[MAC0426] Sistemas de Bancos de Dados
[IBI5013] Bancos de Dados para Bioinformática
Aula 25
Aula 25 – Recuperação de Bancos de Dados

Kelly Rosa Braghetto

DCC–IME–USP

14 de junho de 2016

Esboço da recuperação

- Em geral,**recuperar-se de uma falha** de transação significa **restaurar o BD ao estado consistente mais recente** antes da falha
- Para isso, o SGBD precisa das informações sobre as mudanças aplicadas nos itens de dados, registradas no ***log* do sistema**

Esboço da recuperação

Estratégia típica de recuperação:

1. Se o dano no BD tiver sido extensivo devido a uma falha catastrófica, o método de recuperação:
   - 1.1 restaura um *backup* (antigo) do BD
   - 1.2 reconstrói um estado mais recente, reaplicando as operações de transações confirmadas do *log* armazenado no *backup*
2. Se o BD no disco não está danificado fisicamente e se uma falha não cadastrófica tiver ocorrido, a recuperação involve identificar as mudanças que podem ter causado uma inconsistência no BD
   - É possível realizar a recuperação somente usando as entradas no *log* do sistema
   - Ex: uma transação é confirmada, mas ocorre uma falha antes de todas suas operações de escrita terem sido gravadas no disco $\Rightarrow$ é preciso refazer algumas operações a fim de se restaurar um estado consistente no BD

Recuperação de falhas de transação não catastróficas

Técnicas principais de recuperação para esse tipo de falhas:

- **Atualização adiada**
- **Atualização imediata**
- **ARIES**
- **Paginação de sombra** – não veremos em detalhes

Recuperação de falhas de transação não catastróficas
Técnicas principais

Atualização Adiada (ou algoritmo NO-UNDO/REDO)

Só atualiza fisicamente o BD no disco **depois** que uma transação atinja seu ponto de confirmação.

As atualizações:

- Durante a execução da transação, são registradas no espaço local de trabalho da transação ou nos *buffers* do *cache* de memória principal do SGBD
- Antes da confirmação da transação, são registradas permanentemente no *log*
- Após a confirmação, são gravadas no banco de dados no disco

Recuperação de falhas de transação não catastróficas
Técnicas principais

Atualização Adiada (ou algoritmo NO-UNDO/REDO)

Só atualiza fisicamente o BD no disco **depois** que uma transação atinja seu ponto de confirmação.

- Se uma transação falhar antes da confirmação, ela não terá alterado o banco de dados, por isso não precisará ser desfeita. (NO-UNDO)
- Pode ser necessário refazer o efeito das operações de uma transação confirmada no *log*, pois ele pode não ter sido registrado no disco. (REDO)

Recuperação de falhas de transação não catastróficas
Técnicas principais

Atualização Imediata (ou algoritmo UNDO/REDO)

O BD pode ser atualizado *por algumas operações* de uma transação **antes** que ela alcance seu ponto de confirmação.

- Essas operações precisam ser registadas no *log* no disco antes que sejam gravadas no BD no disco (para possibilitar a recuperação)
- Se uma transação falhar antes da confirmação, mas depois de já ter gravado algumas atualizações no BD, ela precisará ser desfeita. (UNDO)
- Pode também ser necessário refazer o efeito de operações registradas no *log*, mas que podem não ter sido registradas no disco. (REDO)

Recuperação de falhas de transação não catastróficas
Técnicas principais

Atualização Imediata (ou algoritmo UNDO/REDO)

O BD pode ser atualizado *por algumas operações* de uma transação **antes** que ela alcance seu ponto de confirmação.

- O algoritmo UNDO/REDO é usado na prática

Variação do algoritmo UNDO/REDO:
algoritmo UNDO/NO-REDO

Todas as atualizações precisam ser registradas no BD em disco antes que a transação confirme.

Operações UNDO e REDO

- As operações UNDO e REDO precisam ser **idempotentes** – a execução de uma operação dessas várias vezes deve ser equivalente a executá-la uma só vez
- O processo de recuperação inteiro deve ser idempotente
  - Podem ocorrer falhas durante o processo de recuperação; portanto, suas operações devem poder ser reexecutadas

"O resultado da recuperação de uma falha do sistema durante a recuperação deve ser igual ao resultado da recuperação quando não há falha durante esse processo."

