenção de largura de banda do que as propostas para as aplicações distribuídas tradicionais.

A capacidade de transmissão de uma rede é dividida entre os seus vários usuários, e, portanto, a sua distribuição por usuário é uma medida mais útil da capacidade da rede do que a capacidade de transmissão total do meio. Essa medida depende do número de e da distribuição dos usuários [FZ94].

Largura de banda impõe severas restrições no volume de dados que podem ser transmitidos em uma rede composta de sistemas móveis. No entanto, uma vez que o custo para se fazer *broadcasting* sobre o enlace sem fio não depende do número de usuários, *broadcasting* torna-se um método atrativo para disseminações de informações em redes sem fio.

2.3 Desconexões

A principal diferença entre desconexão e falha é que a primeira tem uma natureza eletiva: desconexão pode ser tratada como uma falha planejada, que pode ser antecipada e preparada. Existem vários tipos de desconexões, desde a total até a fraca ou parcial.

Os software atuais não têm suporte para a operação de desconexão dos sistemas móveis. Primeiro, existe o problema de como operar na presença de uma desconexão. Segundo, existe a questão de como eliminar a necessidade de reinicialização dos sistemas ou aplicações individuais após uma reconexão ter acontecido.

Por causa das restrições de energia, sistemas móveis irão operar frequentemente desconectados. Entretanto, esta característica não deve ser um empecilho na busca de soluções para as aplicações distribuídas no ambiente de computação móvel. Com o objetivo de manter a operação sendo desenvolvida pelo usuário mesmo na presença de uma desconexão, várias soluções têm sido propostas. Uma solução é tornar a unidade móvel mais autônoma, isto é, menos dependente da rede, usando métodos como *busca antecipada* e *hoarding*. Os sistemas de arquivos *CODA*, [KS91a], *SEER*, [Kue97a], e o proposto por [TLAC95] são bons exemplos deste tipo de solução.

Uma outra solução é fazer com que a unidade móvel e a rede operem assincronamente. O modelo de acesso remoto aplica-se quando o dado está sendo compartilhado e sendo regularmente removido da *cache* via mecanismos de consistência. Na operação de desconexão, a noção tradicional de consistência forte de banco de dados, usando transações atômicas, tem sido modificada para tornar-se menos restritiva. Isto é motivado pelo fato de que os enlaces de comunicação são lentos e caros, e, portanto, torna-se difícil ou economicamente inviável manter cópias de dados de maneira síncrona. Consistência pode ter diferentes níveis dependendo do tipo de conexão estabelecida. Existe, também, uma relação de custo/benefício entre consistência e recursos disponíveis.

2.4 Mobilidade

Um das características da computação móvel é o fato dos sistemas móveis mudarem suas localizações enquanto mantêm a conexão com a rede estática. Visto que os sistemas móveis são capazes de se conectarem com a porção fixa da rede a partir de diferentes localidades, a topologia da rede como um todo muda dinamicamente. Isto implica que algoritmos distribuídos para o ambiente de computação móvel não podem assumir que uma unidade móvel mantém uma localização fixa e conhecida na rede todo o tempo. Conseqüentemente, uma unidade móvel deve primeiro ter sua localização pesquisada antes que uma mensagem possa ser entregue. Portanto, estruturas lógicas, que são muito utilizadas por algoritmos distribuídos, não podem ser estaticamente mapeadas para um conjunto de conexões físicas

dentro da rede.

A mobilidade introduz alguns problemas relacionados com o projeto de protocolos e algoritmos de aplicações distribuídas, gerência de localização e segurança. Na gerência de localização, o custo de pesquisa para localizar uma unidade móvel deve incluir o custo da comunicação. No projeto de protocolos e algoritmos distribuídos, a configuração do sistema passa de estática para dinâmica. Portanto, a carga do sistema e a noção de localização mudam com o tempo.

Outro ponto a ser considerado é que é mais fácil interceptar mensagens no ambiente de computação móvel do que no tradicional, sendo necessário o uso de técnicas de criptografia. Além disso, fazer o rastreamento de uma unidade móvel que está se comunicando com a parte fixa da rede é uma tarefa relativamente fácil, e nem sempre desejável pelos usuários do sistema.

2.5 Heterogenidade

Como uma unidade móvel pode mudar de localidade, ela pode encontrar diferentes ambientes e serviços de rede, e pode necessitar de diferentes protocolos de acesso. Por exemplo, embora todos os principais serviços de uma rede sem fio operem em uma frequência de 800-900 MHz, eles podem usar diferentes métodos de transmissão e modulação. Consequentemente, o sistema dos usuários deve ser capaz de suportar redes com diferentes características. Tais problemas de interoperabilidade podem afetar a mobilidade, já que a conectividade entre os elementos da rede não pode ser sempre garantida. Além disso, o número de unidades móveis em uma célula varia com o tempo, bem como a carga na estação base e a largura de banda.

Outro problema com redes heterogêneas refere-se ao custo de acesso. Muitos serviços de redes sem fios têm uma taxa para os seus serviços que usualmente cobrem um número limitado de mensagens. Cobranças adicionais são feitas por pacotes ou por mensagens. No entanto, o custo para enviar um dado utilizando telefonia celular é baseado no tempo de conexão. Uma vez que serviços diferentes têm custos diferenciados, os custos de uma consulta para um banco de dados centralizado depende da localização do usuário. O custo de uma consulta quando o usuário está conectado a uma rede LAN sem fio pode ser diferente daquele quando se está conectado a uma rede WAN sem fio. Portanto, novos métodos para otimizar a consulta a dados são necessários para manipular os diversos custos de acesso.

2.6 Portabilidade

Muitos dos sistemas móveis são dispositivos portáteis, tais como PDAs (*Personal Digit Assistants*) e computadores pessoais. Entre os vários fatores que devem ser levados em consideração na busca de soluções para aplicações distribuídas, no ambiente de computação móvel, destaca-se o tamanho dos sistemas móveis, que contribui para a mobilidade e portabilidade, mas impõe uma significativa restrição na interface com o usuário, através da limitação do tamanho da tela e do teclado.

A restrição na interface com o usuário afeta a entrada de dados via teclado e a saída via tela. Assim sendo, o ambiente móvel necessita de novos tipos de interfaces com o usuário que não sejam baseadas em teclado e tamanho da tela. Duas possíveis soluções para a entrada de dados são: escritas na tela e voz. A entrada baseada em escrita na tela é atrativa pela sua facilidade de uso e versatilidade. Além disso, o posicionamento da caneta pode ter uma precisão maior com alta resolução do que um cursor. A taxa de reconhecimento de escrita pode ser rápida quando personalizada para um determinado usuário.

A voz também deve ser considerada uma entrada e saída de dados, tendo uma interface atrativa e permitindo ao usuário operar sem a necessidade das mãos ou da visão. O reconhecimento independente do usuário pode ser feito rapidamente. No entanto, voz requer mais processamento e pode criar ruídos e comprometer a privacidade.

Um outro problema que a portabilidade introduz é a vulnerabilidade a situações tais como roubo, perda ou danos físicos. Conseqüentemente, os discos dos sistemas móveis (quando existem) podem não ser considerados confiáveis para armazenar arquivos como *logs* ou estados locais. Portanto, as aplicações distribuídas no ambiente móvel podem necessitar prever o uso de um local de armazenamento confiável alternativo.

2.7 Baixa capacidade de armazenamento

O espaço de armazenamento nos sistemas móveis é limitado pelo tamanho físico da unidade móvel e pela energia requerida. Tradicionalmente, os discos têm grande capacidade de armazenamento não volátil. Entretanto, os discos dos sistemas estáticos nos sistemas móveis consomem mais energia do que os chips de memória, e podem ser voláteis quando transportados de forma não correta. Assim sendo, a maioria dos sistemas atuais não têm discos rígidos.

Trabalhar com um limite de armazenamento não é um problema novo. Soluções incluem compressão de arquivos, acesso a armazenamento remoto via rede, compartilhamento de bibliotecas de códigos, e compressão de páginas de memória virtual.

2.8 Restrição de energia

Devido ao relativo pequeno avanço na tecnologia de baterias, em termos de tempo de carga útil, a energia para os sistemas móveis vem se tornando uma das maiores restrições de recursos. No ambiente de computação móvel, a energia é consumida pela CPU para tarefas de armazenamento, especialmente não volátil como em CD ROM ou disco rígido, pela memória não volátil, pelo *display* e pelo teclado. Além disso, transmissão de dados consome mais energia do que a recepção. O gerenciamento de energia para transmissão de dados é muito importante visto que um sinal deve ser transmitido com um valor correto de potência, para não interferir na recepção de um outro sinal por uma estação, minimizando a relação sinal/ruído [ML98]. Um outro fator a ser considerado é a velocidade de processamento. Quanto maior a latência tolerada no processamento de dados, menor o consumo de energia [AB93]. O consumo de energia é proporcional à capacitância, à oscilação da voltagem e à frequência do relógio. Assim sendo, uma política para reduzi-lo deve considerar a redução dos seguintes pontos:

- Capacitância;
- Voltagem;
- Frequência do relógio.

A atual tecnologia no campo de memória para unidades móveis é RAM dinâmica (DRAM), com consumo de energia de 0.5 watts para um banco de memória de 4M. *Flash EEPROM*, uma tecnologia de armazenamento compacto, não volátil e de baixo consumo de energia, é considerada como uma alternativa mais barata. Memória *Flash* tem uma latência de leitura próxima à DRAM, uma latência de escrita próxima a de um disco, e suporta somente um número limitado de escritas no seu tempo de carga útil. Entretanto, a energia requerida para fazer acesso a memória *flash* é 20 vezes maior do que quando a mesma está sem atividade (*idle*).

Como os bancos de dados podem ser armazenados em unidades estacionárias, as atualizações e consultas podem ser feitas através de conexões sem fios. Uma vez que a transmissão e a recepção de dados consomem uma considerável quantidade de energia, devem ser levados em consideração no projeto de algoritmos e protocolos distribuídos, para o ambiente de computação móvel, os seguintes fatores: técnicas de consultas a dados, a quantidade de dados transmitidos e recebidos, a velocidade de processamento e a latência tolerada.

2.9 Resumo do capítulo

Neste capítulo foram apresentadas as principais características do ambiente de computação móvel. Dentre estas destacam-se:

- Estreita largura de banda;
- Alta latência;
- Desconexão;
- Mobilidade;
- Restrição de energia;
- Baixa capacidade de armazenamento.

As características acima citadas afetam o projeto de protocolos de algoritmos distribuídos para o ambiente composto por unidades móveis, e, portanto, devem ser levados em consideração. As tabelas 2.1 e 2.2 descrevem outros fatores que influenciam o projeto de algoritmos distribuídos para o ambiente móvel, e os fatores que diferenciam este ambiente do tradicional.

Características	Tipo de unidade	
	Estáticas	Móveis
Poder de processamento	Alta	Baixa
Capacidade de armazenamento	Alta	Baixa
Portabilidade	Limitada	Alta
Vulnerabilidade a danos/falhas	Baixa	Alta

Tabela 2.1: Principais diferenças entre o ambiente estático e móvel.

	Tecnologia de Rede	
	Estática	Sem Fio
Banda passante	Alta	Estreita
Confiabilidade	Alta	Baixa
Latência	Baixa	Alta
Custo de uso	Baixo	Alto
Topologia	Fixa, Contínua	Variável, Dinâmica
Condições de uso	Escritório/laboratório	De qualquer local
Mobilidade	Restrita	Alta
Garantia de consistência	Normal	Fraca ou nenhuma
Modo de operação	Conectado	Conectado/Desconectado

Tabela 2.2: Tabela comparativa de unidades móveis e estáticas.

Capítulo 3

Conceitos Básicos

*“Dá instrução ao sábio, e ele se fará mais sábio;
ensina ao justo, e ele crescerá em entendimento”*
(Pv 9.9)

O estado de um sistema distribuído é composto por um conjunto de estados individuais de cada um dos processos participantes. Intuitivamente, um estado de um sistema é consistente se, para todo evento mensagem (m) recebida, existe o evento correspondente mensagem (m) enviada no processo remetente.

A seguir, apresenta-se alguns conceitos de ações atômicas e pontos de recuperação, necessários para o bom entendimento desta dissertação. Além disso, apresenta-se os principais resultados encontrados na literatura sobre ações atômicas e pontos de recuperação no ambiente de computação móvel.

Este capítulo está estruturado da seguinte forma: a seção 3.1 apresenta o modelo físico do sistema utilizado nas soluções propostas nesta dissertação. A seção 3.2 traz conceitos sobre relógios lógicos e, a 3.3 alguns outros sobre armazenamento estável. Na seção 3.4, apresenta-se os conceitos de ações atômicas e, na 3.5, os principais resultados relacionados com tais ações no ambiente móvel. Na seção 3.6 trata-se sobre pontos de recuperação e, na 3.7 os resultados destes no ambiente móvel. Finalmente, na seção 3.8, apresenta-se o resumo do capítulo.

3.1 O modelo do sistema

O modelo físico utilizado nas soluções propostas nesta dissertação é baseado em [BBI94]. Neste modelo, o termo móvel diz respeito à capacidade de mover-se enquanto mantém sua conexão com a rede, e um computador que possui esta capacidade é chamado de unidade

móvel (*mobile host - mh*). Os sistemas que se comunicam diretamente com as unidades móveis são chamados de estações bases. Uma célula é uma área lógica ou geográfica sobre uma estação base. Todas as unidades móveis são associadas a uma estação base e, a esta, são consideradas locais. A estação base representa a localização corrente de uma unidade móvel i na rede, e, portanto, a unidade i pode pertencer a somente uma célula em um dado instante do tempo. Além disso, i está sempre associada a um endereço fixo, seu id , independentemente de sua localização na rede.

Uma unidade móvel i pode comunicar-se diretamente com uma estação base (e vice-versa) somente se está fisicamente localizada dentro da célula da estação base. Caso contrário, a estação base terá que pesquisar na rede a localização desta unidade, e passar a mensagem para a estação base corrente à unidade i . Esta estação, ao receber uma mensagem, a repassa para a unidade i .

Para uma unidade móvel i localizada em uma célula de uma estação base m enviar uma mensagem, por exemplo, para uma unidade base j que não está na mesma célula, ela deve contactar m para que a mesma possa transmitir a mensagem para a estação base n local à célula de j .

O modelo, portanto, consiste de dois conjuntos distintos de entidades: unidades móveis e estações bases. Todos os sistemas fixos e o caminho de comunicação entre eles constituem uma rede estática ou fixa, enquanto cada estação e as unidades móveis locais dentro de uma célula formam uma rede sem fio. A rede fixa garante entrega confiável e na ordem de envio entre quaisquer duas estações bases, com uma latência finita, mas aleatória. Na rede sem fio, as mensagens são entregues na ordem FIFO entre as unidades móveis. A Figura 3.1 abaixo ilustra o modelo do sistema.

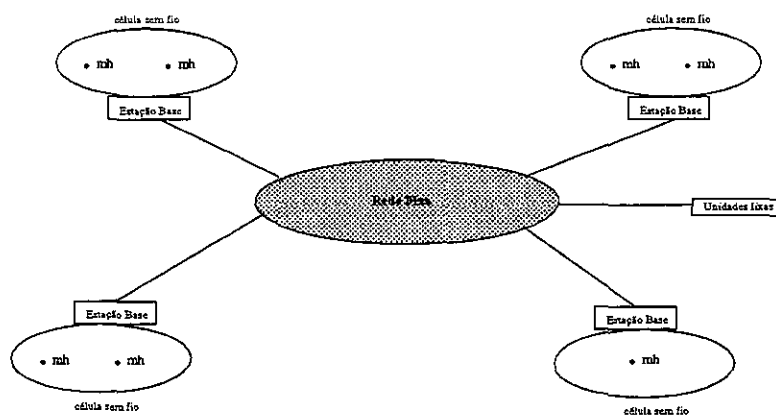


Figura 3.1: Modelo para algoritmos distribuídos

3.2 Relógios lógicos

Para implementação de ações atômicas e pontos de recuperação, é muito importante o estabelecimento de relações temporais entre eventos ocorridos em lugares diferentes. Isto pode ser obtido utilizando o conceito de relógios lógicos.

Um sistema de relógios lógicos consiste de um tempo T , definido sobre um determinado domínio, e um relógio lógico C . Os elementos de T formam um conjunto parcialmente ordenado sobre uma relação ($<$) denominada de *acontece antes* ou *precedência causal*. Intuitivamente esta relação é análoga à noção de *mais cedo que* definida sobre o tempo físico. O relógio C é uma função que mapea um evento ϵ , em um sistema distribuído, para um elemento no domínio de T , denotado por $C(\epsilon)$ e chamado de *timestamp* de ϵ e, definida da seguinte forma: $C : H \mapsto T$. As seguintes propriedades devem ser satisfeitas:

$$\epsilon_1 \mapsto \epsilon_2 \Rightarrow C(\epsilon_1) \leq C(\epsilon_2).$$

Esta propriedade é chamada de *condição de consistência de relógio*. A propriedade abaixo é dita ser *fortemente consistente*, se T e C satisfazem:

$$\epsilon_1 \mapsto \epsilon_2 \Leftrightarrow C(\epsilon_1) < C(\epsilon_2)$$

Implementação de relógios lógicos

Para implementar relógios lógicos, os seguintes fatores devem ser satisfeitos: estrutura de dados local, representando o tempo lógico e, um protocolo (conjunto de regras) para atualizar a estrutura de dados, garantindo a condição de consistência.

