Uma Visão Geral Sobre Algoritmos de Recuperação Em Banco de Dados

Luís Augusto Angelotti Meira*

Campinas, 26 de julho de 2004

Resumo

Este artigo trata de recuperação em banco de dados. Apresentamos uma pesquisa histórica sobre o assunto e uma análise de algumas técnicas utilizadas. Damos especial atenção a técnicas primitivas como forçar a escrita das alterações em mídia estável, baseada ou não em espelhamento (*S*hadow). Discutiremos em maior detalhes o algoritmo de recuperação no caso de falhas do Sistema R, que utiliza *log* e espelhamento, mas não força a escrita das alterações antes da validação. Por fim fazemos uma comparação entre este algoritmo e o algoritmo ARIES, que é um dos mais utilizados atualmente para este fim.

*Este trabalho foi realizado para complementar o processo de avaliação no curso *MO410/A - 2004.1 Banco de Dados* sob orientação do professor doutor Geovane Cayres Magalhães

Sumário

1 Introdução

O principal algoritmo para recuperação no caso de falhas coberto no curso *MO410/A - 2004.1 Banco de Dados* foi ARIES (*Algorithm for Recovery and Isolation Exploiting Semantics*), proposto primeiratente por C. Mohan *et al.* [10]. Existem uma série de trabalhos como [13, 3, 15, 11] sobre o assunto. Em especial, [3] é um artigo conciso e didático sobre ARIES. A necessidade de um bom documento em português sobre o tema pode ser parcialmente preenchida pelas transparências mostradas em aula [12].

Neste trabalho damos maior ênfase aos primeiros algoritmos para recuperação no caso de falhas, que são baseados em espelhamento. Abordamos em detalhes o Sistema R, precursor da linguagem SQL, e seu método de recuperação no caso de falhas.

O trabalho está organizado da seguinte forma: Seção 2 contém as características de um sistema tolerante a falhas; Seção 3 descreve dois protocolos simples baseados em forçar a escrita das alterações em mídia estável; Seção 4 traz uma descrição do Sistema R e seu algoritmo de recuperação no caso de falhas; Seção 5 traz uma comparação entre ARIES e o algoritmo de recuperação de falhas do Sistema R; e Seção 6 traz algumas conclusões.

2 Característica de um Sistema Tolerante a Falhas

Mesmo em um contexto onde pode haver falhas, uma transação deve conter as propriedades *ACID (atomicidade, consistência, isolamento e durabilidade).* Vamos supor que uma transação bancária transfere a quantia de R$ 50,00 de uma conta A para uma conta B. Esta transação precisa de quatro acessos ao banco de dados: uma leitura e uma escrita na conta A e uma leitura e uma escrita na conta B, como pode ser visto na Tabela 1.

- TRANSFERENCIA()

    1. saldo_A = **READ**(A)
    2. saldo_A = saldo_A - 50,00
    3. **WRITE**(A,saldo_A)
    4. saldo_B = **READ**(B)
    5. saldo_B = saldo_B + 50,00
    6. **WRITE**(B,saldo_B)

Tabela 1: Transação de transferência .

Um fator semântico importante para uma transação bancária de transferência é que a soma dos saldos envolvidos não pode aumentar nem diminuir. Dinheiro não pode ser criado nem destruído neste caso. Em [5] temos uma definição de consistência para banco de dados: um banco de dados é *consistente* se *e somente se*

ele contém o resultado das transações que sofreram validação (transações vencedoras). Se o sistema completar TRANSFERENCIA() somente até o passo 3 (ver Tabela 1), ela deixará um resultado (terá sacado 50,00 da conta A) e não será uma operação validada, logo o sistema será inconsistente (semanticamente houve destruição de dinheiro). Este é um exemplo onde atomicidade mostra-se fundamental. *Atomicidade* é a característica de uma transação ocorrer completamente, ou não deixar nenhuma alteração nos dados. Outro fator essencial em banco de dados é a durabilidade. *Durabilidade* consiste no seguinte: uma vez que a transação termina e é validada, os seus efeitos devem persistir no banco de dados mesmo que ocorra uma falha. A última característica ACID a ser a analisada é isolamento. Considere a transação de se obter a soma dos saldos das contas A e B descritos na Tabela 2.