Outros conceitos importantes

Para podermos ver mais detalhes sobre as técnicas de recuperação, é preciso entender alguns conceitos básicos:

- *Caching* de blocos de disco
- *Write-ahead Logging* (WAL)
- Técnicas *steal/no-steal* e *force/no-force*
- *Check point* e *check point fuzzy*
- *Rollback* e *rollback* em cascata
- Ações de transação que não afetam o BD

Caching de blocos do disco

- O SGBD mantém algumas páginas de disco do BD em *buffers* (no *cache*) em memória principal
- Atualizações de itens de dados geralmente são feitas nos *buffers* e, quando conveniente, os *buffers* são gravados de volta para o disco
- O SGDB mantém uma tabela com informações sobre as páginas de disco do BD que estão atualmente nos *buffers* em memória
- Associados a cada *buffer* na tabela, estão dois bits: o **bit sujo** e o bit de **preso-solto**

Caching de blocos do disco

O bit sujo

- O **bit sujo** indica se o *buffer* ao qual está associado foi modificado ou não
- Quando uma nova página do BD é trazida do disco para um *buffer* do *cache*, uma nova entrada para a página é criada na tabela do SGBD
  - O bit sujo dessa nova entrada na tabela é zero
- Quando o *buffer* é modificado, o bit sujo na entrada da tabela correspondente é definido como 1
- Quando o conteúdo do *buffer* é substituído no (= esvaziado do) *cache*, ele antes deve ser gravado de volta no disco **somente se o bit sujo valer 1**

Caching de blocos do disco

O bit de preso-solto

- Uma página no cache está **presa** (bit de preso-solto com valor 1) se ainda não puder ser gravada de volta no disco
- Por exemplo, o protocolo de recuperação pode impedir que certas páginas do *buffer* sejam gravadas no disco até que as transações que mudaram o *buffer* tenham sido confirmadas

Esvaziamento de buffers modificados para o disco

Estratégia 1: **Atualização no Local**

- Grava o *buffer* no mesmo local de disco original – modifica no disco o valor antigo de quaisquer itens de dados alterados
- Só a AFIM é mantida no disco
- Técnica mais empregada na prática

Estratégia 2: **Sombreamento**

- Grava um *buffer* atualizado em outro local no disco – mantém múltiplas versões dos itens de dados
- Tanto a BFIM quanto a AFIM podem ser mantidas no disco
- Elimina a necessidade de se manter um *log* para a recuperação

BFIM – *before image* (valor antigo); AFIM – *after image* (valor novo)

Write-Ahead Logging (WAL)

- Atualização no local depende de um *log* para a recuperação
- Nesse caso, o mecanismo de recuperação precisa garantir o processo *write-ahead logging* (WAL), com os seguintes passos ordenados:
  1. a BFIM do item de dados deve estar registrada no *buffer* de *log* atual no *cache* do SGBD
  2. esse *buffer* de *log* deve ser esvaziado para o disco
  3. a BFIM deve ser modificada pela AFIM no BD em disco
- O WAL é necessário para possibilitar o UNDO

Tipos de informação incluída no *log* para uma **operação de gravação** (`write_item`)

Informação necessária para REDO

- Uma entrada de *log* do tipo REDO inclui a **AFIM** (valor novo) do item gravado pela operação
  - Necessário para **refazer** o efeito da operação com base no *log*

Informação necessária para UNDO

- Uma entrada de *log* do tipo UNDO inclui a **BFIM** (valor antigo) do item gravado pela operação
  - Necessário para **desfazer** o efeito da operação com base no *log*

Em um algoritmo UNDO/REDO, os dois tipos de entrada de *log* são combinados.

Regras que controlam quando uma página do BD pode ser gravada do *cache* para o disco

steal / no-steal

- **Técnica de recuperação *no-steal***: não permite que uma página do *buffer* atualizada por uma transação possa ser gravada em disco antes da confirmação da transação
  - O bit de preso-solto é usado para indicar quando a página pode ser gravada no disco
  - A regra do *no-steal* garante que o UNDO nunca será necessário na recuperação

Regras que controlam quando uma página do BD pode ser gravada do *cache* para o disco

steal / no-steal

- **Técnica de recuperação *steal***: permite gravar uma página do *buffer* atualizada antes que a transação confirme
  - É usada quando o gerenciador de *cache* precisa de um *buffer* para outra transação e, por essa razão, substitui uma página existente que tinha sido atualizada, mas cuja transação não foi confirmada ainda –> página é gravada no disco para que o *buffer* dela possa ser ocupado por outra página

Regras que controlam quando uma página do BD pode ser gravada do *cache* para o disco

force / no-force

- **Técnica de recuperação *force***: obriga que todas as páginas atualizadas por uma transação sejam gravadas em disco antes que a transação confirme
  - A regra do *force* garante que REDO nunca será necessário na recuperação (pois se a transação foi confirmada, então todas suas atualizações já foram gravadas em disco antes)
- **Técnica de recuperação *no-force***: caso contrário do *force* :)