A estrutura de dados é mantida por cada processo com as seguintes características:

- Um *relógio lógico local*, denotado por lc_i , utilizado pelo processo p_i para garantir seu progresso;
- um *relógio lógico global*, denotado por gc_i , utilizado pelo processo p_i para obter uma visão do tempo global. Isto possibilita ao processo atribuir *timestamps* consistentes a seus eventos locais.

O protocolo garante que o relógio lógico do processo é mantido consistente. Este é definido pelas seguintes regras:

- *R1*: determina como o relógio lógico local é atualizado por um determinado processo (capturando seu progresso) quando ele executa um evento (envio, recepção ou evento interno).

- *R2*: determina como um processo atualiza seu relógio lógico global, atualizando sua visão do tempo e progresso global. Ela dita quais informações sobre o tempo lógico são anexadas à mensagem e como esta informação é usada na recepção pelo processo, a fim de atualizar sua visão do tempo global.

Os sistemas de relógios lógicos diferem na maneira como representam o tempo lógico e no protocolo de atualização do relógio. Entretanto, todos os sistemas de relógios lógicos implementam as regras *R1* e *R2* e conseqüentemente garantem a propriedade de monotonicidade associada com casualidade.

3.2.1 Tempo escalar

A representação de tempo escalar foi proposta por [CL85] com a finalidade de ordenar totalmente os eventos em um sistema distribuído. O tempo, neste modelo, é representado por um conjunto de inteiros não negativos. Uma variável inteira C_i representa o relógio lógico local de um processo p_i e sua visão do tempo global.

As regras *R1* e *R2* são como descritas abaixo:

- *R1*: antes de executar um evento (envio, recepção ou interno), o processo p_i executa as seguintes ações:

$$C_i := C_i + d \text{ (} d > 0 \text{ sendo o valor de incremento)}$$

(cada vez que *R1* é executada, d pode ter um valor diferente).

- *R2*: no envio de uma mensagem é anexado o valor de relógio do remetente. Quando um processo p_i recebe uma mensagem com *timestamp* C_{msg} , ele executa as seguintes ações:

1. $C_i := \max(C_i, C_{msg})$;
2. Executa *R1*;
3. Entrega a mensagem.

A figura 3.2 mostra uma computação com tempo escalar, sendo $d=1$

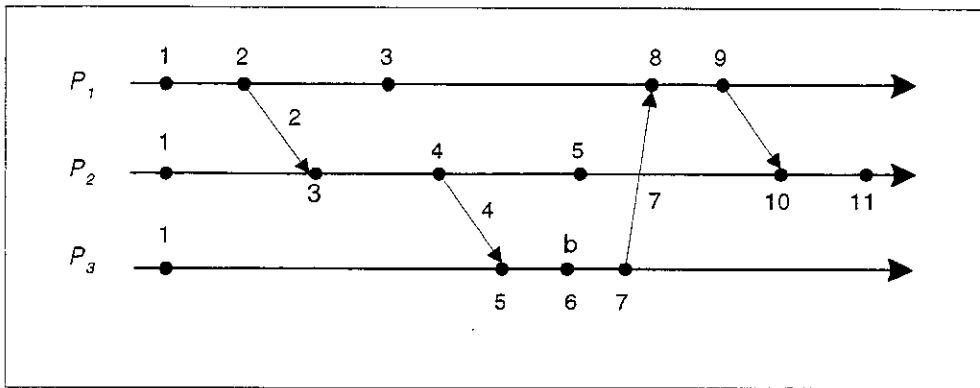


Figura 3.2: Exemplo de uma computação com tempo escalar

Propriedades básicas

A representação de tempo escalar satisfaz a propriedade de consistência. O *timestamp* de um evento é denotado pela tupla (t, i) onde t é o tempo de ocorrência e i é a identidade do processo onde o evento ocorreu. A relação de ordenação total $<$ nos eventos x e y , com *timestamp* (h, i) e (k, j) , respectivamente, é definida como:

$$x < y \Leftrightarrow (h < k \text{ ou } (h = k \text{ e } i < j)).$$

Se o valor de incremento d é 1 (um), o tempo escalar tem a seguinte propriedade: se o evento ϵ tem um *timestamp* h , então $h - 1$ representa a duração mínima (contada em número de eventos), requerida antes do evento ϵ ocorrer. Em outras palavras, $h - 1$ eventos têm ocorridos sequencialmente antes do evento ϵ . Na figura 3.2, cinco eventos precedem o evento b no maior caminho.

O sistema de relógio escalar não é fortemente consistente. A razão disto reside no fato do relógio local e global de um processo serem representados dentro de uma mesma variável, resultando em perda de informação.

3.2.2 Tempo vetorial

No sistema de relógio vetorial, o tempo é representado por um conjunto de vetores de números inteiros não negativos de dimensão n . Cada processo p_i mantém um vetor $vt_i[1..n]$, onde $vt_i[i]$ é o relógio local de p_i e descreve o progresso do processo p_i . $vt_i[j]$ representa o último conhecimento do processo p_i com relação ao tempo local de p_j . Se $vt_i[j]=x$, então o processo p_i sabe que o tempo local do processo p_j tem progredido até x . O vetor vt_i representa a visão do tempo global do processo p_i e é usado para rotular (usando *timestamp*) os eventos.

As regras $R1$ e $R2$ são:

- *R1*: Antes de executar um evento, o processo p_i atualiza seu relógio lógico local, a saber:

$$vt_i[i] := vt_i[i] + d \quad (d > 0)$$

- *R2*: No envio de uma mensagem, m , é anexado o relógio vetorial, vt , do processo remetente. Na recepção de uma mensagem (m, vt) , o processo p_i executa as seguintes ações:

1. Atualiza o relógio lógico global como se segue:

$$1 \leq k \leq n : vt_i[k] := \max(vt_i[k], vt[k]);$$

2. Executa *R1*;
3. Entrega a mensagem m .

O *timestamp*, associado com um evento, é o valor do relógio vetorial do processo onde ele ocorre, quando o mesmo é executado.

Propriedades básicas

Para comparar o *timestamp* de dois vetores vh e vk , as seguintes regras são definidas:

- $vh \leq vk \Leftrightarrow \forall x: vh[x] \leq vk[x]$;
- $vh < vk \Leftrightarrow vh \leq vk \text{ e } \exists x : vh[x] < vk[x]$;
- $vh \parallel vk \Leftrightarrow \text{não } (vh < vk) \text{ e não } (vk < vh)$.

Se dois eventos x e y têm *timestamp* vh e vk , respectivamente, então:

- $x \rightarrow y \Leftrightarrow vh < vk$;
- $x \parallel y \Leftrightarrow vh \parallel vk$.

Se os processos de ocorrência dos eventos são conhecidos, o teste para comparar dois *timestamps* pode ser como a seguir:

Se os eventos x e y respectivamente ocorrem em p_i e p_j , com *timestamp* (vh, i) e (vk, j) , então:

1. $x \rightarrow y \Leftrightarrow vh[i] < vk[j]$

$$2. x \parallel y \Leftrightarrow vh[i] > vk[i] \text{ e } vh[j] < vk[j]$$

O sistema de relógio vetorial é fortemente consistente, ou seja, examinando os *timesteps* de dois vetores pode-se determinar se os eventos são relacionados de forma casual. Entretanto, a dimensão do vetor não pode ser menor do que n .

Se d é igual a 1 (um) na regra $R1$, então o i -ésimo componente do relógio vetorial no processo p_i , $vt_i[i]$, denota o número de eventos que ocorreram em p_i até aquele instante. Assim sendo, se um evento ϵ tem um *timestamp* vh , $vh[j]$ denota o número de eventos executados pelo processo p_j que precede casualmente ϵ . $\sum vh[j] - 1$ representa o número total de eventos que precedem casualmente ϵ na computação distribuída.

3.2.3 Tempo matricial

No sistema de tempo matricial, o tempo é representado por um conjunto de matrizes de dimensão $n \times n$ de números inteiros não negativos. Um processo p_i mantém uma matriz $mt_i[1..n, 1..n]$ onde:

- $mt_i[i, i]$ denota o relógio lógico local de p_i e acompanha o progresso da computação de p_i ;
- $mt_i[k, l]$ representa o conhecimento que o processo p_i tem sobre de p_k , com relação ao relógio lógico de p_l . A matriz mt_i como um todo representa a visão local de p_i com relação ao tempo lógico global.

A linha $mt_i[i, .]$ corresponde ao relógio vetorial $vt_i[.]$ e exhibe todas as propriedades do relógio vetorial.

As regras $R1$ e $R2$ são descritas como:

- $R1$: Antes de executar um evento, o processo p_i atualiza seu relógio lógico, como a seguir:

$$mt_i[i, i] := mt_i[i, i] + d \quad (d > 0);$$

- $R2$: A cada mensagem m enviada é anexado o tempo matricial mt do remetente. Quando o processo p_i recebe uma mensagem (m, mt) enviada por p_j , ele executa as seguintes ações:

1. Atualiza seu tempo lógico como a seguir:

$$1 \leq k \leq n : mt_i[i, k] := \max(mt_i[i, k], mt[j, k])$$

$$1 \leq k, l \leq n : mt_i[k, l] := \max(mt_i[k, l], mt[k, l])$$

2. Executa $R1$;
3. Entrega a mensagem m .

Propriedades básicas

O relógio matricial tem as seguintes propriedades:

$\min_k(mt[k, l]) \geq t \Rightarrow$ o processo p_i sabe que todos os outros processo p_k tem conhecimento do avanço do tempo lógico do processo p_l até t .

Se isto é verdade, o processo p_i sabe que todos os outros processos p_k têm conhecimento de que p_l nunca irá enviar uma mensagem com um tempo lógico menor ou igual a t .

Se d é igual a 1 (um) na regra $R1$, então $mt_i[k, l]$ denota o número de eventos ocorridos em p_l e conhecidos por p_k , bem como por p_i .

3.3 Armazenamento estável

Pontos de recuperação e ações atômicas usam armazenamento estável para salvar os estados locais, os *logs*, e outras informações importantes ao processo de recuperação. Normalmente os discos físicos são usados para implementar a idéia de armazenamento estável. Assim sendo, em sistemas que toleram falhas simples, armazenamento estável pode consistir da memória volátil de outros processos. Um sistema que deseja suportar qualquer tipo de falha transitória pode usar, como armazenamento estável, um disco confiável para armazenar as informações de cada participante. No entanto, os sistemas que suportam falhas não transitórias, devem garantir que informações relacionadas a cada processo sejam sempre armazenadas em um local seguro, fora do local onde os processos estão executando.

3.4 Ações atômicas

Uma atividade é chamada de ação atômica se, durante a execução da ação, o processo que a realiza desconhece a existência de outras atividades e não pode observar qualquer mudança de estado que venha a ocorrer fora da ação. Mais ainda, nenhum outro processo conhece a atividade deste processo, e suas mudanças de estados são ocultas fora da ação. Assim sendo, somente o estado inicial (antes da ação começar a ser executada), ou o estado final, pode ser visto e, nenhum estado interno é visível. Esta é uma propriedade conhecida como “tudo ou nada”, ou seja, uma ação atômica realiza toda a sua computação ou nenhuma das mudanças realizadas terão efeito.

Ações atômicas são de particular interesse na área de banco de dados. Neste contexto, uma ação realizada atômicamente é dita ser uma *transação* (serão usados, no texto, os termos **ações atômicas** e **transação indistintamente**). Ou seja, a execução de um programa ou de uma sequência de operações, que faz acesso ou modifica um banco de dados, é chamada de transação. Uma transação é consistente quando leva o sistema de um estado consistente para outro estado também consistente. Entretanto, transações

sendo executadas concorrentemente podem violar a consistência se suas operações não são coordenadas.

Uma transação geralmente começa com um *begin-transaction* e termina com um *end-transaction*. As ações entre o *begin-transaction* e *end-transaction* compreendem a transação propriamente dita. Um *end-transaction* é freqüentemente precedido por um comando de *commit (validação)*, garantindo que os efeitos da transação serão permanentes no banco de dados. Além disso, a validação de uma transação é uma garantia de que mesmo ocorrendo eventuais falhas as mudanças feitas não serão perdidas.

A figura 3.3 ilustra dois exemplos do conceito de transação.

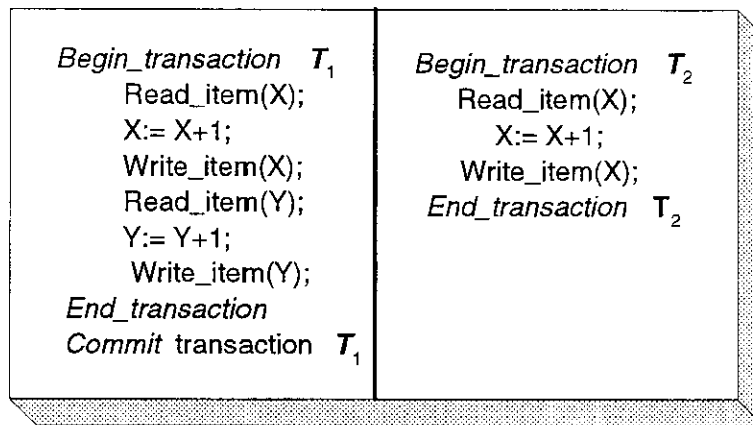


Figura 3.3: Exemplos de transação

Após o sistema ter executado a operação de validação (ou aborto) da transação, a transação é dita ter sido **validada** (ou *abortada*). Uma transação que já iniciou sua execução, mas ainda não foi validada (ou abortada) é dita **ativa**. Caso a execução da transação não possa ser completada por algum motivo, o sistema ou a transação pode emitir uma operação de aborto.

Para suportar ações atômicas, mecanismos devem ser empregados para manipular tanto acessos concorrentes quanto falhas. No caso de acessos concorrentes, os métodos empregados são protocolos de controle de concorrência, enquanto que as técnicas para preservar atomicidade na presença de falhas são freqüentemente chamadas de protocolos de recuperação.

A figura 3.4 ilustra um exemplo de acesso concorrente.

Propriedades de uma transação

Ações atômicas possuem várias propriedades. Estas são freqüentemente chamadas de propriedades ACID, e devem ser levadas em consideração pelos métodos de controle de

Transação T ₁	Transação T ₂	Transação T ₃
<i>Read_item (X);</i> <i>Write_item (X);</i>	<i>Read_item (Z);</i> <i>Read_item (Y);</i> <i>Write_item (Y);</i>	<i>Read_item (Z);</i> <i>Read_item (Y);</i>
<i>Read_item (Y);</i> <i>Write_item (Y);</i>	<i>Read_item (X);</i> <i>Write_item (X);</i>	 <i>Write_item (Y);</i> <i>Write_item (Z);</i>

Figura 3.4: Exemplo de concorrência entre transações

concorrência e recuperação de falhas. As propriedades ACID são descritas da seguinte forma:

- **Atomicidade:** uma transação é uma unidade atômica de processamento, isto é, ela é executada integralmente ou nenhuma de suas mudanças têm efeito no sistema. O mecanismo de recuperação de falhas é responsável por esta propriedade;
- **Consistência:** uma execução de uma transação realizada com sucesso deve levar o banco de dados de um estado inicialmente (antes da execução da ação atômica) consistente, a outro também consistente. Esta propriedade é de responsabilidade do programador ou do mecanismo gerenciador de banco de dados.
- **Isolamento:** os resultados parciais de uma transação não devem ser visíveis a outras transações, até que a transação termine sua execução e suas operações sejam validadas; esta propriedade resolve o problema de atualizações temporárias e evita abortos em cascata. Os mecanismos de controle de concorrência são responsáveis por esta propriedade;
- **Durabilidade:** após uma transação ter modificado o banco de dados e suas mudanças terem sido validadas, estas tornar-se-ão duráveis, ou seja, após uma falha elas poderão ser recuperadas. Esta é de responsabilidade do mecanismo de recuperação.

3.4.1 Ações atômicas em sistemas distribuídos

Informalmente, uma transação distribuída é a execução de um programa ou processo que faz acesso aos dados compartilhados em múltiplos nodos da rede. Assim sendo, os nodos da rede podem utilizar diferentes protocolos de controle de concorrência, e as

transações podem fazer acesso a diferentes nodos. O modelo de ações atômicas em sistemas distribuídos pode ser dividido da seguinte forma:

- **Gerenciador de transações**, que recebe operações do banco de dados e das transações;
- **Gerenciador de escalonamento**, que controla a ordem de execuções das operações;
- **Gerenciador de recuperação**, que é responsável pela validação ou aborto das transações;
- **Gerenciador de *commit***, possibilita aos nodos tomarem uma decisão com relação à validação ou aborto de uma transação.

A figura 3.5 ilustra o modelo descrito acima.