- SOMA_SALDOS()

    1. saldo_A = **READ**(A)
    2. saldo_B = **READ**(B)
    3. print(saldo_A+saldo_B)

Tabela 2: Transação que soma os saldos de A e B .

Suponha que antes de ocorrer a operação TRANSFERENCIA(), saldo_A = 100,00 e saldo_B = 100,00. Se SOMA_SALDOS() for executado antes ou depois de TRANSFERENCIA(), ela deve retornar 200,00. Entretanto, se ambas as transações forem executadas concorrentemente, e TRANSFERENCIA() executar até a linha 3, onde o dinheiro é retirado da conta A, em seguida SOMA_SALDOS() for executada completamente, ela vai imprimir 150,00, mesmo que TRANSFERENCIA() termine. O banco de dados pode executar transações de forma concorrente mas, para haver *isolamento* deve existir pelo menos uma execução seqüencial equivalente. No exemplo acima, não existe nenhuma execução seqüencial onde SOMA_SALDOS() devolve 150,00, logo não houve isolamento*.

*A obtencão de isolamento em um banco de dados onde existe concorrência é feita através do

Um sistema tolerante a falhas deve, no momento da recuperação no caso de uma falha, fazer com que as transações que sofreram validação tenham seus resultados preservados no banco de dados (transações vencedoras) e as que não sofreram validação, no resultado final, pareçam nunca ter existido (transações perdedoras), preservando assim as características ACID.

Os três possíveis modelos de falha são falha na transação, falha do sistema e falha do disco.

Uma falha na transação ocorre quando a transação pede um *abort* e precisa ser desfeita. São mais comuns que as outras e podem ser originadas por um pedido de cancelamento explícito do usuário ou por uma ação unilateral do SGBD no caso de um *dead lock*, violação de autorização, limitações de recursos ou desligamento do sistema, dentre outras causas.

Falhas do sistema ocorrem mais raramente e têm várias causas como falta de luz, falha de hardware, falha no sistema operacional, dados corrompidos. Quando este tipo de falha ocorre, toda memória volátil pode ficar comprometida e o SGBD deve ser capaz de recuperar o último estado de transações validadas contando apenas com o que foi salvo em mídia estável (disco).

O terceiro tipo de falha ocorre quando o disco que suporta o banco de dados falha. É, em geral, mais rara que as anteriores e tem várias causas como falha mecânica ou um programa que desavisadamente corrompe o disco, dentre outras. Para este tipo de falhas, a recuperação consiste em copiar o último *back-up* e se houver um *log*, dependendo do algoritmo, reconstruir o último estado de transações validadas.

3 Protocolos Simples de Recuperação no Caso de Falhas

O protocolo mais simples para se construir um sistema tolerante a falhas que garanta atomicidade, consistência e durabilidade consiste em executar os passos

bloqueio de variáveis.

da transação em memória e, antes da transação sofrer validação, *forçar* a escrita das páginas que sofreram alteração em mídia estável (disco). Se o sistema cair antes da escrita em disco, que deve ser atômica, tal transação não deixa nenhum resultado no BD. Se cair após a escrita em disco, tal transação foi salva e tem durabilidade.

Protocolos para recuperação no caso de falhas baseado em forçar, entretanto, são muito custosos, pois, a cada transação validada, é necessário escrever uma ou mais páginas no disco. Outros protocolos como ARIES e o protocolo do Sistema R, também garantem atomicidade, consistência e durabilidade e, na prática, são muito mais eficientes por não precisar salvar em mídia estável todas as páginas referentes a dados alterados.