Regras que controlam quando uma página do BD pode ser gravada do *cache* para o disco

steal / no-steal e *force / no-force*

- O esquema de recuperação NO-UNDO (= atualização adiada) segue uma técnica *no-steal*
- Porém, SGBDs tipicamente usam uma estratégia *steal/no-force* porque:
  - o *steal* evita a necessidade de um espaço para *buffers* muito grande para manter na memória todas as páginas atualizadas por transações ainda não confirmadas
  - o *no-force* permite que uma página atualizada por uma transação já confirmada ainda possa estar na memória quando outra transação precisar atualizá-la (economizando o custo de múltiplas leituras e gravações)

Protocolo *logging write-ahead* para um algoritmo de recuperação UNDO/REDO

1. A BFIM de um item não pode ser subtituída pela AFIM no BD em disco até que todos os registros de *log* do tipo UNDO (até o ponto atual da execução) para a transação em atualização tenham sido gravados no disco.
2. A confirmação de uma transação não pode ser concluída até que todos os registros do tipo REDO e do tipo UNDO para essa transação tenham sido gravados à força no disco.

Check point no *log* do sistema

- Existe um outro tipo de entrada registrada no *log* do SGBD que se chama ***check point***
- Essa entrada tem a forma:
  `[checkpoint, <lista de transações ativas>]`
- É gravada no *log* periodicamente, justamente depois do sistema **gravar no BD em disco todos os *buffers* modificados**
- Toda transação T com uma entrada `[commit, T]` no *log* antes de uma entrada `[checkpoint]` **não precisa ter suas operações WRITE refeitas em caso de falha**, pois suas atualizações já foram registradas no BD em disco durante o *check point*

Check point no *log* do sistema

Com que intervalo o SGBD deve realizar um *check point*?

- a cada m minutos
- a cada t transações confirmadas desde o último *check point*

Ações envolvidas em um *check point*:

1. Suspender execução de transações temporariamente
2. Forçar a gravação em disco de todos os *buffers* modificados
3. Gravar um registro `[checkpoint]` no *log* e forçar a gravação do *log* em disco
4. Retomar a execução das transações

Check point no *log* do sistema

Check point fuzzy

- Enquanto a gravação em disco dos buffers acontece, as transações ficam suspensas, o que pode atrasar muito suas execuções
- A técnica ***check point fuzzy*** pode reduzir esse atraso
  1. o sistema grava no *log* um registro `begin_checkpoint`
  2. o sistema realiza a gravação em disco dos *buffers*, ao mesmo tempo em que retoma a execução das transações
  3. quando o sistema termina a gravação dos *buffers*, ele grava no *log* um registro `end_checkpoint`
- Até que a gravação em disco dos *buffers* seja encerrada, o registro do *check point* anterior deve continuar válido (ou seja, ele que será usado como ponto de partida para a recuperação em caso de falha)

Rollback de transação

- Quando uma transação T falha depois de atualizar o BD, mas antes que seja confirmada, pode ser necessário reverter (***rollback***) T
  - Todo item de dados do BD que teve seu valor alterado por T, precisa ser restaurado para o seu valor anterior (BFIM)
  - As entradas de *log* do tipo UNDO são usadas nesse caso
- Se T for revertida, qualquer transação S que tenha lido o valor de algum item de dados X gravado por T, também deve ser revertida
  - E se S for revertida, qualquer transação R que tenha lido o valor de algum item de dados Y gravado por S, também deve ser revertida
  - ...

Rollback em cascata (propagação do cancelamento)

Rollback de transação

Rollback em cascata

- Pode ocorrer quando o protocolo de recuperação garante escalonamento *recuperáveis*, mas não *estritos* ou *sem propagação*
- Pode ser muito complexo e demorado
- Quase todos os mecanismos de recuperação são projetados de modo a garantir que **ele nunca seja necessário**
  - Quando não há possibilidade dele ocorrer, operações do tipo `read_item` não precisam ser registradas no *log*, já que elas só são úteis na recuperação para indicar quais transações precisam ser desfeitas em caso de *rollback em cascata*

Rollback em cascata – Exemplo (parte 1)

As operações de 3 transações

(a)

T_1	T_2	T_3
read_item(*A*)	read_item(*B*)	read_item(*C*)
read_item(*D*)	write_item(*B*)	write_item(*B*)
write_item(*D*)	read_item(*D*)	read_item(*A*)
	write_item(*D*)	write_item(*A*)

Rollback em cascata – Exemplo (parte 2)