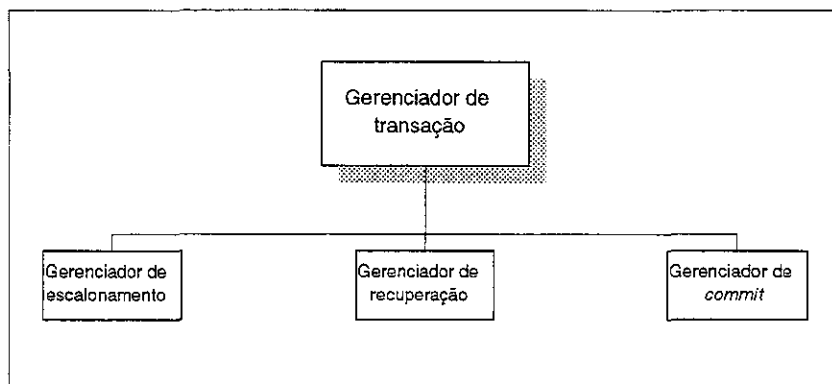


Figura 3.5: Exemplo do modelo de ações atômicas em sistemas distribuídos

Gerenciador de transações

Transações interagem com o banco de dados através do gerenciador de transação. Este recebe operações do banco de dados e das transações, e passa as mesmas para o escalonador. Dependendo do controle de concorrência e protocolos de recuperação que são usados, o gerenciador de transação pode também realizar outras funções. Por exemplo, em um sistema de banco de dados distribuídos, o gerenciador de transações é responsável por determinar quais nodos podem processar cada operação submetida pelas transações.

Gerenciador de escalonamento

Um Escalonador é um programa, ou uma coleção de programas, que controla a execução concorrente das transações. Este controle é realizado restringindo a ordem na qual leituras, escritas, validações e abortos são executados, a partir de diferentes transações. Além disso, escalonador pode evitar abortos em cascata.

Para executar uma operação, uma transação transfere-a para o escalonador. Após receber a operação, este pode tomar uma das seguintes ações:

- **Executar:** a operação do processo é executada e, após seu término, o escalonador transmite os resultados para a transação, caso necessário;
- **Rejeitar:** a requisição do processo para executar a operação é negada, e a transação informada e abortada;
- **Atrasar:** a operação é atrasada e colocada em uma fila, sendo depois removida, podendo ser executada ou rejeitada. Entretanto (enquanto a operação estiver na fila), o escalonador está livre para executar outras operações.

A figura 3.6 ilustra as possíveis ações tomadas por um escalonador.

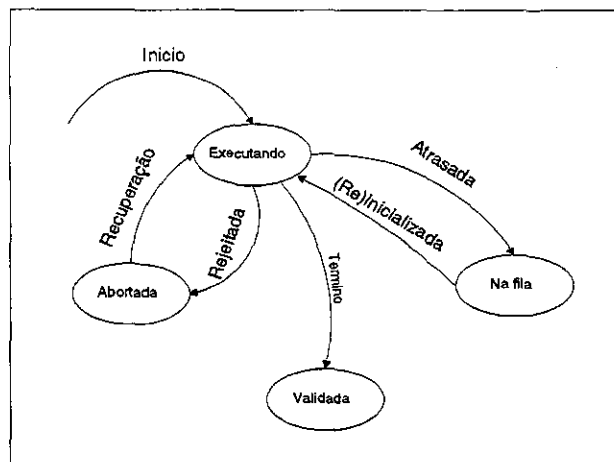


Figura 3.6: Possíveis ações tomadas por um escalonador

Usando estas três ações básicas, o escalonador pode controlar a ordem na qual as operações são executadas. Em um escalonador, a ordem das operações das transações é mantida, mas as operações de diferentes transações podem ser intercaladas.

Uma execução é chamada serial se todas as operações de uma transação ocorrem uma após a outra. Isto significa que a execução de novas transações só ocorre quando a execução das transações anteriores terminarem.

O problema de consistência ocorre quando o intercalamento de operações é permitido, isto é, a execução das transações não é serial. Uma execução não serial pode levar o sistema para um estado inconsistente, no entanto permite acesso a dados compartilhados. Visto que as transações são executadas atômicamente, quaisquer duas transações não devem aparecer serem concorrentes, e os resultados das diferentes transações devem ser vistos como se estas fossem executadas serialmente em alguma ordem. Neste caso, a execução das transações é dita ser serializável, pois é possível obter uma execução serial equivalente. Esta propriedade é conhecida como seriabilidade.

Portanto, o objetivo principal de um escalonador é ordenar operações de modo tal que o resultado da execução das transações seja serializável e recuperável.

Gerenciador de recuperação

O gerenciador de recuperação é responsável por garantir de que o sistema mantém todas as mudanças realizadas pelas transações que foram validadas, e nenhuma mudança realizada pelas transações que foram abortadas.

Uma transação pode ser abortada por diversas razões:

- **Falha do sistema:** o processamento pára e o conteúdo da memória volátil é perdido;
- **Falha de mídia:** embora a memória estável tenha sobrevivido a falhas do sistema, ela torna-se corrompida;
- **Falha da transação:** a transação não consegue terminar sua execução. No entanto, o conteúdo tanto da memória volátil quanto da memória estável, é mantido.

O objetivo dos métodos de recuperação é garantir que se qualquer uma destas falhas ocorrer, o banco de dados é levado para um estado consistente através de recuperação. Qualquer método de recuperação que manipula falhas de sistemas pode manipular falhas da transação. Para manipular falhas de mídia é necessário ter-se cópias redundantes dos dados em diferentes locais (ou discos).

3.4.2 Gerenciador de *commit*

Um *protocolo de commit* é um procedimento cooperativo usado pelo conjunto de nodos nos quais foram feitos acessos por uma transação. Ele possibilita aos nodos tomarem

uma decisão com relação à validação ou aborto de uma transação. Vários protocolos de *commit* têm sido propostos, e uma descrição detalhada destes pode ser encontrada em [EN94], [Vos91], [BHG87]. Abaixo é descrito os dois principais protocolos de *commit*: de duas e de três fases.

Protocolo de *commit* de duas fases

Neste, o nodo (ou o processo executando a transação) onde a transação é iniciada é chamado de *coordenador*. Os outros nodos (ou processos destes nodos), nos quais foram feitos acessos pela transação, são chamados de *participantes*. O protocolo é descrito da seguinte forma, a saber:

- **Primeira fase:** o coordenador envia uma mensagem do tipo *validar* (*commit*) perguntando aos participantes se eles estão prontos para validar a transação ativa. Cada participante envia uma resposta dizendo se está ou não pronto para validar. Se por alguma razão, um dos nodos participantes não estiver pronto para efetivar a validação, ele envia uma mensagem dizendo *não* (ou seja, uma mensagem do tipo *abortar*). Caso contrário responde *sim* (mensagem do tipo *validar*);
- **Segunda fase:** o coordenador coleta os votos de cada participante. Se todos responderam *sim*, o coordenador envia uma mensagem do tipo *validar* para todos. Caso contrário, o coordenador envia *abortar* para todos os que responderam *sim* (aqueles que votaram *não* já decidiram abortar localmente a transação). Cada participante espera por uma resposta do coordenador. Quando os participantes recebem a mensagem eles procuram agir de acordo com o tipo de mensagem recebida.

O protocolo de *commit* de duas fases pode ser modelado como um autômato de estados finitos, como ilustrado na figura 3.7. Os estados do autômato representam os estados do protocolo em um determinado nodo. Uma transição no autômato consiste da recepção de uma ou mais mensagens e o envio ou não de mensagens. Estes autômatos possuem dois possíveis estados finais: estado de validação (*v*), e estado de aborto (*a*). Na figura abaixo o autômato tem 4 (quatro) estados: estado inicial (*i*); estado de espera (*e*); estado de aborto (*a*) e estado de validação (*v*).

Se nenhuma falha ocorrer durante o processo de execução do protocolo, a propriedade de atomicidade é garantida. Entretanto, o protocolo é muito vulnerável a falhas. Em vários estágios, um processo participante (ou coordenador) está esperando por uma mensagem. Se a mensagem não chegar devido a alguma falha no sistema ou outro tipo de falha, o processo ficará bloqueado à espera da mensagem. Para evitar que isso aconteça, ações de *timeout* são adicionadas ao protocolo. O processo muda para outro estado caso o *timeout* ocorra e a mensagem esperada não chegue.

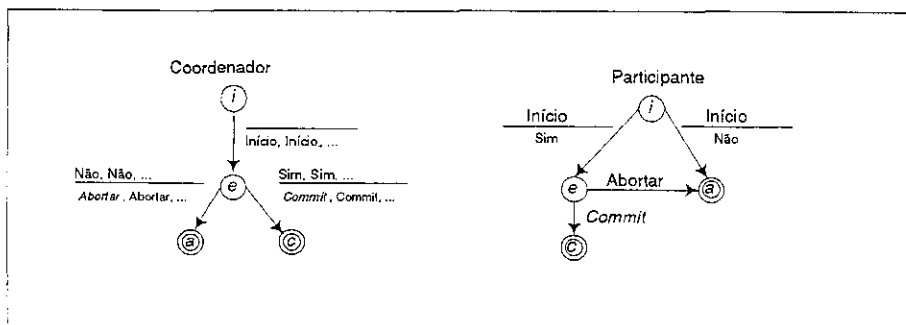


Figura 3.7: Autômato de estados finitos do protocolo de *commit* de duas fases

Protocolo de *commit* de três fases

Um protocolo que nunca bloqueia um nodo, mesmo se os outros nodos falharem, é dito ser *não bloqueante*. Neste, os nodos sempre podem decidir se abortam ou não uma transação. O protocolo de *commit* de duas fases descrito acima é bloqueante. Este fato pode ser resolvido introduzindo-se um estado adicional, chamado de *buffer*, entre os estado de *espera* e *validação*. Em termos de fases, a introdução do estado de *buffer* acrescenta mais uma fase no protocolo, tornando o protocolo de duas fases em um de três. O estado de *buffer* pode ser tratado como um estado de *preparação de validação*. A figura 3.8 ilustra o autômato de estados finitos do protocolo de *commit* de três fases.

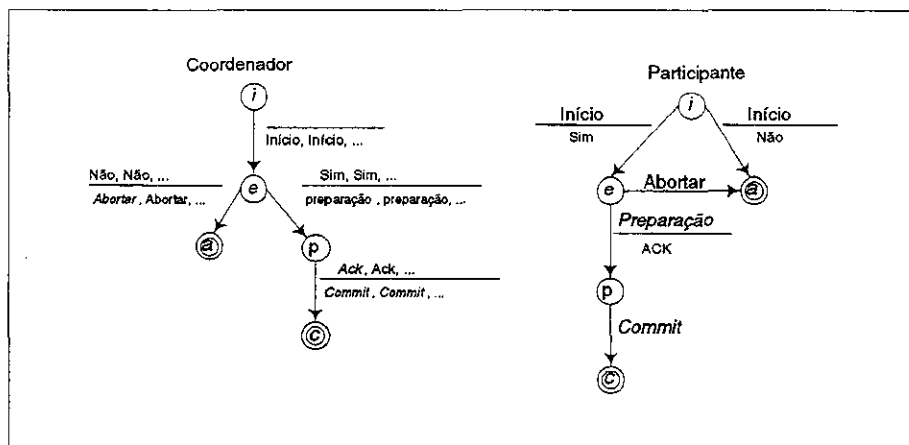


Figura 3.8: Autômato de estados finitos do protocolo de *commit* de três fases

No protocolo de *commit* de três fases, após o coordenador coletar todos os votos *sim*, ao invés de enviar uma mensagem para os nodos participantes validarem a transação, ele envia uma mensagem do tipo *preparação para validação* e passa para o estado *p*. Ao receber a mensagem, cada participante envia uma confirmação do recebimento da mesma

(ou seja envia um *ack*) e passa para o estado *p*. Finalmente, quando o coordenador recebe os *acks* de todos os participantes, ele envia uma mensagem do tipo *validação*.

3.4.3 Controle de concorrência

Como já descrito, escalonadores são programas ou coleções de programas que gerenciam a ordem com que as operações das transações são executadas. No entanto, algumas restrições nas transações e nas suas interações devem ser impostas para que o critério da seriabilidade seja satisfeito. Protocolos que impõem tais restrições são ditos protocolos de controle de concorrência. Uma grande diversidade de protocolos de controle de concorrência têm sido propostos, e uma descrição mais detalhada destes pode ser encontrada em [EN94], [Vos91] e [CDK96]. Vários destes utilizam a técnica de *lock* para evitar que múltiplas transações façam acesso aos mesmos itens de dados simultaneamente. Outros utilizam a técnica de *timestamp*, ou seja, um identificador único para cada transação gerado pelo sistema. Um outro grupo utiliza a técnica de **multiversões**. Além destes, existem ainda os que são baseados no conceito de validação de uma transação, após esta executar suas operações. Esses protocolos são chamados de **otimistas**.

Outro fator que afeta o controle de concorrência é a granularidade dos itens, ou seja, que porção do banco de dados um item representa. Um item pode ser tão pequeno quanto um simples valor, ou tão grande quanto um bloco de um disco rígido, um arquivo ou até mesmo o próprio banco de dados.

Controle de concorrência baseado em *lock*

Uma das principais técnicas usadas para controle de execuções concorrentes de transações é baseada no conceito de bloqueio (*locking*) dos dados. Um *lock* é uma variável associada com o item de dado no sistema e descreve o *status* deste com relação a possíveis operações que podem ser aplicadas a ele. Normalmente, existe um *lock* para cada item de dado. *Lock* é usado com o objetivo de sincronizar o acesso feito pelas transações concorrentes. Vários tipos de *locks* podem ser usados no controle de concorrência:

Lock binário: Um *lock* binário pode ter dois estados ou valores: bloqueado e desbloqueado (1 (um) e 0 (zero)). Se o valor do *lock* no item *X* é 1 (um), então uma operação que o requisite não deve fazer acesso a ele. Caso contrário, se o valor de *X* é 0 (zero), então pode ser feito acesso ao item quando requisitado. O *valor* do *lock* associado com o item *X* será referenciado como **LOCK(*X*)**. Duas operações adicionais, *lock_item* e *unlock_item* devem ser incluídas nas transações quando *lock* binário é utilizado. Uma transação requer acesso ao item *X* emitindo a operação *lock_item(X)*. Se **LOCK(*X*) = 1**, a transação deve esperar. Caso contrário a transação atribui 1 (um) ao valor do *lock* associado ao item

($LOCK(X) = 1$), e faz acesso ao dado. Quando a transação terminar o acesso, ela deve emitir a operação $unlock_item(X)$, o que acarreta $LOCK(X) = 0$ permitindo que qualquer transação faça acesso a X . A figura 3.9 abaixo ilustra o uso de *locks* binários.

```

Lock_item (X):
  B: se LOCK(X) = 0 (*item desbloqueado*)
    então LOCK(X) := 1 (*item bloqueado*)
    senão comece
      espere até (LOCK(X) = 0 e o gerenciador de transações acordar a transação)
      vá para B
    fim;

```

Figura 3.9: Exemplo de *lock* binário

Quando o esquema de *lock* binário é usado, todas as transações devem obedecer às seguintes regras:

1. uma transação T deve emitir a operação $lock_item(X)$ antes de quaisquer operações $read_item(X)$ e $write_item(X)$ serem executadas em T ;
2. uma transação T deve emitir a operação $unlock_item(X)$ após todas suas operações $read_item(X)$ e $write_item(X)$ terminarem;
3. uma transação T não emitirá uma operação $lock_item(X)$ se uma outra transação estiver fazendo acesso ao item X .
4. uma transação T não emitirá uma operação $unlock_item(X)$ a menos que a mesma tenha emitido a operação $lock_item(X)$.

Locks exclusivos e compartilhados: Neste esquema há três operações, a saber: $read_lock(X)$, $write_lock(X)$ e $unlock(X)$. $LOCK(X)$ tem agora três valores associados: $read_locked$, $write_locked$ ou $unlocked$. Um *lock* para leitura ($read_locked$) é também chamado de *lock* compartilhado, visto que outras transações podem ler o mesmo item de dado concorrentemente. Já um *lock* para escrita em um item de dado ($write_locked$) é dito um *lock* exclusivo, visto que somente uma transação pode escrever no item de dado em um determinado instante. Como no caso anterior, as três operações devem ser indivisíveis.

Este esquema é ilustrado na figura 3.10.