3.1 Algoritmo de Recuperação Baseado em Cópia *(Shadow)*

Apresentamos nesta seção uma técnica de recuperação baseada em cópia ou espelhamento *(shadow-pages-based)* que não usa *log* para a recuperação e tem a característica de forçar a escrita das páginas referentes a dados alterados a cada validação. A idéia é a seguinte: quando uma página de dados é modificada por uma transação, uma nova página é alocada para o dado modificado e o dado antigo perdura como uma cópia *espelhada*. Para fazer isto, basta manter duas tabelas de endereçamentos de página, uma tabela que é a *atual* e outra que é a *espelhada*. No início de uma transação as duas tabelas são iguais, mas a medida que as transações vão fazendo modificações no banco de dados, novas páginas são alocadas e a tabela de páginas atual é modificada para corresponder às novas páginas, enquanto a tabela espelho mantêm os apontadores para as páginas antigas. Veja Figura 1.

As páginas atuais podem estar em memória ou em disco, mas antes da transação ser validada, as páginas têm de ser gravadas em mídia estável. Se uma falha acontece antes de uma transação ser validada, a tabela de páginas *espelhadas* é utilizada para recuperar o estado do banco de dados antes da transação ser iniciada.

Quando uma transação sofre validação, as páginas na tabela atual, que neste

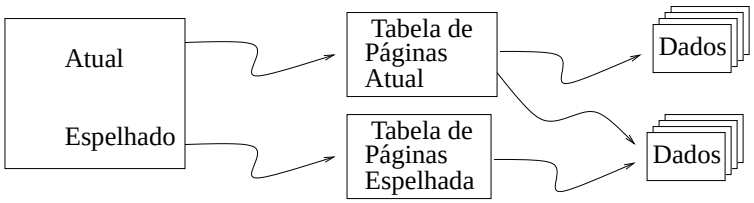

Figura 1: Sistema de espelhamento dos dados.

momento devem estar no disco, transformam-se em páginas na tabela de páginas espelhada. As páginas na tabela de páginas espelhada devem ser cuidadosamente trocadas por páginas novas em uma operação atômica. Para conseguir isto, a tabela de páginas atual é sincronizada após todas as páginas terem sido gravadas. Em seguida, o endereço das páginas espelhadas é substituído pelo endereço atual, validando a transação. Ocorrendo uma falha antes da validação, a tabela de endereços das páginas espelhadas é utilizada para recuperar o banco de dados no instante anterior à transação.

Este algoritmo é melhor que o descrito no início da Seção 3 pois naquele a escrita de todas as páginas referentes a dados alterados por uma determinada transação a ser validada tinha de ser atômica, o que é uma característica díficil de ser garantida. No algoritmo baseado em espelhamento visto nesta subseção, apenas o ato de copiar a tabela de páginas atual sobre a tabela de páginas espelhada é que precisa ter tal característica.

Este algoritmo de recuperação baseado em espelhamento elimina a necessidade de *log*, apesar de ser algumas vezes criticado no que se refere ao desempenho quando não há falhas. Porém, o tempo gasto para a recuperação quando uma falha acontece é geralmente menor utilizando esta técnica, se comparado a ARIES por exemplo. Além disto, trata-se de um algoritmo simples de ser implementado, apesar de necessitar técnicas avançadas para remoção de lixo *(garbage collection)*, que deve reconhecer todas as páginas espelhadas que não são mais necessárias.

Outro ponto fraco do sistema baseado em espelhamento apresentado acima é a dificuldade de tratar transações concorrentes. A granularidade da memória é

a página, geralmente de tamanho em torno de 4KB, e se mais de um processo concorrente modificar a mesma página e um validar, as modificações do segundo processo vão persistir, mesmo que ele não seja validado, o que pode gerar quebra de consistência e atomicidade.

4 O Algoritmo de Recuperação no Caso de Falhas do Sistema R

Sistema R é um sistema de banco de dados construído a partir de um projeto de pesquisa na IBM San Jose Research (agora Almaden Research Center) nos anos 70. O Sistema R introduziu a linguagem SQL e também demonstrou que um sistema relacional poderia ter boa performance no processamento de transações.

Tal sistema era composto de dois níveis além do sistema operacional:

- Sistema de Pesquisa de Dados
  - Suporta o modelo relacional,
  - suporta a linguagem relacional SQL,
  - provê suporte a sistema de nomes e autorização e
  - transforma declarações SQL em chamadas ao sistema de armazenamento de dados.
- Sistema de Armazenamento de Dados
  - Provê acesso aos dados,
  - mapeia registros em arquivos do sistema operacional e
  - provê o conceito de transação (recuperação e bloqueio).
- Sistema Operacional
  - Provê sistema de arquivos para gerenciar discos,
  - provê sistema de E/S e

– provê gerenciamento de processos (multi-processos).