O *log* do sistema até o ponto da falha

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

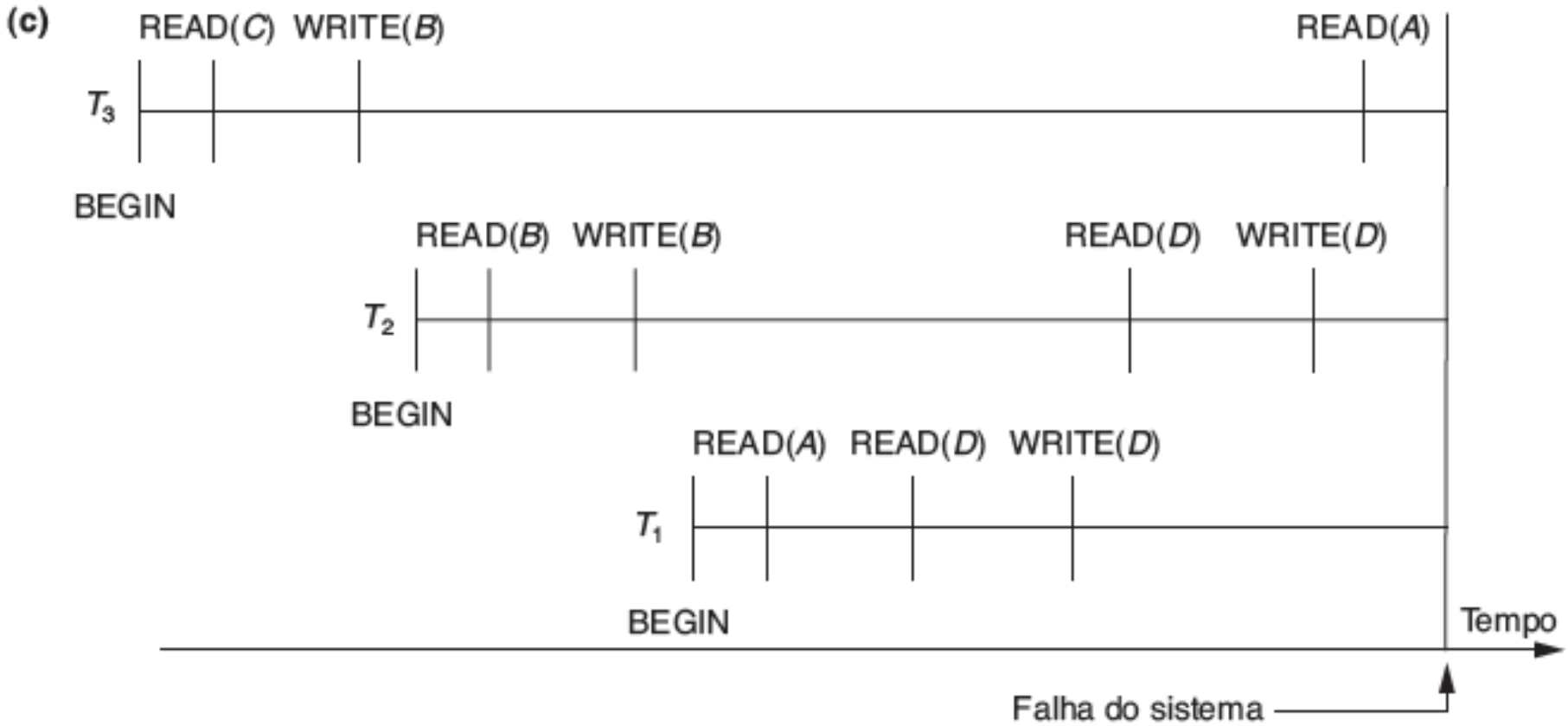
← Falha do sistema

* T_3 é revertido porque não atingiu seu ponto de confirmação.

** T_2 é revertido porque lê o valor do item B gravado por T_3.

Rollback em cascata – Exemplo (parte 3)

Linha do tempo das operações antes da falha



Ações de transação que não afetam o BD

- Transações podem ter ações que não afetam o BD
- Exemplos: impressão de mensagens, geração de relatórios a partir dos dados do BD
- Se a transação falhar antes de concluir, os usuários não deveriam ter "recebido" essas informações (que podem estar erradas!)
- Dados desse tipo só devem ser mostrados depois da confirmação da transação
- Abordagem comum para tratar isso: emitir os comandos que geram os relatórios, mas mantê-los como tarefas *batch*, executadas somente depois que a transação atinge seu ponto de confirmação. Em caso de falha da transação, as tarefas em *batch* são canceladas

Técnicas de recuperação baseada em Atualização Adiada

Ideia geral (relembrando do início da aula)

1. Durante a execução da transação, atualizações são registradas apenas no *log* e nos *bufers* do *cache*
2. Depois que a transação atinge seu ponto de confirmação
   - 2.1 o *log* é gravado em disco à força
   - 2.2 as atualizações são registradas no BD em disco

Em caso de falha da transação antes de seu ponto de confirmação, não é preciso desfazer nenhum operação.