Quando o esquema de *locking* múltiplos é usado, deve-se obedecer às seguintes regras:

```

read_lock (X):
B: se LOCK(X) = unlocked (*item desbloqueado*)
    então início LOCK(X) := read_lock ; (*lock para leitura*)
        lendo (X) := 1;
        fim
    senão se LOCK(X) = read_locked
        então lendo (X) := lendo (X) + 1;
        senão início
            espere até (LOCK(X) = unlocked e o gerenciador de transações acordar a transação);
            vá para B
        fim;

write_lock (X)
B: se LOCK(X) = unlocked (* item desbloqueado*)
    então LOCK(X) := write_lock ;
    senão espere até início LOCK(X) = (unlocked e o gerenciador de transações acordar a
transação);
        vá para B
    fim;

unlock_item (X):
B: se LOCK(X) = write_lock
    então início LOCK(X) := unlocked ;
        acorde uma das transações, se houver alguma
    fim
    senão se LOCK(X) = read_locked
    então início lendo(X) := lendo(X) - 1;
        se lendo(X) := 0;
        então início LOCK(X):= unlocked
            acorde uma das transações, se houver alguma
        fim
    fim;
fim;

```

Figura 3.10: Exemplo de locks múltiplos

1. uma transação T deve emitir a operação $read_lock(X)$ ou $write_lock(X)$ antes de qualquer operação $read_item(X)$ ser realizada em T ;
2. uma transação T deve emitir a operação $write_lock(X)$ antes de qualquer $write_item(X)$ ser realizada em T ;
3. uma transação T deve emitir a operação de $unlock(X)$ após todas as operações $read_item(X)$ e $write_item(X)$ terem terminado;
4. uma transação T não emitirá uma operação $read_lock(X)$ se ela já tiver um *lock* de leitura compartilhado ou de escrita exclusiva no item X ;
5. uma transação T não emitirá uma operação $write_lock(X)$ se ela já tem um *lock* de leitura compartilhado ou de escrita exclusiva no item X ;
6. uma transação T não emitirá uma operação $unlock(X)$ a menos que ela tenha um *lock* de leitura compartilhado ou de escrita exclusiva no item X ;

Locking de duas fases: uma transação segue o protocolo de *locking* de duas fases se todas as suas operações ($read_lock$, $write_lock$) precedem a primeira operação $unlock$ na transação. Neste caso, ela (a transação) pode ser dividida em duas fases:

- **Fase de expansão:** durante a qual novos *locks* nos itens podem ser adquiridos, mas nenhum já obtido pode ser liberado;
- **Fase de encolhimento:** durante a qual os *locks* podem ser liberados, mas novos *locks* não podem ser adquiridos.

O mecanismo de *lock* de duas fases pode limitar a concorrência no sistema, visto que uma transação T pode não ser capaz de liberar um item X , após ele ter sido usado, se a transação obter um *lock* de outro item.

Controle de concorrência baseado em *timestamp*

Nesta abordagem, valores (*timestamp*) são atribuídos às transações na ordem em que estas são submetidas ao sistema, ou seja, um *timestamp* define a posição de uma transação no tempo, podendo ser gerado de várias maneiras. Isto impõe uma ordem de execução das transações. Este mecanismo é classificado como pessimista (ou seja, baseado na suposição de que conflitos entre as transações ocorrem com frequência), assim como os baseados em *locking*.

Outros mecanismos de controle de concorrência

Além dos controles de concorrência citados acima, existem ainda os baseados em multi-versões e os otimistas. Uma descrição mais detalhada sobre os mesmos pode ser encontrada em [EN94], [Vos91] e [CDK96].

3.4.4 Transações aninhadas

Uma transação pode ser estruturada como uma árvore possuindo vários níveis. Transações estruturadas desta forma são ditas **transações aninhadas**. Estas são particularmente úteis em sistemas distribuídos, visto que as transações filhas (um nível imediatamente abaixo) podem ser executadas em diferentes servidores. Além disso, transações aninhadas possuem as seguintes características:

- Podem executar concorrentemente com outras transações aninhadas. Isto pode permitir adicional concorrência no sistema;
- Podem abortar ou decidir validar unilateralmente. O aborto de uma transação filha não implica no aborto da transação pai (um nível imediatamente acima) ou raiz (o nível mais externo). A transação raiz pode tomar diferentes ações com relação aos abortos e validações das transações filhas.

3.5 Modelo de ações atômicas no ambiente móvel

Vários autores têm estudado o problema de transações em um ambiente distribuído composto por unidades móveis [Chr93], [Nar94], [LS94], [PB95], [EJB95] e [WC97]. Nesta seção apresenta-se os principais resultados encontrados na literatura relacionados com ações atômica no ambiente móvel.

3.5.1 O modelo de Chrysanthis

[Chr93] propõe um modelo de transações aninhadas abertas usando as noções de *reporting transaction* e co-transação, onde as transações em execução compartilham seus resultados parciais, mantendo seus estados na estação base. Estes dois componentes adicionais (*reporting transaction* e co-transação) mudam de localização, ou seja, realocam suas execuções de uma estação base para outra, à medida que a unidade móvel (onde a transação está em execução) muda de localização.

3.5.2 O modelo de Narasayya

No modelo proposto por [Nar94], os dados armazenados na unidade móvel são tratados como cópias secundárias, enquanto que a parte estática da rede armazena as cópias primárias. A consistência dos dados é mantida pela parte fixa, mas o processamento das transações é feito na unidade móvel usando as cópias secundárias. Para validar uma transação o usuário a envia para o servidor de dados, que irá tentar validá-la. O servidor de dados irá notificar à unidade móvel o resultado do processamento. O modelo baseia-se no fato de que uma validação de transação será, na maioria das vezes, executada.

3.5.3 O modelo de Qi Lu e Satyanarayanan

O modelo proposto por [LS94] (chamado de *Isolation-Only Transaction(IOT)*) é composto por uma sequência de operações de acesso a arquivos delimitada por um $begin_{iot}$ e end_{iot} . A garantia de consistência de um *IOT* depende das condições de conectividade do sistema, sendo que atomicidade com relação a falhas não é garantida. Em [LS94], as propriedades de atomicidade e durabilidade são consideradas não necessárias no ambiente de computação móvel. As transações iniciadas por uma unidade móvel são executadas localmente, sendo que o acesso a dados remotos é realizado através da *cache* local. No modelo, uma transação pode estar em quatro possíveis estados: execução, validação, pendência e resolução. Quando uma transação termina, ela entra no estado de validação ou pendência, dependendo das condições da conexão. Se a transação fez acesso a apenas dados locais, ela é validada e suas modificações tornadas visíveis aos servidores. Caso contrário, a transação entra no estado de pendência, e espera até ser validada de acordo com um conjunto de critérios de consistência. Se a validação ocorrer com sucesso, as mudanças efetuadas pela transação são atualizadas nos servidores. Caso contrário a transação entra no estado de resolução, onde os conflitos são resolvidos manual ou automaticamente, e a transação submetida novamente para validação.

3.5.4 O modelo de Evangelia e Bharat

[PB95] apresenta um modelo estruturado em dois níveis, baseado na noção de *clusters*. Neste, os dados semanticamente relacionados ou próximos são agrupados em um mesmo *cluster* (cuja configuração é dinâmica), e dados não relacionados em *clusters* diferentes. Todos os membros de um determinado *cluster* são mutuamente consistentes, mas certo grau de inconsistência entre membros de diferentes *clusters* é permitido. O grau de consistência pode variar dependendo da banda passante disponível entre os *clusters*. Usuários fazem acesso aos dados localmente consistentes (dentro de um mesmo *cluster*) através de **operações fracas**, e a dados consistentes de forma global (entre *clusters* diferentes)

através de operações estritas.

3.5.5 O modelo de Jing, Bukhres e Elmagarmid

[EJB95] lida com o problema de mobilidade propondo um esquema de gerenciamento de *lock*, no qual um *lock* de leitura em um item de dado pode ser desbloqueado a partir de diferentes cópias deste item. Além disso, o nodo onde o desbloqueio ocorre pode ser diferente daquele onde o *lock* foi bloqueado. Neste modelo, é assumido que os *locks* de leitura são executados imediatamente, quando operações de leituras são realizadas. Assim sendo, as transações sempre buscam cópias consistentes dos dados. [EJB95] implementa este modelo através de um algoritmo otimista de *locking* de duas fases, o qual é chamado O2PL (*Optimistic Two Phase Locking*).

3.5.6 O modelo de Walborn e Chrysanthis

O modelo proposto por [WC97] (chamado de *Pro-Motion*) baseia-se na noção de *compact*, que encapsulam métodos de acesso a dados na *cache*, informações sobre o estado do *compact*, regras de consistência e métodos que fornecem uma interface com a qual a unidade móvel pode gerenciar o *compact*. Ou seja, *compact* é uma requisição à *cache* de dados, acrescentada de obrigações (tais como *deadline*), restrição (tais como um conjunto de operações permitidas) e informações de estado (tais como o número de acessos ao objeto). Assim sendo, *compact* representa um acordo entre o servidor de dados e a unidade móvel. Além disso, o modelo usa uma combinação de técnicas de *cache* e replicação dinâmica, para permitir execuções concorrentes de transações executando nas unidades móveis e fixas. Durante o período de desconexão as transações executando nas unidades móveis podem validar localmente, fazendo as atualizações no servidor de banco de dados após reconexão. O modelo explora a semântica dos objetos sempre que possível, para aumentar o grau de autonomia das unidades móveis e a concorrência. Um subsistema de gerenciamento de transação (consistindo de um gerenciador de *compact* no servidor de dados, um agente de *compact* nas unidades móveis, um gerenciador de mobilidade nas estações bases) é introduzido, para gerenciar e negociar os *compacts* e fornecer um gerenciamento de transação nas unidades móveis.

3.6 Pontos de recuperação

Um outro bom mecanismo, para se garantir consistência de dados em um ambiente distribuído, é o uso de gravação e recuperação de estados globais consistentes. Nesta seção

discute-se a utilização do conceito de pontos de recuperação como um mecanismo para se garantir consistência dos dados em um sistema.

Pontos de recuperação são obtidos periodicamente durante a execução normal de um determinado processo, salvando-se todas as informações necessárias à restauração do mesmo, após uma falha, em um local estável¹ ou seguro. As informações a serem salvas incluem os valores das variáveis no processo, seu ambiente, as informações de controle, os valores dos registros, entre outras. Um estado global consistente é formado de N estados locais, um para cada processo, que formam um estado consistente do sistema. Qualquer estado global pode ser usado para restaurar o sistema após uma falha. No entanto, o estado global consistente mais recente, chamado de linha de recuperação, é obviamente a melhor escolha. Quando um erro é detectado, o processo é reinicializado a partir do seu último estado salvo, restaurando-se o último estado salvo do processo. Este procedimento é chamado de recuperação por retrocesso (*rollback-recovery*).

3.6.1 Estado consistente do sistema

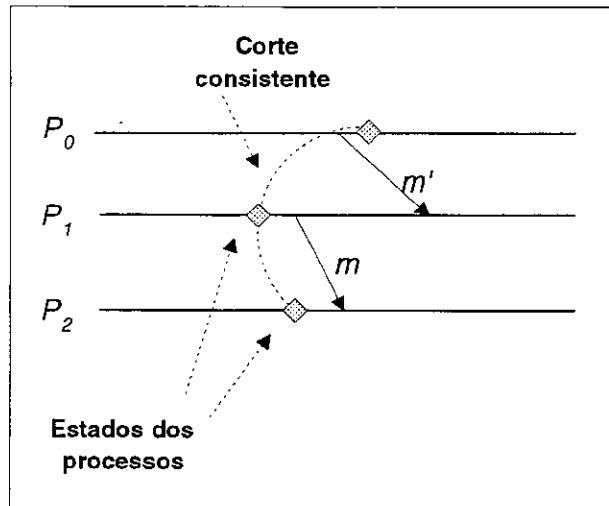
Como mencionado anteriormente, o estado de um sistema distribuído é composto por uma coleção de estados individuais de todos os processos participantes. Intuitivamente, um estado de um sistema é consistente se, para todo evento mensagem (m) recebida, existe o evento correspondente mensagem (m) enviada no processo remetente. O corte na figura 3.11(a) ilustra um estado consistente, enquanto que a figura 3.11(b) mostra um corte inconsistente, visto que o processo P_2 mostra ter recebido uma mensagem m , mas P_1 não reflete o envio da mensagem.

O conjunto de estados locais consistente, um para cada processo, segue as seguintes regras:

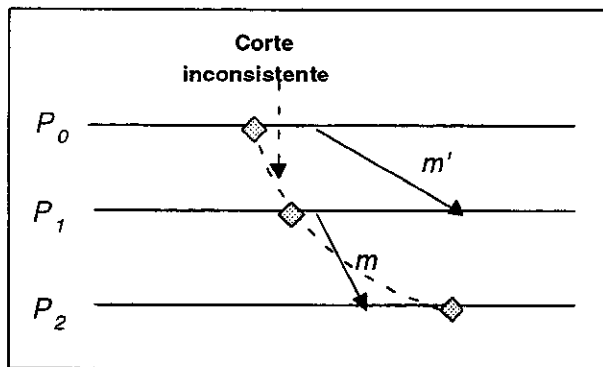
- o conjunto contém exatamente um estado para cada processo;
- não existem eventos de envio de mensagens em um processo P , depois do seu ponto de recuperação, cujo correspondente evento de recepção, em outro processo Q , ocorra antes de seu ponto de recuperação. Ou seja, não existem mensagens órfãs;
- não existe nenhum evento de envio de mensagem em um processo P , antes do seu ponto de recuperação, cujo correspondente evento de recepção, em outro processo Q , ocorra depois do seu ponto de recuperação. Ou seja, não existem mensagens perdidas.

O conjunto que satisfaz as restrições acima é dito um **corte consistente**.

¹Armazenamento estável é definido na subsecção 3.3



(a)



(b)

Figura 3.11: (a) Corte consistente; (b) Corte inconsistente

Um estado inconsistente representa um estado que nunca ocorrerá em uma execução normal do sistema, devido à falhas. Por exemplo, a inconsistência mostrada na figura 3.11(b) pode ocorrer se o processo P_1 falhar após o envio de m para P_2 . O objetivo principal dos protocolos de recuperação por retrocesso é levar o sistema para um estado consistente quando inconsistências ocorrerem devido à falhas.

3.6.2 Efeito dominó

O retrocesso de um processo pode causar uma sequência indefinida de outros retrocessos. Este efeito é denominado *efeito dominó*. Na figura 3.12, as barras em negrito representam pontos de recuperação dos processos. Supondo que ocorra uma falha no processo P_2 , e este retroceda para o ponto de recuperação C , o envio da mensagem m é desfeito. Consequentemente, o processo P_1 deve retroceder para o ponto de recuperação B , desfazendo o evento de recebimento da mensagem m . Isto cancela o envio da mensagem m' e, acarreta no retrocesso do processo P_0 . A linha de recuperação, após uma falha do processo P_2 , consiste dos pontos iniciais de recuperação. Portanto, o sistema deverá retroceder para o início de sua execução, perdendo todo trabalho feito, embora todos os estados locais tenham sido salvos convenientemente. Para evitar o efeito dominó, os processos devem coordenar o estabelecimento de pontos de recuperação tal que, a linha de recuperação avance à medida que novos estados forem sendo salvos.

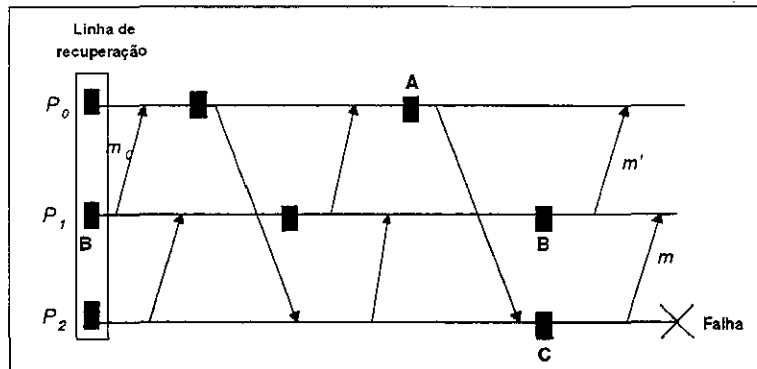


Figura 3.12: Linha de recuperação e efeito dominó

3.6.3 Coleta de lixo

Pontos de recuperação e *logs* consomem uma quantidade considerável de espaço de armazenamento. Parte destas informações podem tornar-se inúteis, para fins de recuperação,

com o passar do tempo. Para descartar estas informações são utilizadas técnicas de **coleta de lixo**. Uma técnica comum e simples é identificar a linha de recuperação e descartar os eventos que ocorreram antes desta linha. Por exemplo, no estabelecimento coordenado de pontos de recuperação, os processos irão sempre reinicializar a partir dos pontos de recuperação mais recentes, e portanto os anteriores podem ser descartados. Coleta de lixo é um mecanismo importante pois pode liberar espaço de armazenamento, embora possa ter um alto *overhead*.

3.6.4 Recuperação por retrocesso baseada no estabelecimento de pontos de recuperação

Recuperação por retrocesso baseada no estabelecimento de pontos de recuperação faz uso do conjunto mais recente de estados locais consistentes (linha de recuperação) para restaurar o estado do sistema após uma falha. Este mecanismo pode ser dividido em três categorias: coordenados, não coordenados e quase síncrono.

Estabelecimento de pontos de recuperação de forma não coordenada

Nos protocolos baseados no estabelecimento de pontos de recuperação de forma assíncrona, os processos salvam seus pontos de recuperação assincronamente. Quando há uma falha, os processos devem encontrar um conjunto consistente de pontos de recuperação, um para cada processo, entre aqueles salvos. O sistema é restaurado, então, para um estado consistente. A principal desvantagem deste mecanismo é a possibilidade do efeito dominó. Uma outra é a grande quantidade de estados locais salvos por processo, e, portanto, um algoritmo de coleta de lixo necessita ser executado periodicamente para liberar espaço de armazenamento.

Estabelecimento de pontos de recuperação de forma coordenada

Nos protocolos baseados no estabelecimento de pontos de recuperação de forma síncrona, os processos coordenam-se durante o processo de gravação dos estados locais, tal que um conjunto de pontos de recuperação formam um corte consistente. A cooperação entre os processos requer comunicação entre eles. Assim sendo, o estabelecimento de pontos de recuperação torna-se mais complexo, mas a restauração é simplificada. Além disso, poucos pontos de recuperação por processo necessitam ser salvos em um dado instante de tempo. Os protocolos síncronos são livres do efeito dominó, visto que os processos sempre reiniciam a partir do estado mais recente. Além disso, a restauração do sistema e o processo de coleta de lixo são simplificados, e o *overhead* de armazenamento é menor do

que nos protocolos assíncronos. A principal desvantagem deste mecanismo é o *overhead* de mensagens durante o estabelecimento de pontos de recuperação.