Para ilustrar como era uma chamada SQL no início dos anos 80 a ser executada dentro do Sistema R, veja o código abaixo de um programa simples PL/I-SQL que transfere dinheiro entre duas contas.

```
1. TRANSFERENCIA
2.   $BEGIN_TRANSACTION
3.   ON ERROR DO
4.      $RESTORE_TRANSACTION
5.   $UPDATE CONTAS
6.           SET SALDO = SALDO - 100,00
7.      WHERE CONTAS.NUMERO = A;
8.   $UPDATE CONTAS
9.           SET SALDO = SALDO + 100,00
10.     WHERE CONTAS.NUMERO = B;
11.  $COMMIT_TRANSACTION
```

SQL foi primeiramente publicado pela ANSI 1986 e retificado pela OSI em 1987 e a versão mais atual é de 2003, SQL:2003. O código acima, extraído de [4], trata de uma versão bem mais antiga, de 1981, chamada PL/I-SQL, suportada pelo Sistema R.

No PL/I-SQL os dados já são tratados em um nível bastante alto. O programador da transação não precisava se preocupar com acesso a registros, leitura em arquivos, indexação, o que foi uma imensa vantagem introduzida pelo SQL e que perdura até os dias de hoje. Referente as características ACID, PL/I-SQL implementava as seguintes primitivas:

BEGIN: Início de uma transação;

SAVE: No caso de erro, a transação não precisaria retornar ao início. Permite *backtraking*.

UNDO: Desfaz a transação até o ultimo SAVE.

ABORT: Desfaz a transação completamente.

COMMIT: Valida a transação.

O programador das transações precisaria se preocupar com recuperabilidade apenas neste nível. Se for feita uma validação, o SGBD garante que a transação perdura. Se foi feito um ABORT, o SGBD faz com que o banco de dados se comporte como se a transação nunca tivesse existido.

0 restante, no que diz respeito a recuperabilidade, fica no nível mais baixo do Sistema R, que é o *Sistema de Armazenamento de Dados*. Dentro do Sistema de Armazenamento de Dados fica toda a lógica do algoritmo para recuperação no caso de falhas.

O algoritmo de recuperação no caso de falhas do Sistema R utiliza espelhamento, mas não força a escrita das páginas referentes aos dados modificados pela transação a ser validada. Ao invés disto, utiliza um *log* onde guarda o valor antigo do dado alterado (para operações do tipo desfazer) e o valor novo (para operações do tipo refazer). O Sistema R faz periodicamente um *checkpoint*, que nada mais é que uma cópia em mídia estável das páginas espelhadas, a partir de onde o algoritmo pode aplicar o *log* para obter o último estado consistente antes da falha. O tempo de recuperação é proporcional à idade do último *checkpoint* e com estas características, que serão melhor detalhadas nas próximas subseções, é possível garantir as propriedades ACID no SGBD.

4.1 Arquivos

Os arquivos dentro do sistema R são tratados de forma semelhante ao descrito na Seção 3.1. Duas diferenças são identificáveis entretanto. Em primeiro lugar, o Sistema R permite arquivos não espelhados. Veja Figura 2. Para estes, não é dada nenhuma garantia no caso de falhas, podendo haver perda das propriedades ACID.