Entradas necessárias no *log*

- somente entradas do tipo REDO são necessárias no *log* – incluem AFIM (valor novo) do item gravado pelo `write_item`
- entradas do tipo UNDO não são necessárias

Técnicas de recuperação baseada em Atualização Adiada

Protocolo de atualização adiada típico

1. Uma transação não pode mudar o banco de dados no disco até que atinja seu ponto de confirmação.
2. Uma transação não atinge seu ponto de confirmação até que todas suas entradas do tipo REDO sejam registradas no *log* e o *buffer* do *log* seja gravado à força no disco. (reafirmação do protocolo WAL)

- UNDO não é necessário porque nenhuma atualização é feita no BD em disco antes da confirmação da transação
- REDO é necessário caso o sistema falhe depois da atualizações terem sido gravadas no log e confirmadas, mas antes de todas terem sido gravadas no disco

Técnicas de recuperação baseada em Atualização Adiada

Inter-relação entre processos de controle de concorrência e recuperação

Exemplo: Considere um SGBD que usa:

- atualização adiada (NO-UNDO/REDO)
- **bloqueio bifásico estrito** para lidar com concorrência (múltiplos usuários)
  - Os itens usados na transação devem ficar bloqueados até o ponto de confirmação da transação
  - Isso garante escalonamentos estritos e seriáveis
- entradas `[checkpoint]` no *log*

Um possível algoritmo de recuperação para esse cenário, chamado aqui de `RAA_M` (*recuperação usando atualização adiada em ambiente multiusário*) é descrito nos dois próximos slides.

Técnicas de recuperação baseada em Atualização Adiada

Procedimento `RAA_M` (NO-UNDO/REDO com checkpoint)

- Use 2 listas de transações mantidas pelo SGBD
  - T – as transações confirmadas desde o último *check point*
  - T' – as transações ativas (que ainda não confirmaram)
- Para cada `WRITE_OP` registrada no *log*, onde `WRITE_OP` é uma entrada do tipo `WRITE` de alguma transação em T (obs: **as `WRITE_OP` devem ser percorridas na ordem em que foram gravadas no *log*)** faça:
  - Execute `REDO(WRITE_OP)`
- Cancele as transações em T' e submeta-as novamente

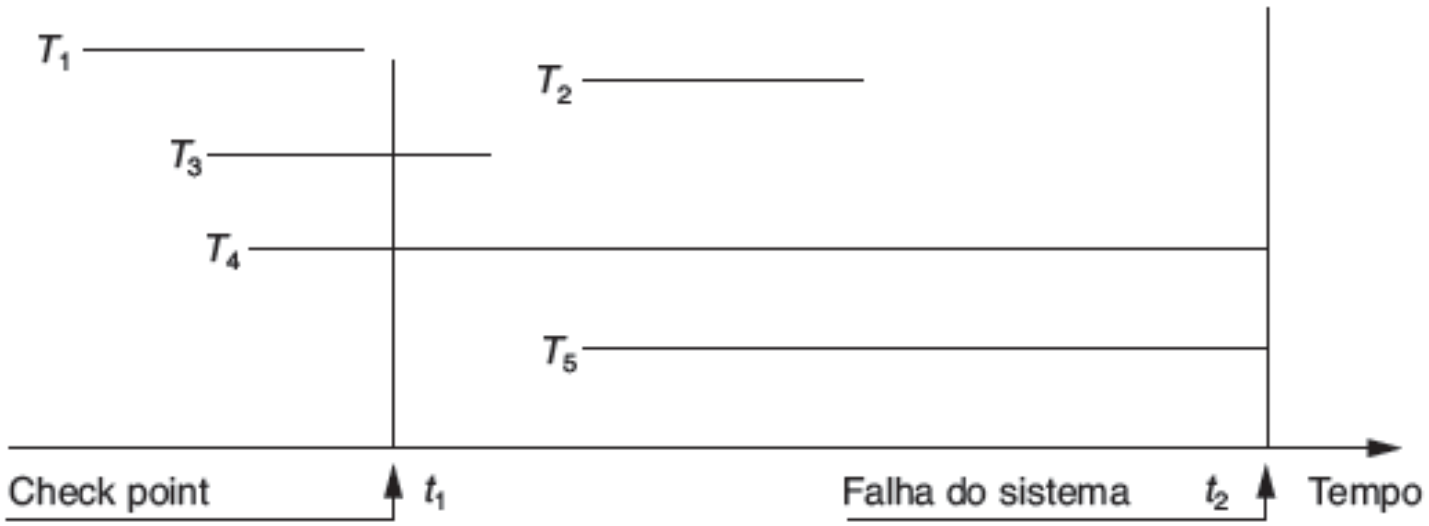
Técnicas de recuperação baseada em Atualização Adiada