Estabelecimento de pontos de recuperação de forma quase síncrona

Estabelecimento de pontos de recuperação de forma quase síncrona é um outro mecanismo para evitar o efeito dominó presente nos protocolos assíncronos. Restrições no padrão de pontos de recuperação e na comunicação entre os processos são impostas para garantir o progresso da linha de recuperação. Informações suficientes são adicionadas (*piggybacked*) em cada mensagem, tal que o destinatário possa examiná-la antes seu processamento. Se o processamento desta violar a restrição especificada, o processo destinatário é forçado a salvar seu estado antes do processamento ocorrer. Ao contrário dos protocolos síncronos, nenhuma mensagem de controle necessita ser trocada durante o estabelecimento de pontos de recuperação. Os protocolos quase síncronos podem ser divididos em duas categorias, a saber: (i) baseado em modelo, que mantém uma estrutura de pontos de recuperação e comunicação (como o modelo *MRS* [Rus80]) e, (ii) baseado em índice, que focaliza consistência entre os pontos de recuperação com mesmo índice [EJW96].

3.7 Algoritmos de pontos de recuperação para o ambiente móvel

Recentemente foram propostos dois protocolos para gravação e recuperação de estados globais consistentes em um ambiente móvel utilizando esquemas síncronos, [NF97] e [PM96], e três utilizando esquemas assíncronos, [AB94], [PKV96] e [CdOA98].

3.7.1 O algoritmo de Neves e Fuchs

O algoritmo de [NF97] assume que a unidade móvel possui uma memória estável, segura e que pode participar no processo de recuperação de falha. No protocolo proposto por [NF97], os processos utilizam um relógio local para determinar o momento em que um novo ponto de recuperação deve ser estabelecido. O protocolo utiliza dois tipos diferentes de pontos de recuperação: *soft* e *hard*. Pontos de recuperação *soft* são armazenados na unidade móvel, e são relacionados a falhas como do sistema operacional. Pontos de recuperação *hard* são armazenados na estação base, utilizando o enlace sem fio, e referem-se a falhas que causam problemas sérios à unidade móvel, como danos permanentes na memória. Os processos participantes obtêm seus estados locais sempre que seus relógios locais expiram, independentemente dos outros processos. O protocolo utiliza duas técnicas para manter os relógios sincronizados. Quando uma aplicação inicia, o protocolo ajusta o

relógio em todas as unidades móveis com um valor fixo, chamado de período de ponto de recuperação. Uma vez que os processos não começam a executar no mesmo instante, os relógios irão expirar em diferentes tempos. O protocolo tem um mecanismo que realiza uma nova sincronização, ajustando os relógios durante a execução da aplicação. Cada processo adiciona às mensagens enviadas o intervalo de tempo até o próximo ponto de recuperação. Quando um processo recebe uma mensagem, ele compara seu relógio com o recebido. Se o intervalo de tempo recebido é menor, o processo reinicia seu relógio com o valor recebido.

O protocolo mantém um contador de ponto de recuperação, CN , em cada processo, para garantir que os pontos de recuperação estabelecidos são consistentes. O valor de CN é incrementado sempre que os processos criam um novo ponto de recuperação e é adicionado em toda mensagem enviada.

3.7.2 O algoritmo de Prakash e Sihghal

No esquema proposto por [PM96], só participam do processo de gravação e recuperação de estados globais, as unidades móveis que afetam diretamente, ou indiretamente, o último ponto de recuperação consistente. As outras unidades móveis não participam do processo. A informação de dependência entre as unidades móveis é codificada como um vetor de *bits*, ao ser transmitida em uma mensagem do sistema. O algoritmo de Prakash e Sihghal minimiza a comunicação no enlace sem fio, visto que limita o número de unidades móveis que participam do processo de detecção de estado global consistente.

3.7.3 O algoritmo de Acharya e Badrinath

O algoritmo desenvolvido por [AB94] não requer um controle explícito das mensagens enviadas para as unidades móveis, evitando com isso o *overhead* do mecanismo síncrono. Ao invés disso, a cada mensagem é adicionada uma informação de controle, e cada unidade móvel grava seu estado local assincronamente. O protocolo utiliza uma regra chamada de duas fases, que determina quando uma unidade móvel deve gravar seu estado local.

Associado a cada unidade móvel i existem dois vetores, $CKPT_i[1..N]$ e $LOC_i[1..N]$, ambos armazenados na estação base local à unidade i . $CKPT_i[i]$ é inicializado com 1, e as outras posições do vetor com zero. Sempre que i gravar seu estado local, ela incrementa o $CKPT_i[i]$. Portanto, $CKPT_i[i]$ sempre armazena o número do próximo estado local de i a ser gravado. $LOC_i[i]$ armazena o *id* da estação base local à i . Este vetor muda sempre que i se mover para uma nova célula.

3.7.4 O algoritmo de Pradhan, Krishna e Vaidya

[PKV96] utiliza duas estratégias para salvar os estados dos processos: *log* imediato (*No Logging (N)*) e *log* não imediato (*Logging (L)*). Além disso, [PKV96] propõem três esquemas de *handoff*, a saber: *Pessimista (P)*, *Lazy(L)* e *Trickle(T)*. As duas estratégias e os três mecanismos de *handoff* definem seis combinações de algoritmos de recuperação de falhas, que variam de acordo com a qualidade da rede.

No esquema *No Logging*, o estado do processo pode ser alterado na recepção de uma mensagem ou numa entrada de dados pelo usuário. As mensagens ou as entradas de dados pelo usuário que modificam o estado são chamadas de eventos de escrita. Se, no pior caso, a semântica da mensagem não é conhecida, é assumido que o estado é alterado na recepção de toda mensagem ou na entrada de dados pelo usuário. Neste esquema, o estado da unidade móvel é salvo na estação base sempre que acontecer um evento de escrita no dado da unidade móvel. Após uma falha, a unidade móvel envia uma mensagem para a estação base, que então transmite o último estado da unidade móvel. O fator limitante deste esquema são as frequentes transmissões de estados sobre o enlace sem fio.

No esquema *Logging*, também chamado de esquema pessimista, e usado nos sistemas estáticos, uma unidade móvel salva seu estado local periodicamente. Para facilitar a recuperação de falha os eventos de escrita são colocados em um arquivo *log*. Se uma mensagem de escrita é recebida, a estação base primeiro coloca a mesma no arquivo *log*, e passa esta para a unidade móvel. De forma similar, na entrada de dados pelo usuário, a unidade móvel primeiro passa uma cópia da mesma para a estação base, a fim de ser armazenada no *log*. Após colocar a cópia dos dados no *log*, a estação base envia uma confirmação para a unidade móvel. A unidade móvel pode processar a entrada, mas não pode enviar uma resposta enquanto não receber uma confirmação da estação base. Esse esquema garante que nenhuma mensagem ou entrada de dados pelo usuário será perdida devido a uma falha da unidade móvel [PKV96]. Os eventos de escrita são colocados no *log* até um novo ponto de recuperação ser enviado para a estação base, que ao recebê-lo retira do *log* os antigos eventos de escrita. Após uma falha, a unidade móvel envia uma mensagem para a estação base, a qual então envia o último ponto de recuperação da unidade móvel, bem como o *log* contendo os eventos de escrita.

3.8 Resumo do capítulo

Um estado é consistente se pode ter existido em uma execução do sistema livre de falhas. Mesmo para um dado estado inicial, a execução de um sistema distribuído pode seguir diferentes sequências de estados, possibilitando diferentes estados consistentes, alguns dos quais podem não ter ocorrido durante a execução. O uso de ações atômicas e pontos de

recuperação são dois bons mecanismos para se obter consistência de dados.

Relógios lógicos

Na implementação de ações atômicas e pontos de recuperação é utilizado o conceito de relógios lógicos. Um sistema de relógios lógicos consiste de um tempo T , definido sobre um determinado domínio, e um relógio lógico C . Os principais esquemas de relógios lógicos são:

- Tempo escalar;
- Tempo vetorial;
- Tempo matricial.

Armazenamento estável

Pontos de recuperação e ações atômicas usam o conceito de armazenamento estável para salvar os estados locais e outras informações importantes. Armazenamento estável pode ser implementado usando os discos físicos ou a memória volátil.

Ações atômicas

Uma ação é atômica se, durante a execução da ação, o processo a realizá-la desconhece a existência de outras atividades e não pode observar qualquer mudança de estado que possa ocorrer fora da ação. A execução de um programa ou seqüências de operações que fazem acesso ou modificam um banco de dados é chamada de transação. Uma transação é consistente quando leva o sistema de um estado consistente para outro estado também consistente. Transações possuem várias propriedades. Estas são freqüentemente chamadas de propriedades ACID:

- **Atomicidade;**
- **Consistência;**
- **Isolamento;**
- **Durabilidade.**

Protocolos de controle de concorrência são usados para garantir consistência na presença de execuções concorrentes de transações. Os principais componentes de controle de concorrência são:

- **Gerenciador de transações**, que recebe operações do banco de dados e das transações;
- **Escalonador**, que controla a ordem de execuções das operações;
- **Gerenciador de recuperação**, que é responsável pela validação ou aborto das transações;
- **Gerenciador de *commit***, possibilita aos nodos tomarem uma decisão com relação à validação ou aborto de uma transação.

A figura 3.13 ilustra os possíveis estados de uma transação.

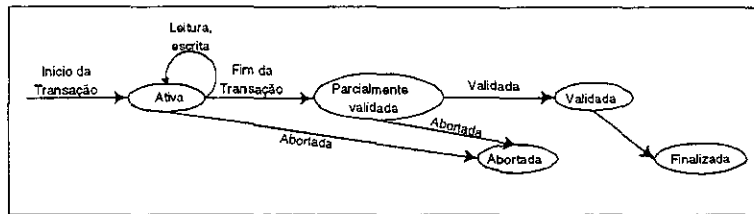


Figura 3.13: Possíveis estados de uma transação

Transações podem ser estruturadas como conjunto de transações aninhadas. Estas subtransações podem executar concorrente ou sequencialmente, e em outros nodos da rede. A transação raiz pode tomar diferentes ações com relação aos abortos ou validações das subtransações.

As características intrínsecas ao ambiente de computação móvel, principalmente desconexão, latência e banda passante, tornam os tradicionais modelos de validação de transação inadequados ao ambiente de computação móvel. Vários protocolos de validação de transação para este ambiente têm sido propostos, [Chr93], [Nar94], [LS94], [PB95], [EJB95] e [WC97].

Pontos de recuperação

Um outro bom mecanismo para se garantir consistência de dados em ambiente distribuído, composto por unidade móveis, é o uso de gravação e recuperação de estados globais consistentes. Os mecanismos de recuperação por retrocesso baseados em pontos de recuperação, podem ser classificados em: coordenados ou síncronos, não coordenados ou assíncronos e quase síncronos. Nos protocolos síncronos, os processos coordenam-se durante o processo de gravação dos estados locais. Já nos protocolos assíncronos, os processos salvam os seus estados independentemente, enquanto que nos quase síncronos, restrições no padrão de

pontos de recuperação e na comunicação entre os processos são impostas para garantir o progresso da linha de recuperação.

Recentemente foram propostos dois protocolos para gravação e recuperação de estados globais em sistemas móveis utilizando esquemas síncronos, [NF97] e [PM96], e três utilizando esquemas assíncronos, [AB94], [PKV96] e [CdOA98]. As tabelas 3.1 e 3.2, obtidas em [ML98], ilustram as principais características destes protocolos.

	Tipo de conectividade		
Tipo de Registro (<i>log</i>)	Conectado	Desconectado	Conectado às vezes
	Imediato ou Periódico	Periódico	Periódico
Mecanismo	Síncrono/Assíncrono	Assíncrono	Assíncrono

Tabela 3.1: Estabelecimento de pontos de recuperação no ambiente móvel

	Algoritmos propostos				
Registro (<i>log</i>)	Acharya	Vaidya 1	Vaidya 2	Neves	Prakash
	Imediato	Imediato	Periódico	Periódico	Periódico
Pontos de recuperação	<i>Hard</i>	<i>Hard</i>	<i>Hard</i>	<i>Hard e Soft</i>	<i>Hard</i>
Mecanismo	Assíncrono	Assíncrono	Assíncrono	Síncrono	Síncrono
Sincronização	–	–	–	<i>Timestamp</i>	Mensagens
Armazenamento	Não Confiável	Não Confiável	Não Confiável	Confiável	Confiável

Tabela 3.2: Comparação entre os algoritmos de gravação e recuperação de estados globais no ambiente móvel

Capítulo 4

Consistência de dados usando ações atômicas

*“Não desampares a sabedoria, e ela te guardará;
ame-a, e ela te conservará”
(Pv 4.6)*

4.1 Introdução

A validação de transação em um ambiente distribuído móvel é afetada, principalmente, pela desconexão e o meio de comunicação sem fio. A maioria das desconexões das unidades móveis são voluntárias por uma razão de custo financeiro ou consumo de energia. Manipulação de desconexão tem sido extensivamente discutida dentro do contexto de particionamento de rede. Soma-se a isso o fato do meio de comunicação sem fio ter como característica possuir atrasos consideráveis.

Diante destes fatos, apresenta-se um novo modelo de execução de transação que utiliza a noção de *proxy*. Através deste, uma unidade móvel pode submeter operações e validações de transação enquanto conectada, antes de se desconectar e logo após sua reconexão, além de gerenciar os *locks* mantidos por esta durante o período de desconexão. O modelo lida, também, com a execução de transações aninhadas. Portanto, a solução apresentada é adequada ao ambiente de computação móvel visto que, permite a execução de transações aninhadas, proporciona autonomia às unidades móveis para executarem transações durante o período de desconexão, e, conseqüentemente, alto grau de concorrência. Além disso, minimiza o uso da estreita banda passante, reduzindo o consumo de energia e custo financeiro.

Este capítulo está estruturado da seguinte forma: Na seção 4.2 discute-se alguns problemas relacionados aos modelos tradicionais de transação, e na subseção 4.2.1 a desconexão das unidades móveis no contexto das transações móveis. Na seção 4.3 descreve-se o modelo proposto, e na subseção 4.3.4 a execução de transações aninhadas dentro do modelo. Na seção 4.4 apresenta-se o resumo do capítulo.

4.2 Problemas com os esquemas atuais

Uma transação no ambiente móvel é uma ação atômica distribuída, onde parte da computação é executada nas unidades móveis e uma outra parte nas unidades estáticas. As características intrínsecas dos sistemas móveis (e.g., meio de comunicação sem fio, possibilidade de desconexão e mobilidade dos dados) afetam o processamento da transação. Transações atômicas fazem acesso aos dados compartilhados em um ambiente distribuído, para que se possa preservar a consistência na presença de concorrência e falhas. O objetivo principal da computação móvel é fornecer aos usuários um ambiente composto de um conjunto de serviços comparáveis aos existentes na computação distribuída convencional. Assim sendo, os usuários dos sistemas móveis desejam fazer acesso aos dados locais e remotos, armazenados nas unidades móveis e nas estações bases, fazendo consultas e atualizações, utilizando as redes sem fio e tradicionais. Além disso, para que estes usuários realizem as suas tarefas durante o período de desconexão, é necessário que as unidades móveis tenham um certo grau de autonomia no gerenciamento das transações [AK93]. Visto que o meio de comunicação sem fio possui atrasos consideráveis, é necessário também que as unidades móveis minimizem o uso do enlace sem fio, reduzindo, portanto, o consumo de energia e o custo financeiro. O modelo tradicional de execução de transação é inadequado ao ambiente móvel devido à incapacidade de lidar adequadamente com desconexão de unidades móveis.

4.2.1 Desconexão

Quando a desconexão não é voluntária, esta pode ser tratada como uma falha. Neste sentido, as soluções para manipular transações na presença de falhas no ambiente de computação móvel não diferem muito das soluções já conhecidas, e, portanto, qualquer solução que se apresente será uma variação ou adaptação das soluções tradicionais já conhecidas. As soluções tornam-se diferentes quando desconexão é tratada como uma falha preparada, ou seja, voluntária. Portanto, as frequentes desconexões das unidades móveis implicam que o sistema deve ser capaz de tomar ações especiais com relação às transações ativas antes da desconexão. Diante deste contexto, pode-se levantar duas questões igualmente importantes. Uma vez que o usuário sabe que vai se desconectar,

quais ações podem ser tomadas antecipadamente, por ele e pelo sistema, a fim de continuar o processamento das transações e a operação do sistema, no período de desconexão? Quais as vantagens e desvantagens para o usuário e para o ambiente de computação móvel? Uma boa solução para estas questões é as unidades móveis transferirem algumas informações para a parte estática da rede enquanto conectadas. Além disso, a parte estática da rede pode tomar ações especiais com relação às transações no lugar de uma unidade móvel, não somente no período de desconexão mas também durante a conexão. Basicamente, esta é a estratégia utilizada no modelo proposto.