Outra diferença é que a validação não força a escrita das páginas referentes a dados alterados em mídia estável no momento da validação. Em vez disso, o algoritmo simplesmente salva um registro no *log* com dados necessários para se

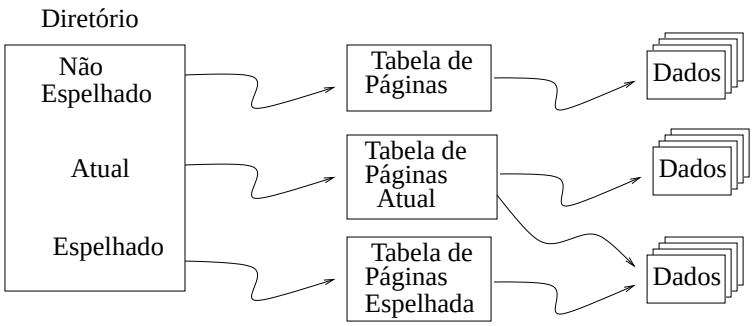

Figura 2: Sistema de Arquivos do Sistema R.

refazer ou desfazer a operação e faz a cópia dos dados correntes sobre os dados espelhados, a princípio, apenas em memória (a sincronização com o disco fica a cargo do sistema operacional).

4.2 Característica do *Log*

O *log* no Sistema R é um arquivo seqüencial incremental que guarda informações diversas sobre as transações. As principais são BEGIN, UPDATE e COMMIT. O registro de UPDATE no *log* traz informações para desfazer e refazer a transação (valor novo e valor antigo do dado modificado) e é um registro fundamental para obtenção de um estado consistente após uma falha do sistema. Por ser crítico, o *log* deve ser duplicado, aumentando assim sua confiabilidade. Mesmo sendo guardado em mídia estável, não é recomendável que seja salvo diretamente em fita pois são muito comuns as operações de ABORT e estando em fita, seria necessário reposicionar a fita antes de dar continuidade ao sistema.

4.3 *Commit*

No momento da validação, deve ser escrito um registro de COMMIT no *log* e forçar a escrita do *log* em mídia estável até este ponto. A partir do momento que este registro foi escrito em mídia estável, deve-se copiar, mesmo que em memória volátil, a tabela de páginas atual sobre a tabela de páginas espelhadas

relativa a transação validada. A utilização do *log* traz um ganho de desempenho em relação aos algoritmos descritos na Seção 3 pois uma página de *log* pode conter um registro de COMMIT de dezenas de transações e cada transação pode ter feito atualização em centenas de páginas. Utilizando E/S em disco como medida e lembrando que cada página alterada tem pelo menos um registro de UPDATE no *log*, o ganho em eficiência pode chegar a centena de vezes.

4.4 *Checkpoint*

Checkpoint é uma espécie de foto do banco de dados em um determinado momento e seu objetivo é minimizar o trabalho no caso de uma falha. No caso do Sistema R o *checkpoint* consiste basicamente em salvar todas as páginas espelhadas em mídia estável. Além das páginas espelhadas, algumas informações de sincronismo entre *log* e banco de dados também são salvas.

4.5 Reiniciando o Sistema

A Figura 3 mostra um banco de dados antes de ocorrer uma falha e seus cinco tipos possíveis de transações que precisam ser analisadas.

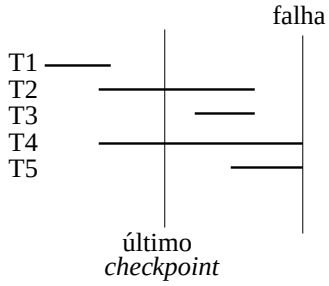

Figura 3: Exemplo de um sistema antes de uma falha.

As transações podem ter uma das seguintes características:

- Começar e terminar antes do último *checkpoint* (T1),
- começar antes e terminar depois do último *checkpoint* (T2),

- começar depois e terminar depois do último *checkpoint* (T3) ,
- começar antes do último *checkpoint* e não terminar (T4) e
- começar depois do último *checkpoint* e não terminar (T5).

Havendo uma falha do sistema, o que o algoritmo faz é buscar pelo último *checkpoint* que guardou as páginas espelhadas em mídia estável e recolocá-las no lugar das que foram perdidas no momento da falha do sistema. A partir deste ponto deve-se repassar no *log* por todas as transações ativas no momento do *checkpoint* ou em momento posterior.