Procedimento `REDO(WRITE_OP)`, usado em `RAA_M`

- Refaça a operação `write_item` registrada em `WRITE_OP`
  1. examinando sua entrada de *log* `[write_item, t, X, valor_novo]`
  2. atribuindo `valor_novo` (AFIM) ao item de dados do BD `X`

O método de recuperação RAA_M

Exemplo de uma linha de tempo, para ilustrar o efeito *check point*



Segundo o alg. RAA_M, T_2 e T_3 seriam refeitas após a falha. E T_4 e T_5 seriam simplesmente canceladas (e talvez reiniciadas posteriormente), já que elas não foram confirmadas antes do último *check point* e, portanto, nenhuma de suas atualizações foi escrita no disco, por causa do uso do protocolo de atualização adiada. Como T_1 já havia sido confirmada antes do último *check point*, nada precisaria ser feito com ela.

O método de recuperação RAA_M

Exemplo de uma recuperação (parte 1)

(a)

T_1	T_2	T_3	T_4
read_item(*A*)	read_item(*B*)	read_item(*A*)	read_item(*B*)
read_item(*D*)	write_item(*B*)	write_item(*A*)	write_item(*B*)
write_item(*D*)	read_item(*D*)	read_item(*C*)	read_item(*A*)
	write_item(*D*)	write_item(*C*)	write_item(*A*)

Operações de 4 transações

O método de recuperação RAA_M

Exemplo de uma recuperação (parte 2)

(b)

[start_transaction,T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item,T_2, D, 25] ← Falha do sistema

T_2 e T_3 são ignorados porque não atingiram seus pontos de confirmação.

T_4 é refeito porque seu ponto de confirmação está depois do último check point do sistema.

Sobre o método de recuperação RAA_M

Contras

- limita a execução concorrente das transações (por conta do tipo de bloqueio usado)
- pode exigir um espaço para *buffer* excessivo (para manter os itens atualizados até que as transações sejam confirmadas)

Prós

- as operações das transações nunca precisam ser desfeitas, por dois motivos:
  - uma transação não registra nenhuma atualização no BD em disco até que complete sua execução com sucesso
  - uma transação nunca lerá um item gravado por uma transação não confirmada (não há *rollback* em cascata)

Técnicas de recuperação baseada em Atualização Imediata

Ideia geral (relembrando do início da aula)

- Quando uma transação emite um comando de atualização, o banco de dados no disco **pode ser atualizado imediatamente** (mais isso não é um requisito!)
  - Não há a necessidade de esperar que a transação atinja seu ponto de confirmação

Em caso de falha da transação, é preciso desfazer as atualizações já gravadas no disco.

Entradas necessárias no *log*

- entradas do tipo UNDO são necessárias no *log* – incluem BFIM (valor antigo) do item gravado pelo `write_item`
- também pode requerer entradas do tipo REDO (que incluem AFIM), como veremos a seguir

Técnicas de recuperação baseada em Atualização Imediata

- O métodos de atualização imediata seguem uma estratégia *steal* para decidir quando um *buffer* atualizado pode ser gravado no disco
- Na atualização adiada, a estratégia usada é *no-steal*

Técnicas de recuperação baseada em Atualização Imediata

"Categorias" de algoritmos de recuperação baseados em Atualização Imediata

- Algoritmo **UNDO/NO-REDO** – garante que **todas as atualizações** de uma transação são gravadas no BD em disco **antes da confirmação da transação** –> REDO não é necessário
  - precisa usar também a estratégia *force* na gravação dos *buffers* no disco
- Algoritmo **UNDO/REDO** – permite que a transação seja confirmada antes que todas as mudanças sejam gravadas no BD no disco
  - precisa usar também a estratégia *no-force*
  - essa é a técnica mais geral, mas também é a mais complexa

Técnicas de recuperação baseada em Atualização Adiada

Esboço de algoritmo de recuperação UNDO/REDO

Exemplo – Considere um SGBD que usa:

- atualização imediata
- entradas `[checkpoint]` no *log*
- **bloqueio bifásico estrito** para lidar com concorrência (múltiplos usuários)
  - Os itens usados na transação devem ficar bloqueados até o ponto de confirmação da transação
  - Isso garante escalonamentos estritos e seriáveis
  - Nesse tipo de bloqueio, podem ocorrer *deadlocks* –> pode haver o cancelamento de transações que, no caso de atualização imediata, implica na necessidade de UNDOs

Um possível algoritmo de recuperação para esse cenário, chamado aqui de `RAI_M` (*recuperação usando atualização imediata em ambiente multiusário*) é descrito nos dois próximos slides.