4.3 O modelo proposto

Na solução apresentada, uma unidade móvel armazena tanto dados locais quanto remotos, sendo que uma cópia dos dados locais é armazenada na sua estação base, como uma cópia secundária, a partir da qual as outras unidades podem fazer acesso. Como já mencionado, trata-se apenas de desconexão voluntária, assumindo os outros tipos como uma falha, onde se pode aplicar o modelo tradicional de transação. No modelo, a unidade móvel designa uma unidade estática disponível como seu *proxy*, o qual se torna responsável pelas validações das transações submetidas pela mesma. Portanto, sempre que uma unidade móvel desejar validar uma transação, seja durante o período de conexão ou após reconectar-se, ela deve delegar todo o controle da execução da transação a seu *proxy*. Note que o *proxy* é responsável, também, pelo gerenciamento dos *locks* da unidade móvel durante o tempo que esta permanece desconectada. Além disso, sempre que uma unidade móvel se mover ou se reconectar em uma nova célula, o sistema informa à estação base anterior a nova localização da unidade móvel. A unidade móvel possui, ainda, um gerenciador de transações e controle de concorrência. A figura 4.1 ilustra o modelo.

A seguir são analisadas três situações possíveis, nas quais uma unidade móvel pode submeter operações e validações de transações, a saber:

- a unidade móvel está conectada;
- a unidade móvel vai desconectar-se;
- a unidade móvel está desconectada.

Sempre que a unidade móvel for se desconectar, ela deve transferir o estado das transações sendo executadas para o seu *proxy*: os *locks* mantidos, os *ids* das transações, os valores (ou versões) dos dados atualizados, o número de transações filhas e seus estados (no caso de execução de transações aninhadas).

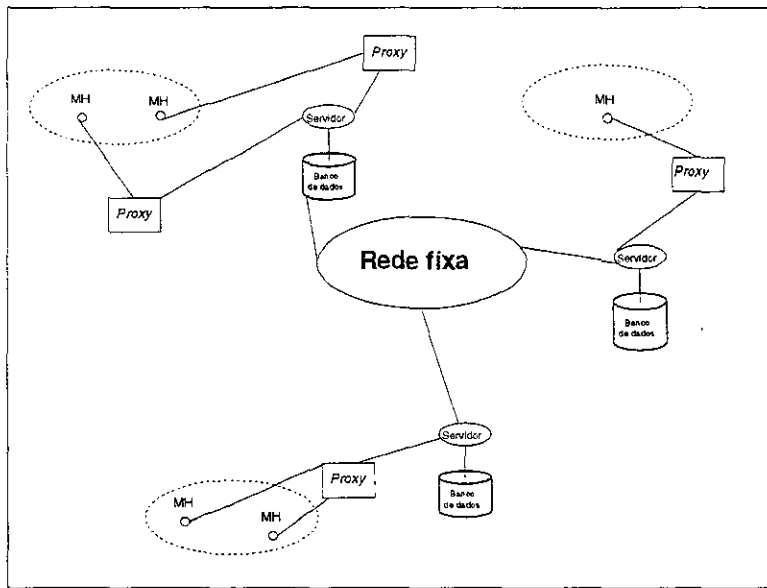


Figura 4.1: Modelo proposto de execução de transação

4.3.1 Unidade móvel conectada

No primeiro caso, ou seja, quando uma unidade móvel está conectada, o processamento das operações é feito na unidade móvel e, portanto, o *proxy* não mantém dados replicados com o intuito de servir como fonte de recuperação (no caso de uma falha), ou substituição (no caso de uma desconexão). No entanto, as validações das transações são submetidas ao servidor de dados através do *proxy* da unidade móvel. Logo, após uma transação ter sido submetida para validação, toda computação necessária é feita entre o *proxy* e o servidor, evitando com isso o uso excessivo do enlace sem fio, e, conseqüentemente, o uso da estreita banda passante. Após obter o resultado das validações, o *proxy* envia-o para a unidade móvel.

4.3.2 Unidade móvel vai desconectar-se

No segundo caso, dois fatos podem ocorrer:

- (i) a unidade móvel vai desconectar-se sem antes submeter quaisquer transações para validação;
- (ii) e/ou a unidade móvel vai desconectar-se submetendo algumas transações para validação.

Transações em processamento

No caso (i), a unidade móvel deve transferir para seu *proxy* os *locks* obtidos pelas transações sendo executadas. Além disso, deve passar também, os valores e versões dos dados já validados.

Transações a serem validadas

Já no caso (ii), similar ao da unidade móvel estar conectada, as validações de transações são feitas através do *proxy* da unidade móvel. Neste caso, portanto, a unidade móvel deve transferir o controle das transações ativas para o seu *proxy*. Este irá negociar a validação da transação na parte estática da rede, da mesma forma como descrito na subseção 4.3.1. Para cada transação que for validada, a mudança é propagada para a estação base e a unidade móvel informada quando se reconectar. Caso contrário, para cada transação que for abortada, uma mensagem é enviada para a unidade móvel informando o aborto. Durante o período de desconexão a unidade móvel trabalha assumindo que a transação foi validada.

Note que os dois casos, (i) e (ii), podem ocorrer simultaneamente, e assim sendo todas as ações descritas acima são executadas.

4.3.3 Unidade móvel desconectada

Durante o período de desconexão, a unidade móvel pode iniciar outras transações e algumas podem ser validadas no contexto da unidade móvel. Nesta situação, a transação tem uma **validação provisória**. O *proxy* tentará validar todas as transações executadas durante o período de desconexão, bem como transformar as validações provisórias em permanentes. Após um **período máximo de desconexão** estabelecido pelo sistema, a unidade móvel deve reconectar-se à rede transferindo as transações para seu *proxy*. Após a reconexão, tal unidade pode iniciar novas transações, desde que os dados manipulados por estas não dependam daquelas ainda sendo validadas. Com isso, evita-se abortos em cascata. Caso a unidade móvel não reconecte no período estabelecido, todos os seus *locks* mantidos no *proxy* são retirados, e todas as transações executadas durante o período de desconexão são abortadas.

4.3.4 Transações aninhadas

Tendo em vista as restrições intrínsecas do ambiente móvel, o modelo de transações aninhadas¹ fornece uma flexibilidade maior para este, no suporte de divisão da computação

¹Uma breve descrição de transações aninhadas podem ser vista na subseção 3.4.4

e falhas parciais, onde diferentes estratégias de recuperação podem ser aplicadas, minimizando os efeitos das falhas. Em comparação com o modelo de transação em um só nível, o modelo de transações aninhadas é, neste sentido, mais robusto.

No modelo proposto, dois casos podem acontecer quando transações aninhadas são usadas:

- (i) uma ou mais subtransações são executadas na unidade móvel;
- (ii) a transação raiz está sendo executada na unidade móvel e algumas, ou todas, subtransações na parte fixa da rede.

O caso (i) é similar aos descritos nas subseções 4.3.1, 4.3.2 e 4.3.3. No entanto, supondo que algumas transações são executadas no período de desconexão, dois novos fatos podem ocorrer. O primeiro caso dá-se quando a transação raiz, sendo executada na parte estática da rede, é abortada. Por conseguinte, quando a unidade móvel se reconectar, o seu *proxy* a informa do aborto da transação aninhada, e suas subtransações são, portanto, abortadas. O segundo caso ocorre quando uma ou mais subtransações, sendo executadas na unidade móvel durante o período de desconexão, abortam. Neste caso, ao reconectar-se, a unidade móvel deve informar ao *proxy* os abortos ocorridos. Note que a decisão, se a transação aninhada vai ou não abortar, é tomada na parte estática da rede.

No caso (ii), o *proxy* da unidade móvel gerencia as subtransações sendo executadas na parte fixa da rede. Portanto, quando as subtransações forem validadas, os objetos modificados são transferidos para o *proxy* da unidade móvel, bem com os estados das subtransações. Os *locks* obtidos pelas subtransações são, também, transferidos para o *proxy*, que mantém ainda, informações das subtransações abortadas. Por outro lado, três casos podem novamente acontecer quando a unidade móvel se encontra:

- conectada;
- a ponto de desconectar-se;
- desconectada.

Unidade móvel conectada

Assumindo que, no modelo de transações aninhadas, as transações filhas têm acesso aos *locks* obtidos pelos seus ancestrais, os *locks* mantidos pela transação raiz são transferidos para o seu *proxy*. Quando a transação raiz decide validar, a unidade móvel transfere o controle da execução da transação para o *proxy*, que irá negociar na parte estática da rede a sua validação. Portanto, caso uma das subtransações tenha abortado, a decisão, se a transação vai ser validada ou não, é tomada no *proxy*. Caso seja validada, a mudança é

propagada e a unidade móvel notificada. Caso contrário, a unidade móvel é informada que a transação abortou.

Unidade móvel vai desconectar-se

Similarmente à subseção 4.3.2, dois fatos podem ocorrer neste caso:

- (i) a unidade móvel vai desconectar-se, sem antes submeter a transação raiz para validação;
- (ii) a unidade móvel vai desconectar-se, submetendo antes da desconexão a transação raiz para validação.

No caso (i) a unidade móvel deve, antes de desconectar-se, transferir os *locks* obtidos e ainda não transferidos para o *proxy*. Além disso, a unidade móvel transfere o estado mais recente da transação raiz, bem como os valores e versões dos dados já validados.

No caso (ii) a unidade móvel transfere a transação raiz, bem como todos os *locks* ainda não transferidos, para o *proxy*. Logo, o controle da execução da transação passa a ser de responsabilidade do *proxy*, e a unidade móvel trabalha, durante o período de desconexão, assumindo que a transação foi validada. Caso a transação seja abortada, o *proxy* irá informar à unidade móvel quando esta se reconectar. Por conseguinte, todas as subtransações executadas no período de desconexão, pertencentes à transação aninhada são abortadas.

Em ambos os casos, sempre que a unidade móvel for se desconectar, o estado da transação raiz é enviado para o *proxy*. Assim sendo, as subtransações obtêm informações sobre o estado da transação raiz no *proxy* da unidade móvel durante o período de desconexão. Após reconectar-se, a unidade móvel transfere o estado mais recente da transação raiz.

Unidade móvel desconectada

Como descrito na subseção 4.3.3, a unidade móvel pode iniciar novas transações, bem como validá-las. Quando a unidade móvel se reconectar, ela deve transferir para o *proxy* a transação raiz, caso a mesma tenha sido executada durante o período de desconexão, bem como seu estado e todas as outras transações executadas. O *proxy* tentará validar todas as transações executadas naquele período. Após reconectar-se, a unidade móvel pode iniciar outras transações desde que os dados manipulados por estas não dependam das transações cujas validações ainda estejam sendo negociadas pelo *proxy*. Tal procedimento minimiza o número de abortos em cascata. Durante todo o tempo de desconexão, a informação sobre o estado da transação raiz obtida pelas subtransações é a última passada antes da desconexão da unidade móvel. Caso a unidade móvel não se reconecte durante o período

máximo de desconexão estabelecido pelo sistema, todas as subtransações serão abortadas. Quando se der a reconexão, a unidade móvel será informada dos abortos ocorridos e a transação raiz e todos seus descendentes também serão abortados.

4.4 Resumo do capítulo

Neste capítulo apresentou-se um modelo otimista de validação de transação em um ambiente de computação móvel, o qual é baseado na noção de *proxy*. O *proxy* de uma unidade móvel é responsável pelas validações das transações e gerenciamento dos *locks* durante o período de desconexão. No modelo, são analisados três casos possíveis no qual uma unidade móvel pode submeter validações de transações: estando conectada, antes de se desconectar e quando está desconectada. Em todos os três casos o processamento da transação é feito na unidade móvel. No entanto, as validações são submetidas através do *proxy* da unidade móvel. O segundo caso é subdividido em dois outros casos, a saber: a unidade móvel vai desconectar-se, sem antes da desconexão submeter quaisquer transações para validação, e/ou a unidade móvel vai se desconectar, submetendo antes da desconexão algumas transações para validação. Durante o período de desconexão, a unidade móvel pode iniciar novas transações bem como validá-las. As validações de transações que ocorrem durante o período de desconexão são ditas *validações transitórias* e devem tornar-se permanentes quando a unidade móvel for reconectada.

Toda computação necessária para validação de uma transação é feita entre o *proxy* da unidade móvel e o servidor de banco de dados, evitando o uso do enlace sem fio e, conseqüentemente, o uso da estreita banda passante. O modelo pode ser aplicado, também, na execução de transações aninhadas. Desta forma, argumenta-se que o modelo proposto é adequado ao ambiente de computação móvel visto que, permite às unidades móveis executarem e validarem transações antes e durante o período de desconexão, proporcionando-lhes autonomia e alto grau de concorrência, minimizando o uso da estreita banda passante e, conseqüentemente, reduzindo o consumo de energia e o custo financeiro.

Capítulo 5

Consistência de dados usando pontos de recuperação

*“O entendimento, para aqueles que o possuem, é uma fonte de vida”
(Pv 16.22)*

5.1 Introdução

Um outro mecanismo para se conseguir consistência de dados em um ambiente de computação móvel é o uso de gravação e recuperação de estados globais consistentes. Tal mecanismo é necessário para garantir que apesar da desconexão, falha ou perda de um ou mais destes dispositivos, a computação executada pelo grupo possa ser restaurada a partir do último estado global salvo. No entanto, gravação e recuperação de estados globais consistentes, de aplicações distribuídas no ambiente de computação móvel, são dificultadas devido a um conjunto de restrições e características intrínsecas ao ambiente.

Os mecanismos atuais de gravação e recuperação de estados globais consistentes são inadequados para o ambiente de computação móvel. Mecanismos síncronos requerem que todas as unidades móveis, executando uma aplicação distribuída, cooperem na gravação de seus estados locais. Isso garante que um conjunto de estados locais seja consistente. Consequentemente, mensagens de controle necessitam ser enviadas para as unidades móveis a fim de que as mesmas gravem os seus estados locais. Entretanto, isso acarreta um custo de pesquisa para determinar a localização de uma unidade móvel na rede, isto é, a estação base onde a mensagem de controle deve ser entregue. Além disso, acrescenta-se o fato de uma unidade móvel poder mudar de célula antes que o algoritmo termine a sua execução. Se o algoritmo requer que a unidade móvel seja contactada n vezes durante

a sua execução, então o custo da pesquisa pode ser multiplicado por n . Outro problema relacionado ao esquema síncrono é como obter um corte consistente, no caso de uma unidade móvel estar desconectada. Mecanismos assíncronos são mais adequados para o ambiente de computação móvel, pois não requerem que mensagens de controle sejam enviadas para que as unidades gravem os estados locais. No entanto, na obtenção de um corte consistente é necessário haver troca de mensagens entre as unidades móveis, e os problemas com os esquemas síncronos descritos acima ocorrem aqui também.

Diante destas restrições, apresenta-se um novo protocolo para gravação e recuperação de estados globais consistentes, cujo tamanho das mensagens é minimizado e adequado ao ambiente de computação móvel. Levando-se em consideração as restrições de força, banda passante e latência nos sistemas móveis, tal redução é de vital importância para este ambiente. No protocolo proposto, no caso de ocorrência de falhas, a recuperação do estado global torna-se mais cara devido a dependências transitivas. No entanto, espera-se que o progresso da computação seja mais frequente do que a reconstruções de estados.

Este capítulo está estruturado da seguinte forma: na seção 5.2, apresenta-se um mecanismo para gravação e recuperação de estados globais consistentes e, a regra das duas fases. Na seção 5.3, descreve-se uma técnica pra minimizar o tamanho das mensagens e, o algoritmo que é proposto para gravação e recuperação de estados globais consistentes em ambiente de computação móvel. A seção 5.4 apresenta o resumo do capítulo.

5.2 Pontos de recuperação no ambiente móvel

No algoritmo proposto utiliza-se o esquema assíncrono para gravação de estados locais. Uma vez que os discos das unidades móveis não são considerados confiáveis, usa-se os discos das estações bases para armazenar os estados locais e os *logs* das mensagens das unidades móveis. No entanto, devido às mudanças de célula de uma unidade móvel, sucessivos estados locais de uma mesma unidade móvel podem estar armazenados em diferentes estações bases. Além disso, a desconexão de uma unidade móvel pode suspender a execução do algoritmo de recuperação de estados globais consistentes, até que esta unidade seja reconectada. Entretanto, a natureza voluntária de uma desconexão sugere que é possível uma unidade móvel gravar seu estado local e transferi-lo para sua estação base local como parte de um protocolo de desconexão. O algoritmo pode então usar este estado como um substituto do estado local mais recente da unidade móvel, na construção do estado global.

5.2.1 Estado global consistente

Para se modelar o estado global de uma aplicação distribuída, executando nos sistemas móveis, assume-se que existe um canal FIFO, C_{ij} , de uma unidade móvel i a uma unidade móvel j . O estado global G_Ckpt da aplicação distribuída executando nas N unidades móveis consiste de um estado local, $local_ckpt_i$, para cada unidade i , tal que:

- $local_ckpt_i$ inclui um estado local da unidade móvel i . O estado local de uma unidade móvel muda devido à recepção ou envio de uma mensagem, ou devido a um evento local que não envolve comunicação. Para uma unidade móvel i com um estado inicial s_{i0} , o estado após uma seqüência de eventos E_i é representado por (s_{i0}, E_i) ; se i está gravando seu estado local, então todos os eventos E_i são ditos incluídos no estado local e, portanto, no G_Ckpt .
- $local_ckpt_i$ também inclui uma cópia de toda mensagem m enviada, de i para qualquer outra unidade móvel j , se o evento $send(m)$ é incluído em G_Ckpt enquanto $recv(m)$ não o é.