Transações do tipo T1 já estão salvas no último *checkpoint* e não precisam ser consideradas; transações do tipo T2 precisam ser completadas ; transações do tipo T3 precisam ser refeitas; transações do tipo T4 precisam ser desfeitas; e transações do tipo T5 podem ser ignoradas. As operações de desfazer, refazer e completar uma transação são feitas utilizando-se os registros de UPDATE do *log* que contêm os valores novos e antigos dos dados alterados. Aplicando estas operações, chega-se no último estado consistente no momento da falha.

Se houver uma falha de um disco de armazenamento, mais rara e mais grave que uma falha do sistema onde apenas a memória volátil é perdida, deve-se pegar o último *back-up* e utilizá-lo como se fosse um *checkpoint* antigo. Se a falha for em um dos discos do *log*, deve-se utilizar a cópia não afetada (lembrando que o *log* está em mídia duplicada) e reconstruir o sistema.

5 Comparação Entre ARIES e o Sistema R

A similaridade entre o algoritmo de recuperação de falhas do Sistema R e ARIES é considerável. Tanto o Sistema R quanto ARIES (conforme visto em aula) suportam concorrência e têm, em geral, menos de 1 E/S por transação validada. Ambos utilizam *checkpoint*, garantem as propriedades ACID, não são baseados em forçar a escrita, utilizam *log* e podem ter granularidade de bloqueio equivalente ao dado alterado.

No que concerne às diferenças, ARIES possui um *checkpoint* menos custoso, como veremos três parágrafos abaixo. ARIES utiliza WAL (escrita antecipada no *log*) e *in place updating* (o dado alterado sobrescreve o dado antigo) em contraste com atualizações em sistemas baseados em espelhamento, sendo este o caso do Sistema R, no qual todas as páginas alteradas são duplicadas.

Outra diferença que afeta positivamente o desempenho de ARIES é a utilização de REDO baseado em páginas (*page-based redo*), em contraste com Sistema R no qual o REDO é baseado em variáveis (*logical redo*). O REDO do Sistema R é seletivo, ao passo que ARIES faz uma repetição da história. Repetição da história significa a repetição de todas as transações, vencedoras ou não, até o banco de dados chegar no estado equivalente ao do momento da falha. O *Log* do ARIES possui um registro chamado CLR (*compensation log record*) e o Sistema R não. Veremos adiante as implicações positivas do CLR.

Na Tabela 3 encontra-se um quadro comparativo entre os dois sistemas, cujas diferenças serão analisadas a seguir.

Pelo que foi descrito na seção 4.4 e no artigo Gray *et al.* [4], a execução de um *checkpoint* no Sistema R é custoso e chega a perturbar o banco de dados, pois consiste em passar para mídia estável todas as páginas espelhadas e mais algumas informações de sincronismo com *log*. O *checkpoint* do ARIES, por sua vez, é bem menos custoso, em razão de copiar a tabela de transações e a tabela de páginas "sujas" e não as páginas "sujas" propriamente. Esta diferença confere ao ARIES um dos mais significativos ganhos de desempenho em relação aos seu precursor, Sistema R, durante o funcionamento normal.

Ademais, ARIES também é mais eficiente na utilização de recursos de memória. O motivo é facilmente inteligível: ele utiliza alteração no local (sem duplicação). O Sistema R é baseado em espelhamento e duplicação, o que faz com que a eficiência de utilização da memória caia em 50% nos dados alterados pelas transações. Como a utilização mais eficiente da memória pode evitar escritas desnecessárias ao disco (utilização de *swap*), que são em torno de 100 mil vezes mais lentas que escritas na memória principal, o desempenho de ARIES pode ter um ganho considerável em relação ao Sistema R.

	ARIES	Sistema R
Suporta Concorrência	$\surd$	$\surd$
Em geral < 1 E/S por transação validada	$\surd$	$\surd$
Utiliza *checkpoint*	$\surd$	$\surd$
Checkpoint custoso		$\surd$
Garante ACID	$\surd$	$\surd$
Baseado em forçar a escrita		
Utiliza *Log*	$\surd$	$\surd$
Alta granularidade de bloqueio	$\surd$	$\surd$
Utiliza a técnica WAL (*write ahead log*)	$\surd$	
Utiliza espelhamento (*shadow*)		$\surd$
REDO seletivo		$\surd$
UNDO antes do REDO		$\surd$
REDO baseado em páginas	$\surd$	
REDO baseado em variáveis		$\surd$
Repetição da história	$\surd$	
Uso do CLR (*compensation log record*)	$\surd$	
Alteração no local (sem duplicação)	$\surd$	
Mais eficiente no funcionamento normal	$\surd$	
Mais eficiente na recuperação		$\surd$

Tabela 3: Comparação entre Sistema R e ARIES.