Técnicas de recuperação baseada em Atualização Adiada

Procedimento RAI_M (UNDO/REDO com checkpoint)

- Use 2 listas de transações mantidas pelo SGBD
  - T – as transações confirmadas desde o último *check point*
  - T' – as transações ativas (que ainda não confirmaram)
- Para cada WRITE_OP registrada no *log*, onde WRITE_OP é uma entrada do tipo WRITE de **alguma transação em T'** (obs: **as WRITE_OP devem ser percorridas na ordem reversa em que foram gravadas no *log*)** faça:
  - Execute UNDO(WRITE_OP)
- Para cada WRITE_OP registrada no *log*, onde WRITE_OP é uma entrada do tipo WRITE de **alguma transação em T** (obs: **as WRITE_OP devem ser percorridas na ordem em que foram gravadas no *log*)** faça:
  - Execute REDO(WRITE_OP)
    (Obs.: esse REDO é o mesmo definido para RAA_M)

Técnicas de recuperação baseada em Atualização Imediata

Procedimento `UNDO(WRITE_OP)`, usado em `RAI_M`

- Desfaça a operação `write_item` registrada em `WRITE_OP`
  1. examinando sua entrada de *log* `[write_item, t, X, valor_antigo]`
  2. atribuindo `valor_antigo` (BFIM) ao item de dados do BD `X`

Algoritmo ARIES

Algorithms for Recovery and Isolation Exploiting Semantics(ARIES)

- Exemplo de algoritmo de recuperação usado em grandes SGBDs (IBM DB2, MS SQL SERVER, etc.)
- Possui uma técnica *steal*/*no-force* para gravação de páginas no disco
- Minimiza o trabalho de recuperação, ao fazer progressos "incrementais" (*forward progress*) em casos de falhas na recuperação

ARIES

Baseado em 3 conceitos:

- ***write-ahead logging***
- **histórico repetitivo** – depois da falha, o ARIES retraça todas as ações realizadas antes da falha, para deixar o sistema no mesmo estado em que ele estava antes da falha ocorrer
  - Depois, ele desfaz as transações que estavam ativas (= não terminadas) no momento da falha
- ***logging* durante o UNDO** – mudanças feitas no BD enquanto as transações estão sendo desfeitas são gravadas no *log*, para garantir que uma ação não será repetida caso a recuperação precise reiniciar

Algoritmo ARIES

Possui três etapas principais

- **Análise**: obtém todas as informações necessárias do *log*. Identifica as páginas sujas (atualizadas) no *buffer* e a lista de transações ativas no momento da falha
- **REDO**: restaura o BD ao seu estado no momento da falha (incluindo também as mudanças feitas por transações ainda ativas no momento da falha)
- **UNDO**: desfaz todas as modificações feitas por transações não confirmadas, deixando o BD em um estado consistente

Algoritmo ARIES

Arquivo de log

- Todo registro no log tem um número monotonicamente crescente – o ***Log Sequence Number*** **(LSN)**.
  - O LSN indica o endereço do registro de *log* no disco
- Todo registro no *log* também mantém o **LSN anterior** da transação
  - O LSN anterior liga (em ordem reversa) os registros de *log* de uma transação
- Cada entrada de *log* que registra uma operação de escrita sobre um item de dado, registra também o ID da página que contém o item
- Um novo registro é gravado no *log* a cada vez que uma das seguintes operações ocorrer:
  - `write`, `commit`, `abort`, `undo` (desfazer uma transação), `end` (encerrar transação)

Algoritmo ARIES

Tabelas (mantidas em memória) para apoiar a recuperação eficiente:

- **Tabela de transações** – com uma entrada para cada transação ativa
  - Em cada entrada, mantém-se também o LSN do registro de *log* mais recente associado à transação (`ultimo_LSN`) e o status da execução da transação (ex.: em progresso, *commited*, *aborted*)
- **Tabela de páginas sujas**, com uma entrada para cada página suja no *buffer*
  - Em cada entrada, mantém-se o ID da página e o LSN correspondente à primeira atualização feita sobre a página depois que ela foi carregada para o *buffer*

Algoritmo ARIES

Funcionamento do **check point** no ARIES:

- Gravar um registro `begin_checkpoint` no *log*
- Gravar um registro `end_checkpoint` no *log*, junto com o conteúdo da Tabela de Transações e da Tabela de Páginas Sujas

Observações:

- *Check point fuzzy* é usado neste caso, para possibilitar que o SGBD continue a execução das transações enquanto o *check point* é realizado
- O conteúdo do cache não precisa ser gravado em disco durante o *check point*, porque as tabelas contêm as informações necessárias para a recuperação
- O LSN do registro `begin_checkpoint` é mantido em um arquivo especial, usado no processo de recuperação para

Algoritmo ARIES – Fase de Análise (parte 1)