Um estado global G_Ckpt é dito consistente se ele satisfaz a seguinte condição: para qualquer mensagem m , se $recv(m)$ é incluído em G_Ckpt , então $send(m)$ também o é.

A Regra das duas fases

Um algoritmo de gravação e recuperação de estados globais consistentes deve garantir que, apesar das unidades móveis gravarem seu estado local assincronamente, de acordo com algum protocolo local, é sempre possível a obtenção de um corte consistente utilizando o conjunto de estados disponíveis. Dado um estado local qualquer de uma unidade móvel, a partir do qual inicia-se o processo de obtenção de um estado global, o algoritmo deve ser capaz de selecionar um conjunto de estados locais, um para cada unidade móvel, de tal maneira que formem um estado global consistente. Para ilustrar o problema que surge quando cada unidade móvel grava seu estado local assincronamente, considere um sistema com três unidades móveis h_1, h_2, h_3 mostrado na figura 5.1. Sejam C_1, C_2, C'_3 os estados locais mais recentes de h_1, h_2, h_3 , respectivamente. Seja C_3 o estado anterior de h_3 armazenado em uma estação base diferente daquela de C'_3 . Assuma agora que o algoritmo começa a construção do estado global a partir de C_3 . A inclusão de C_3 no estado global implica que o evento $recv(M_2)$ seja incluído. Portanto, para manter um estado global consistente, o $send(M_2)$ deve ser incluído, ou seja, C_2 . Uma vez que $recv(M_1)$ está incluído em C_2 , e, portanto, no estado global, o evento $send(M_1)$ deve, também, ser incluído. O único estado em h_1 que inclui $send(M_1)$ é C_1 , sendo este selecionado como estado local representativo de h_1 . Entretanto, a inclusão de C_1 implica que $recv(M_3)$

pertence ao estado global, e, portanto, o evento $send(M_3)$ deverá ser incluído para mantê-lo consistente. Entretanto, isso só é possível se o estado C'_3 for incluído no estado global, representando o estado local de h_3 no lugar de C_3 . Assim sendo, o algoritmo não é capaz de formar um estado global consistente a partir de C_3 , ou seja, a partir de um estado local qualquer.

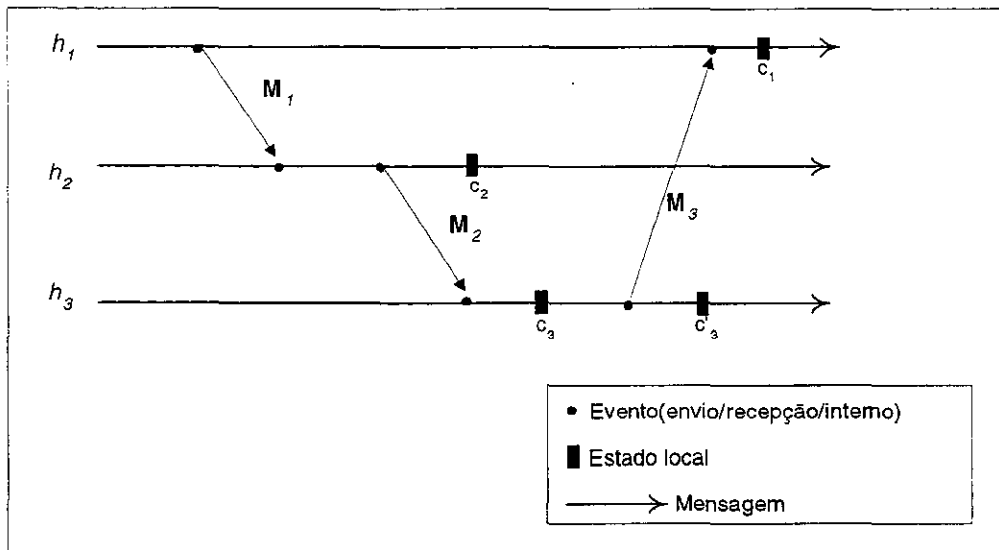


Figura 5.1: Gravação de estados locais assincronamente

A regra das duas fases proposta por [AB94] determina quando uma unidade móvel h_i deve gravar seu estado local, e garante que nenhuma dependência é criada entre dois estados locais de uma mesma unidade móvel, como no exemplo ilustrado acima. A regra é a seguinte:

- Se h_i está na $fase_i = SEND$ e recebe uma mensagem M , então ela deve gravar seu estado local e mudar para a fase $RECV$, ou seja, $fase_i := RECV$ antes de entregar a mensagem para o aplicativo.
- Se h_i está na $fase_i = RECV$, então, antes de enviar uma mensagem M para uma outra unidade móvel, h_i deve mudar para a fase $SEND$, isto é, $fase_i = SEND$.

5.3 O Algoritmo proposto

No algoritmo proposto, cada unidade móvel h_i mantém um relógio vetorial parcial, $h.vt_i$ e, um *timestamp* $e.vt_i$ representando a sua lista de dependência. A tupla (j, v) desse relógio e do *timestamp* significa que h_i é pseudo dependente de um estado local, cujo

número é v , da unidade móvel h_j . h_vt_i é inicializado com $\{(i, 1)\}$, e sempre que h_i gravar seu estado local, ela atualiza o seu relógio como definido pela técnica de Jard-Jourdan descrita abaixo, bem como seu *timestamp*. Assim sendo, $h_vt_i[i]$ sempre armazena o próximo estado de h_i a ser gravado. Associado a cada h_i existe também um vetor loc_i , que armazena as posições das unidades móveis h_j relacionados diretamente com h_i através da relação pseudo-direta descrita abaixo. Além disto, existe um vetor, L_vh_i , associado com a lista de dependência da unidade móvel e atualizado de forma similar a esta. $loc_i[i]$ armazena o *id* da estação base corrente a h_i . Este vetor muda sempre que h_i se mover para uma nova célula. A regra que rege esses vetores, h_vt_i e loc_i , é como se segue: se $h_vt_i[j]$ é p e $loc_i[j]$ é q , então um estado global que inclui $h_vt_i[i]$ como o representante do estado local de h_i deve incluir o p -ésimo estado local h_j , que está localizado na estação base cujo *id* é q . Além disso, cada unidade móvel grava seu estado local sempre que: (i) mover para uma nova célula, (ii) antes de desconectar-se e (iii) quando requerido pela regra das duas fases. O algoritmo consiste de duas partes: uma executada por toda unidade móvel para gravar seu estado local independentemente, baseado na regra das duas fases, e uma outra parte executada por uma estação base para obter um conjunto de estados locais consistentes, que é executado em caso de falhas. Toda vez que uma unidade móvel h_i envia uma mensagem m para uma outra h_j , ela adiciona à mensagem uma cópia dos vetores h_vt e loc . Por outro lado, quando uma estação base inicia o processo de obtenção de um estado global consistente, devido a uma falha em uma unidade móvel, a partir de um dado estado local de h_i , consistindo de $local_estado_i$, log_i , loc_i e h_vt_i , ela envia uma mensagem $request(h_j, h_vt_i[j])$ para todas as estações q , onde q é o valor de $loc_i[j]$, requisitando pelo estado local da unidade móvel h_j cujo número é $h_vt_i[j]$. Caso o relógio da unidade móvel h_i contenha apenas um componente, ou seja, $h_vt_i = (i, v)$ sendo v o *id* da estação base corrente, então a estação base utiliza a lista de dependência, e_vt_i , e o vetor de localização, L_vh_i , associado à lista, para enviar a mensagem $request$ para as unidades h_j relacionadas diretamente com h_i . Observe que, em ambos os casos, a mensagem $request$ é enviada apenas para as estações que contêm em suas células as unidades móveis relacionadas diretamente com h_i .

5.3.1 Reduzindo o tamanho das mensagens

O algoritmo proposto utiliza a técnica de manutenção de dependência proposta por [JJ94] para reduzir o número de dependências enviadas nas mensagens. Nesta técnica, desenvolvida para um sistema de depuração distribuída, os eventos podem ser observados enquanto é mantida a capacidade de recuperar todas as dependências casuais. Jard-Jourdan definem uma relação pseudo-direta, denotada por \ll , nos eventos de uma computação distribuída, da seguinte maneira: se os eventos e_i e e_j acontecem nos processos p_i e p_j ,

respectivamente, então $e_j \ll e_i$ (e_j está relacionado com e_i) se existe um caminho de transferências de mensagem que se inicia após e_j , em p_j , e antes de e_i , em p_i , tal que não existe nenhum evento observado no caminho.

O relógio vetorial parcial, p_vt_i , no processo p_i é uma tupla da forma (j, v) , indicando que o corrente estado de p_i é pseudo dependente de um evento, cujo número é v , no processo p_j . Inicialmente o relógio num processo p_i é: $p_vt_i = \{(i, 0)\}$. Além disso, temos que:

1. Sejam $p_vt_i = \{(i_1, v_1), \dots, (i, v), \dots\}$ e e_vt_i , respectivamente, o corrente relógio vetorial parcial de p_i , e o *timestamp* do evento observado. Quando um evento é observado num processo p_i , as seguintes ações são executadas:
 - $e_vt_i = \{(i_1, v_1), \dots, (i, v), \dots\}$.
 - $p_vt_i = \{(i, v + 1)\}$.
2. Quando um processo p_j envia uma mensagem para p_i , ele adiciona o valor corrente de p_vt_j na mensagem.
3. Quando p_i recebe uma mensagem com um valor de relógio p_vt , p_i procede da seguinte maneira: Seja $p_vt = \{(i_{m1}, v_{m1}), \dots, (i_{mk}, v_{mk})\}$ e $p_vt_i = \{(i_1, v_1), \dots, (i_l, v_l)\}$. p_i atualiza o p_vt_i , onde p_vt_i é a união de:
 - todos (i_{mx}, v_{mx}) tal que $(i_{mx}, .)$ não aparece em p_vt_i ,
 - todos (i_x, v_x) tal que $(i_x, .)$ não aparece em p_vt , e
 - todos $(i_x, \max(v_{m1}, v_{xm}))$ para todo $(v_x, .)$ que aparece em p_vt e em p_vt_i .

A figura 5.2 ilustra a técnica descrita acima. Nesta, eN_pt_n representa o *timestamp* do N -ésimo evento observado no processo p_n . Por exemplo, o terceiro evento observado no processo p_i é $e3_pt_3 = \{(3,2), (4,1)\}$. Este *timestamp* significa que o estado anterior do p_3 é pseudo dependente dos eventos que estão localizados nos processos p_3 e p_4 . Observe que o estado corrente do processo p_3 ($v_pt_3 = \{(3,3)\}$) é dependente de um evento 3 em p_3 .

5.3.2 Estabelecimento de pontos de recuperação

1. Gravar o estado local, $estado_local_i$, da aplicação distribuída executando no h_i ;
2. Transferir o estado local $estado_local_i$, log_i , h_vt_i , para a estação base local;
3. Sejam $h_vt_i = \{(i_1, v_1), \dots, (i, v), \dots\}$, denotando o corrente relógio vetorial parcial de h_i , a variável e_vh_i , denotando o *timestamp* do evento observado, e $loc_i = \{(i_1, estação_1), \dots, (i, estação_i), \dots\}$, denotando o vetor de posições correntes da unidade móvel. Quando um evento é observado em h_i , as seguintes ações são executadas:

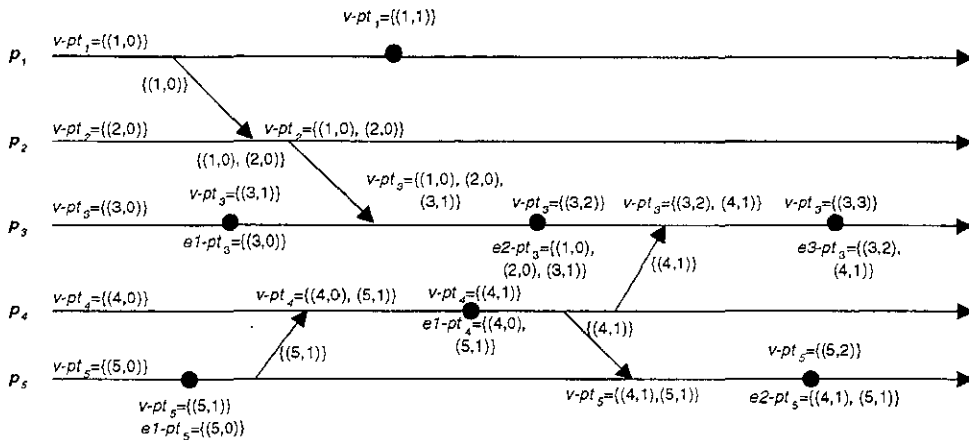


Figura 5.2: Exemplo de uma computação com a técnica de Jard-Jourdan

- $e_vh_i = \{(i_1, v_1), \dots, (i, v), \dots\}$,
- $h_vt_i = \{(i, v + 1)\}$,
- $L_vh_i = \{(i_1, \text{estação}_1), \dots, (i, \text{estação}_i), \dots\}$
- $loc_i = \{(i, \text{estação})\}$.

5.3.3 Protocolo de gravação de estado

No envio de uma mensagem m

1. Se $fase_i$ é *RECV*, então $fase_i$ é mudada para *SEND*;
2. Adiciona uma cópia de h_vt_i e do loc_i à mensagem M ;
3. Adiciona M ao log das mensagens, log_i .

No recebimento de uma mensagem m de uma unidade móvel h_j

1. Se $fase_i = SEND$, então execute o procedimento de estabelecimento de pontos de recuperação
2. $fase_i := RECV$
3. Seja $h_vt = \{(i_{m1}, v_{m1}), \dots, (i_{mk}, v_{mk})\}$ e $h_vt_j = \{(j_1, v_1), \dots, (j_l, v_l)\}$. Atualize o h_vt_j , onde h_vt_j é a união de:
 - todos (i_{mx}, v_{mx}) , tal que (i_{mx}, \cdot) não aparece em h_vt_j ,
 - todos (j_x, v_x) , tal que (j_x, \cdot) não aparece em h_vt , e

- todos $(j_x, \max(v_{m1}, v_{xm}))$, para todo $(v_x, .)$ que aparece em h_vt e em h_vt_i .
4. Seja $L_vt = \{(i_{m1}, \text{estação}_{m1}), \dots, (i_{mk}, \text{estação}_{mk})\}$ e $loc_j = \{(j_1, \text{estação}_1), (j_l, \text{estação}_l)\}$.
Atualize o loc_j , onde o loc_j é a união de:
- todos $(i_{mx}, \text{estação}_{mx})$, tal que $(i_{mx}, .)$ não aparece em loc_j ,
 - todos $(j_x, \text{estação}_x)$, tal que $(j_x, .)$ não aparece em L_vt , e
 - todos $(j_x, \text{estação}_{m1})$, para todo $(v_x, .)$ que aparece em loc e em loc_j

5.3.4 Protocolo de reconstrução de estados globais

Dado um estado local da unidade móvel, h_i , consistindo de estado_local_i , log_i , e h_vt_i , a partir do qual se inicia o processo de obtenção de um estado global, uma estação base executa os seguintes passos para obter um conjunto consistente de estados locais:

Iniciador

1. Se $h_vt_i = \{(i, v)\}$, através da lista de dependência da unidade móvel, ou seja, e_vh_i e do vetor de localização L_vh_i , associado a esta lista, faça:

Para $1 \leq j \leq N$, requisiute o estado local de h_j , cujo número é p , para a estação _{q} , através do envio de uma mensagem $\text{request}(h_j, e_vt_i[j])$, sendo N o número de unidades móveis h_j relacionadas diretamente com h_i (através da lista de dependência), p é o valor de $e_vt_i[j]$, e q é o valor de $L_vh_i[j]$.
2. Senão, para $1 \leq j \leq M$, requisiute o estado local de h_j , cujo número é p , para a estação _{q} , através do envio de uma mensagem $\text{request}(h_j, h_vt_i[j])$, onde M é o número de unidades móveis h_j relacionadas diretamente com h_i , onde p é o valor de $h_vt_i[j]$, e q é o valor de $loc_i[j]$.
3. O iniciador do processo espera até que receba um estado local de cada h_j e uma lista de dependência transitiva de cada uma delas. Se houver alguma unidade móvel, h_k , em alguma das listas recebidas para a qual ainda não foi enviada uma mensagem request , o iniciador repete o passo anterior. O iniciador espera até receber um estado local de todas as estações bases para o qual ele enviou uma mensagem request . Se alguma estação base responder que o estado local desejado não está disponível devido a uma falha da unidade móvel h_j , então o algoritmo de reconstrução de estados globais aborta, visto que não é possível conseguir um corte consistente com um estado local de cada participante. Se todas as estações responderem com o estado local desejado, o conjunto destes estados locais formam um estado global consistente.