Concluídas as comparações entre os dois algoritmos durante o funcionamento normal do sistema, onde ARIES obteve grandes vantagens, passemos agora a analisar as diferenças entre os algoritmos durante a recuperação.

Sistema R possui um REDO seletivo baseado em variáveis, com UNDO antes do REDO. ARIES possui REDO baseado em páginas, com repetição da história e REDO antes do UNDO.

A recuperação no Sistema R é extremamente simples, sem deixar de ser correta. Trata-se de selecionar as transações que precisam ser desfeitas (transações

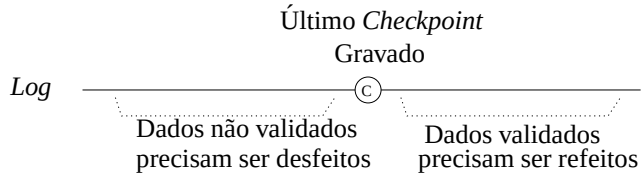

Figura 4: Visão simples da recuperação no Sistema R

perdedoras) e desfazê-las, para, em seguida, selecionar as transações que precisam ser refeitas (transações vencedoras) e refazê-las. Veja Figura 4.

A recuperação do algoritmo ARIES é mais complexa, pois o primeiro passo consiste em recontruir as tabelas páginas sujas e de transações referentes ao momento da falha (fase de análise) para então começar a repetir todo o histórico das transações, através do *log*, até chegar ao estado do momento da falha (fase refazer). Neste ponto, existe uma série de transações ativas que não sofreram validação e que são, então, desfeitas (fase desfazer).

Deve estar clara uma relação de custo-benefício em que, para ter um *checkpoint* e um funcionamento eficiente durante o funcionamento normal (benefícios), ARIES teve que pagar o custo de reconstruir as páginas "sujas", optando, para tanto, pela repetição do histórico.

Cabe ressaltar que ARIES, mesmo sendo mais complexo e custoso na fase de recuperação, provê certas otimizações, por fazer o REDO baseado em páginas, ao contrário do Sistema R, que é baseado em variáveis (*logical redo*). No caso do REDO baseado em páginas, o registro do *log* contém a página alterada, que é acessada diretamente, sem ter de atravessar uma árvore de índices ou consultar outras tabelas e páginas. O mesmo já não acontece no Sistema R, que guarda a variável que foi alterada, mas a página que ela se encontra ainda precisa ser descoberta, consultando tabelas e árvores de índice, que acabam precisando ser refeitas.

Como última diferença a ser analisada neste estudo entre os dois algoritmos de recuperação, existe o fato de ARIES utilizar CLR e Sistema R não. CLR é um registro do *log* escrito toda vez que uma atualização é desfeita. Tem a característica de guardar o dado novo, para o caso de precisar ser refeito, mas não

guarda o dado antigo, pois não é esperado que um “desfazer” seja desfeito. Com CLR, um registro de *log* não é desfeito mais de uma vez, mesmo havendo falhas sucessivas. Traz também a possibilidade de tratar bloqueio de variáveis: por exemplo desfazer parte de uma transação vítima de um *dead lock* utilizando um registro CLR.

6 Conclusão

O presente trabalho foi elaborado de forma a complementar o material sobre recuperação tratado no curso *MO410/A - 2004.1 Banco de Dados* sob orientação do professor doutor Geovane Cayres Magalhães, durante o qual tivemos a chance de nos dedicar, em alto grau de detalhamento, ao algoritmo ARIES.

No presente estudo, a preocupação principal foi apresentar os métodos predecessores, como os que se baseiam em forçar a escrita, com ou sem espelhamento, e o algoritmo de recuperação de falhas do Sistema R. Além disso, foi feita uma criteriosa comparação entre o Sistema R e ARIES.

Quanto ao objetivo prático deste trabalho, cabe mencionar três características que o fazem relevantes para os estudiosos do tema:

Primeiramente, trata-se de um documento didático. As Seções 2 e 3 podem servir como texto auxiliar para aqueles que estejam tendo um primeiro contato com sistemas tolerante a falhas, pois essas seções são dedicadas a uma descrição detalhada das caracterísicas de sistemas tolerantes a falha e algoritmos básicos, baseados em forçar a escrita, para se obter tais características.

Em segundo lugar, apesar da descrição do Sistema R e de ARIES estar contida no trabalho de Gray *et al.* [4] e no trabalho de C. Mohan *et al.* [10], respectivamente, onde se pode encontrar, de forma espalhada, parte da informação da Seção 5, este trabalho revela-se pioneiro, segundo nosso conhecimento, em razão de compilar tal informação de forma concisa, como, por exemplo, na Tabela 3.

Ademais, cabe ressaltar que o documento está em português, o que permite a que brasileiros interessados em recuperação e que não tenham o domínio completo da lígua inglesa possam ter um contato com técnicas de recuperação. O único

trabalho em português que trata simultaneamente do Sistema R e de ARIES é o trabalho de Tula Kraiser* [6], aluna da PUC-RJ, cuja interseção com o presente é pequena.

Referências

[1] Rakesh Agrawal and David J. Dewitt. Integrated concurrency control and recovery mechanisms: design and performance evaluation. *ACM Press*, Volume 10, December 1985.

[2] Margaret H. Eich. A classification and comparison of main memory database recovery techniques. In *Proceedings of the Third International Conference on Data Engineering*, pages 332–339. IEEE Computer Society, 1987.

[3] Michael J. Franklin. Concurrency control and recovery. *The Computer Science and Engineering Handbook*, pages 1058–1077, 1997.

[4] Jim Gray, Paul McJones, Mike Blasgen, Bruce Lindsay, Raymond Lorie, Tom Price, Franco Putzolu, and Irving Traiger. The recovery manager of the system r database manager. *ACM Comput. Surv.*, 13(2):223–242, 1981.

[5] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, 1983.

[6] Tula Kraiser. Recuperação de banco de dados. Disciplina de Banco de Dados, PUC-RJ.

[7] Dean Kuo. Model and verification of a data manager based on aries. *ACM Trans. Database Syst.*, 21(4):427–479, 1996.

[8] Raymond A. Lorie. Physical integrity in a large segmented database. *ACM Trans. Database Syst.*, 2(1):91–104, 1977.

*Ao digitar <”Sistema R” ARIES> no Google (24/07/2004) este foi o único documento em português a retornar.

[9] C. Mohan. Commit-lsn: a novel and simple method for reducing locking and latching in transaction processing systems. In *Proceedings of the sixteenth international conference on Very large databases*, pages 406–418. Morgan Kaufmann Publishers Inc., 1990.

[10] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. Aries: A transaction method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):145, March 1992.

[11] C. Mohan and Frank Levine. Aries/im: an efficient and high concurrency index management method using write-ahead logging. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 371–380. ACM Press, 1992.

[12] Ramakrishnan and Gehrke. *Database Management Systems*, chapter Crash Recovery. McGraw-Hill Higher Education, 2003. Tradução dos Slides: Amanda Meincke Melo, Janaína Ruas e Luís Augusto Meira http://www.dcc.unicamp.br/˜meira/temp/ .

[13] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2003.

[14] Andreas Reuter. Performance analysis of recovery techniques. *ACM Trans. Database Syst.*, 9(4):526–559, 1984.

[15] Kurt Rothermel and C. Mohan. Aries/nt: A recovery method based on write-ahead logging for nested transactions. In Peter M. G. Apers and Gio Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases, August 22-25, 1989, Amsterdam, The Netherlands*, pages 337–346. Morgan Kaufmann, 1989.

[16] W. E. Weihl. The impact of recovery on concurrency control. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 259–269. ACM Press, 1989.