Respondendo a uma mensagem do tipo $request(h_j, h_vt_i[j])$

1. Se o estado local da unidade móvel h_j com número p está disponível na estação base que recebeu a mensagem $request()$, então ela responde imediatamente ao iniciador do processo, enviando-lhe o estado requerido. Caso contrário, se h_j ainda não gravou o próximo estado local que se iguale a p , a estação espera até que h_j grave o próximo estado local cujo número seja igual a p . A unidade móvel irá gravar e transferir o estado local requerido antes da sua próxima desconexão, ou antes de entregar, ou receber, qualquer mensagem. Entretanto, nenhum destes dois eventos são garantidos ocorrerem. Portanto, a estação base pode forçar a unidade móvel a gravar seu estado local. Além disso, é possível que h_j falhe antes de gravar o seu estado local. Neste caso, a estação informa ao inicializador a falha de h_j .

A figura 5.3 ilustra o comportamento do algoritmo proposto. Nesta, sejam $h_vt_1, h_vt_2, h_vt_3, h_vt_4, h_vt_5$ os relógios das unidades h_1, h_2, h_3, h_4 e h_5 , respectivamente. Assuma ainda que todas as unidades estão na fase de *SEND*. Assim sendo, quando a unidade h_2 envia uma mensagem para a unidade h_1 , esta deve, pela regra das duas fases, gravar seu estado local antes de entregar a mensagem ao aplicativo. Além disto, a unidade h_1 atualiza o seu relógio de (1,1) para (1,2), (2,1), e o seu $timestamp(e_1-hv_1)$ é inicializado com (1,1). Da mesma forma, quando h_4 envia uma mensagem para h_5 e h_3 , e h_5 envia uma mensagem para h_4 , as unidades h_3, h_4 e h_5 também devem gravar os seus estados locais antes de entregar as respectivas mensagens, atualizar os seus relógios e inicializar seus respectivos $timestamps$. Entretanto, quando as unidades h_4 e h_5 recebem uma mensagem das unidades h_3 e h_2 , respectivamente, elas entregam as respectivas mensagens direto ao aplicativo, visto que se encontram na fase de *RECV*.

No melhor caso, apenas um componente do relógio h_vt é enviado, no caso médio, dois e, no pior caso, o relógio terá o mesmo tamanho que os vetores do algoritmo proposto por [AB94].

5.4 Resumo do capítulo

Neste capítulo apresentou-se um novo protocolo de ponto de recuperação para o ambiente de computação móvel. Aplicando a técnica proposta por [JJ94], reduziu-se o número de informações que são enviadas em uma mensagem em relação a outras abordagens. No protocolo proposto, uma unidade móvel h_i guarda somente as informações referentes às estações das quais ela é pseudo-dependente num determinado estado corrente. Estas informações são armazenadas em um relógio vetorial parcial, h_vt , e quando uma mensagem é enviada, h_i adiciona à mensagem o valor corrente do relógio vetorial parcial h_vt . Com isto, temos no melhor caso, o envio de somente um componente do relógio vetorial parcial,

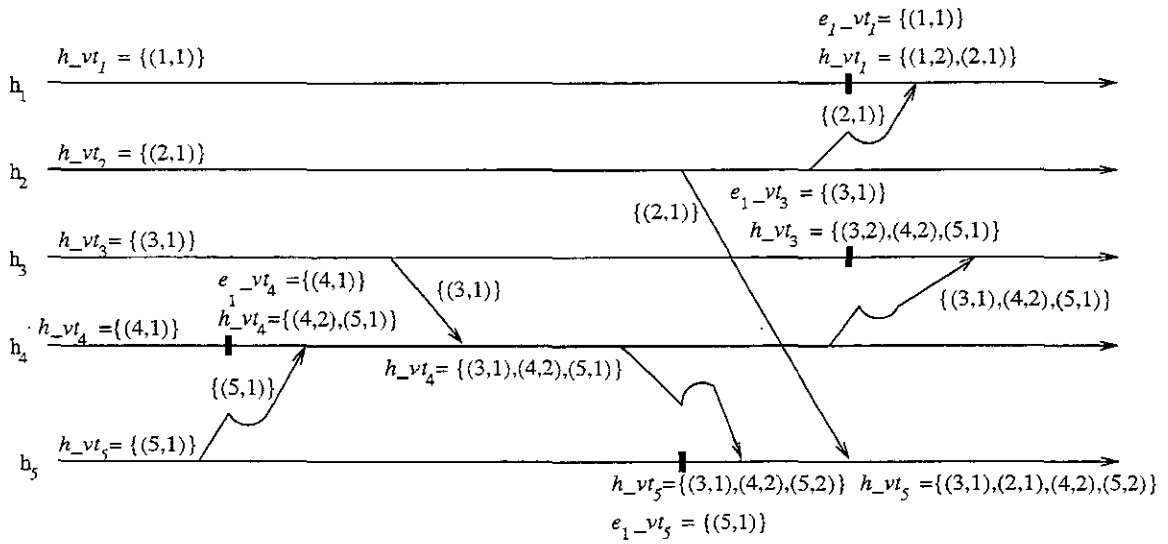


Figura 5.3: Exemplo de uma computação com o protocolo proposto

no caso médio, dois componentes, e no pior caso, o mesmo tamanho dos vetores CKPT e LOC do algoritmo desenvolvido por [AB94]. Levando-se em consideração as restrições do ambiente móvel, esta redução é de vital importância. Deve-se notar que no algoritmo proposto para a obtenção do estado global, a reconstrução de estado pode tornar-se mais cara devido às dependências transitivas. Todavia, espera-se que o progresso da computação seja mais frequente do que restaurações de estados.

Capítulo 6

Conclusões e trabalhos futuros

*“O temor do Senhor é o princípio da sabedoria.
Os loucos desprezam a sabedoria e a instrução”
(Pv 9.10;1.7)*

Nesta dissertação analisou-se aspectos de um problema específico a consistência de dados distribuídos, no contexto de computação móvel, como validação de transação e pontos de recuperação.

Durante o desenvolvimento deste projeto, focalizou-se no problema de consistência de dados no ambiente móvel. Observou-se que, quando a desconexão é tratada como uma falha, qualquer solução que se apresente será uma mera adaptação ou variação das existentes nos sistemas distribuídos convencionais. As soluções tornam-se diferentes quando tal problema passa a ser tratado como uma falha preparada e planejada. Isto implica que tanto o sistema quanto a unidade móvel devem ser capazes de tomar ações especiais antes da desconexão. Um ponto central a ser considerado pelos protocolos para o ambiente de computação móvel é fornecer autonomia às unidades móveis de uma maneira tal que, os usuários possam executar suas aplicações durante o período de desconexão. Portanto, pode-se levantar algumas questões importantes. Uma vez que o usuário pretende se desconectar, quais ações podem ser tomadas por ele e pelo sistema, antes da desconexão, a fim de que o mesmo possa continuar o processamento de suas aplicações e o sistema possa operar normalmente? Qual o nível de consistência que pode ser garantida? Quais as vantagens e desvantagens para o usuário?

Um boa solução para estas questões é a apresentada pelo modelo de execução de transação proposto nesta dissertação. Neste, as unidades móveis transferem algumas informações para a parte estática da rede antes da desconexão. Além disso, a parte estática da rede pode tomar ações especiais com relação às transações, no lugar da unidade móvel, não somente no período de desconexão mas também durante a conexão da unidade

móvel. Com isto, o modelo proporciona autonomia às unidades móveis, alto grau de concorrência, faz pouco uso da estreita banda passante, lida como a restrição de latência, obtendo ganhos de energia e reduzindo o custo financeiro envolvido, tornando a solução apresentada adequada ao ambiente de computação móvel.

Uma outra boa solução igualmente adequada a este ambiente é o protocolo de gravação e recuperação de estados globais consistentes apresentado. Na abordagem proposta, o tamanho das mensagens é minimizado, sendo um fator importante para o ambiente móvel, levando-se em consideração as restrições deste ambiente.

Assim sendo, as principais contribuições deste trabalho são: o protocolo de gravação e recuperação de estados globais consistentes e o modelo de execução de transação propostos, os quais formam um arcabouço de consistência de dados para o ambiente móvel.

A principal conclusão reside no fato de a desconexão das unidades móveis ser o ponto central que torna as soluções no ambiente móvel diferentes daquelas dos sistemas convencionais.

Como trabalho futuro sugere-se uma verificação tanto do modelo de execuções de transação como do protocolo de gravação e recuperação de estados globais, através de protótipos e experimentos a fim de que ambos possam ser validados.

Bibliografia

- [AAFZ95] Swarup Acharya, Rafael Alonso, M. Franklin, and S. Zdonik. Broadcast disk: Data management for asymmetric communications environments. *Proc. Acm SIGMOD Conf., San Jose, CA*, may 1995.
- [AB93] Arup Acharya and B.R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proc. of the third Intl. Conf. On Parallel and Distributed Information Systems*, May 1993.
- [AB94] Arup Acharya and B.R. Badrinath. Checkpointing distributed applications on mobile computers. In *Proc of the third Intl. Conf. On Parallel and Distributed Information Systems*, Septembr 1994.
- [AFZ96] Swarup Acharya, M. Franklin, and S. Zdonik. Prefetching from broadcast disk. *12th International Conference on Data Engineering, New Orleans, LA*, feb 1996.
- [AK93] R. Alonso and Henry F. Korth. Database system issues in nomadic computing. In *proceeding of the ACM SIGMOD, Conference on Management of Data*, pages 388–392, 1993.
- [BBI92] Badrinath, B.R., and T. Imielinski. Replication and mobility. In *Proc. of the 2nd workshop on the management of replicated data*, pages 9–12, 1992.
- [BBI94] Arup Acharya B.R. Badrinath and T. Imielinski. Designing distributed algorithms for mobile computing networks. In *Proc. of the fourth Intl. Conf. On Distributed Computing Systems*, May 1994.
- [BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley Publishing Ltd., 1987.

- [BK91] Naser S. Barghouti and Gail E. Kaiser. Concurrency control in advanced database applications. *ACM Computing Surveys*, 23(3):269–317, September 1991.
- [CDK96] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, second edition, 1996.
- [CdOA98] D rí o Vieira Concei o and Ricardo de Oliveira Anido. Detec o de estado global em sistema de computa o m vel. *160 Simp sio de Redes de Computadores*, pages 195–205, May 1998.
- [Chr93] Panos K. Chrysanthis. Transaction processing in mobile computing environment. *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–82, Oct 1993.
- [CL85] K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed system. *ACM transactions on Computer System*, 1(3):63–75, February 1985.
- [DM91] J. Ioannidis;D. Duchamp and G.Q. Maguire. Ip-based protocols for mobile networking. In *Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pages 235–245, September 1991.
- [Duc92] Dan Duchamp. Issues in wireless mobile computing. In *Proc. Third Wkshp. on Workstation Operating Systems, IEEE, Key Biscayne FL*, pages 1–7, April 1992.
- [EJB95] Ahmed Elmagarmid, Jin Jing, and Omran Bukhres. An efficient and reliable reservation algorithm for mobile transactions. *To appear in Proceedings of the 4th International Conference on Information and Knowledge Management*, 1995.
- [EJW96] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang. A survey of rollback-recovery protocols in message-passing systems. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, October 1996.
- [EN94] Ramez Elmasri and Shamkant B. Navathe. *Fudamentals of Database Systems*. Buschlen/Mowatt Fine Arts Ltd., 2 edition, 1994.
- [FZ94] George H. Forman and John Zahorjan. The challenges of mobile computing. In *Available as UW CSE Tech Report 93-11-03 from ftp.cs.washington.edu*, March 1994.

- [HKT93] Lars Krombholz Henning Koch and Oliver Theel. A brief introduction into world of mobile computing. In *Department of Computer Science, University of Darmstad*, Alexanderstr 10, D-W-6100 Darmstadt, Germany, May 1993.
- [HL95] P. Honeyman and L.B.Huston. Communications and consistency in mobile file systems. In *CITI Technical Report in IEEE Personal Communications, Special Issue on Mobile Computing*, December 1995.
- [IA96] Cristian Ionitoiu and Birger Andersen. Replication objects with lazy consistency. *ECOOP'96 II Workshop on Mobility and Replication*, 1996.
- [IB93] Tomasz Imielinski and B.R. Bandrinath. Data management for mobile computing. In *SIGMOD RECORD*, volume 22, No 1, pages 34–39, March 1993.
- [IB94] T. Imielinski and B.R. Badrinath. Wireless mobile computing: challenges in data management. *Communications of ACM.*, 37(10), 1994.
- [JBE95] Jin Jing, Omran Bukhres, and Ahmed K. Elmargarmid. Distributed lock management for mobile transactions. *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS'95), Vancouver, BC, Canada*, May 1995.
- [JJ94] C. Jard and G-C. Jourdan. Dependency tracking and filtering in distributed computations. In *A full presentation appeared as IRISA Tech Report No. 851*, 1994.
- [JP73] R.A. Jarvis and Edward A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE Transactions on Computers*, 22(11):1025–1034, November 1973.
- [KRP97] Geoffrey H. Kuenning, Peter Reiher, and Gerald J. Popek. Experience with automated hoarding system. 1997.
- [KS91a] J. J. Kistler and M Satyanarayanan. Disconnected operation in the coda filesystem. *Thirteenth ACM Symp. on Operating System Principles*, pages 213–225, october 1991.
- [KS91b] J.J. Kistler and M. Satynarayanan. Disconnected operation in the coda file system. In *Thirteenth ACM Symp. on Operating System Principles*, pages 213–225, October 1991.

- [Kue97a] Geoffrey H. Kuenning. *SEER: Predictive File Hoarding for Disconnected Mobile Operation*. Ucla csd technical report ucla-csd-970015, University of California, Los Angeles, Los Angeles, CA, may 1997.
- [Kue97b] Geoffrey Houston Kuenning. *Seer: predictive file hoarding for disconnected mobile operation*. *University of California, Los Angeles*, 1997.
- [LS94] Qi Lu and M. Satyanarayanan. Isolation-only transactions for mobile computing. *Operation System Review*, 28(2):81–87, April 1994.
- [ML98] Geraldo Robson Mateus and Antonio Alfredo Ferreira Loureiro. *Introdução à Computação Móvel*. 11a escola de computação, 1998.
- [Mos82] J. Eliot B. Moss. Nested transactions and reliable distributed computing. *2nd. Symposium on Reliability in Distributed Software and database Systems*, 1982.
- [Nar93] Vivek R. Narasayya. Distributed transactions in a mobile computing system. In *Submitted as part of requirement for CSE552*, October 1993.
- [Nar94] Vivek R. Narasayya. Distributed transactions in a mobile computing system. *First IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA*, December 1994.
- [NF96] Nuno Neves and W. Kent Fuchs. Using time to improve the performance of coordinated checkpointing. in *proceedings of the IEEE International Computer Performance and Dependability Symposium*, September 1996.
- [NF97] Nuno Neves and W. Kent Fuchs. Adaptive recovery for mobile environments. In *Communications Of The ACM*, volume 40, pages 68–74, January 1997.
- [Nie95] Jeppe Damkjaer Nielsen. Transactions in mobile computing. *Universidade do Minho, Portugal*, 1995.
- [PB94] Evangelia Pitoura and Bharat Bhargava. Building information systems for mobile environments. *Proceedings of the 3th International Conference on Information and Knowledge Management*, November 1994.
- [PB95] Evangelia Pitoura and Bharat Bhargava. Maintaining consistency of data in mobile distributed environments. *Proceeding of the 15th International Conference on Distributed Computing Systems*, May 1995.

- [PK88] Calton PU and Gail E. Kaiser. Split transactions for open-ended activities. *14th VLDB conference*, 1988.
- [PKV96] D.K. Pdrahan, P.P. Krishna, and N.H. Vaidya. Recovery in mobile wireless environment: Design and trade-off analysis. In *Proceeding of 26th international Symposium on Fault-Tolerance Computing*, pages 16–25, June 1996.
- [PM96] R. Prakash and M.Sihghal. Low-cost checkpointing and failure recovery in mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(7), October 1996.
- [RRPK97a] David Ratner, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kenning. Replication requirements in mobile environments. *1st international workshop on discrete algorithms and methods for mobile computing and communications*, oct 1997.
- [RRPK97b] David Ratner, Peter Reiher, Gerald J. Popek, and Geoffrey H. Keunning. Replication requirements in mobile environments. *1st international workshop on discrete algorithms and methods for mobile computing and communications, Budapest Hungary, Applications*, oct 1997.
- [Rus80] David L. Russell. State restaration in systems of communicating processes. *IEEE Transactions on Software Engineering*, SE-6(2):183–194, March 1980.
- [SKM+93] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with disconnected operation in mobile computing environment. June 1993.
- [Sor96] Michael G. Sorensen. A model for multi-level consistency. *OOPSLA '96 Workshop on Object Replication and Mobile Computing(ORMC'96)*, December 1996.
- [TD91] Carl D. Tait and Dan Duchamp. Detection and exploitation of file working sets. In *Proc. Eleventh Intl. Conf. on Distributed Computing Systems, IEEE*, pages 2–9, may 1991.
- [TLAC95] Carl D. Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. In *Proceedings of MobiCom'95: The First International Conference on Mobile Computing and Networking*, pages 119–125, nov 1995.
- [Vos91] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley Publishers Ltd., 1991.

- [WC97] Gary D. Walborn and Panos K. Chrysanthis. Pro-motion: Management of mobile transactions. *Proceedings of the 11th ACM Annual Symposium on Applied Computing, Special Track on Database Technology*, March 1997.