- Após a ocorrência de uma falha, o gerenciador de recuperação começa a **fase de análise**, a partir do último registro `begin_checkpoint` registrado no *log*
- Quando o `end_checkpoint` é encontrado, carrega para a memória principal as tabelas de Transações e Páginas Sujas

Algoritmo ARIES – Fase de Análise (parte 2)

Na continuidade da análise, os registros de log sendo analisados podem causar mudanças nas tabelas:

- se encontrar um registro de fim para uma transação T que aparece na tabela, então T deve ser removida da tabela
- se algum outro registro for encontrado para uma transação T', então uma entrada para T' deve ser criada na tabela de transações (se ela não existir ainda) e o valor de `ultimo_LSN` para T' na tabela deve ser atualizado com o valor do LSN do registro atual
  - se o registro de *log* corresponde a uma efetivação (`commit` ou `abort`) da transação, então o status dela deve ser atualizado na tabela
  - se o registro de *log* corresponde a uma modificação na página P, então uma entrada para P deve ser criada na tabela de Páginas Sujas (se não existir ainda) e o LSN da entrada de P deve ser atualizado com o valor do LSN do registro atual

Algoritmo ARIES – Fase de REDO (parte 1)

- **A fase do REDO vem em seguida**; o ARIES primeiro identifica a posição no log onde ele precisa iniciar a fase de REDO
  - Ele encontra M = menor LSN entre as páginas mantidas na tabela de Páginas Sujas
- Para transações que podem ser refeitas, quaisquer mudanças com LSN $<$ M correspondem a alterações já feitas nos *buffers* ou que já foram aplicadas no disco
- Assim , o REDO começa no registro de *log* com LSN = M e "varre" para frente até o final do *log*

Algoritmo ARIES – Fase de REDO (parte 2)

Durante a fase de REDO, para cada mudança registrada no *log*, é preciso verificar se a mudança já foi aplicada ou não. Vamos chamar de *LSN_atual* o LSN do registro de *log* atual:

- Pule o registro de *log* se o *LSN_atual* modifica uma página P que não está na tabela de páginas sujas (porque essa modificação já está no disco)
- Pule o registro de *log* se o *LSN_atual* modifica uma página P que está na tabela de páginas sujas com um LSN > *LSN_atual* (porque a mudança já está presente)
- Se nenhum dos casos anteriores ocorrer, leia a página P do disco. Se o LSN armazenado nessa página for > que *LSN_atual*, a mudança já foi aplicada e a página não precisa ser regravada no disco. Senão, será preciso reaplicar a mudança.

Algoritmo ARIES – Fase de UNDO

- Após a fase de REDO, o BD estará no estado exato em que estava antes da ocorrência da falha
- O conjunto de transações ativas (= `undo_set`) foi identificado na fase de análise
- **A fase de UNDO** percorre o *log* de trás para frente, desfazendo as operações das transações no `undo_set`
  - um registro de *log* de compensação é gravado para cada ação desfeita
- Depois disso, o processo de recuperação termina
  - o processamento normal pode ser iniciado novamente

Exemplo de Recuperação em ARIES (Parte 1)

O estado do arquivo *log* no ponto da falha:

Lsn	Ultimo_lsn	Id_transacao	Tipo	Id_pagina	Outra_informacao
1	0	T_1	update	*C*	...
2	0	T_2	update	*B*	...
3	1	T_1	commit		...
4	begin check point				
5	end check point				
6	0	T_3	update	*A*	...
7	2	T_2	update	*C*	...
8	7	T_2	commit		...

As tabelas de Transações e de Páginas Sujas no momento do *check point*:

TABELA DE TRANSAÇÕES

Id_transacoes	Ultimo_lsn	Status
T_1	3	commit
T_2	2	in progress

TABELA DE PÁGINAS SUJAS

Id_pagina	Lsn
C	1
B	2

Exemplo de Recuperação em ARIES (Parte 2)

- A **fase de análise** começa no registro 4 do *log*
- Na análise do registro 5, as tabelas de Transações e Páginas Sujas são carregadas do *log* para a memória principal
- Na análise do registro 6, é criada uma nova entrada na tabela de Transações para T3, e uma nova entrada na tabela de Páginas Sujas para a página A
- A análise do registro 7 causa apenas a atualização do `Ultimo_LSN` de T2 na tabela de Transações
- Na análise do registro 8, o `Ultimo_LSN` de T2 é atualizado novamente e seu status na tabela é mudado para `commit`

Após a fase de análise, as tabelas ficam:

TABELA DE TRANSAÇÕES

Id_transacoes	Ultimo_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

TABELA DE PÁGINAS SUJAS

Id_pagina	Lsn
C	1
B	2
A	6

Exemplo de Recuperação em ARIES (Parte 3)

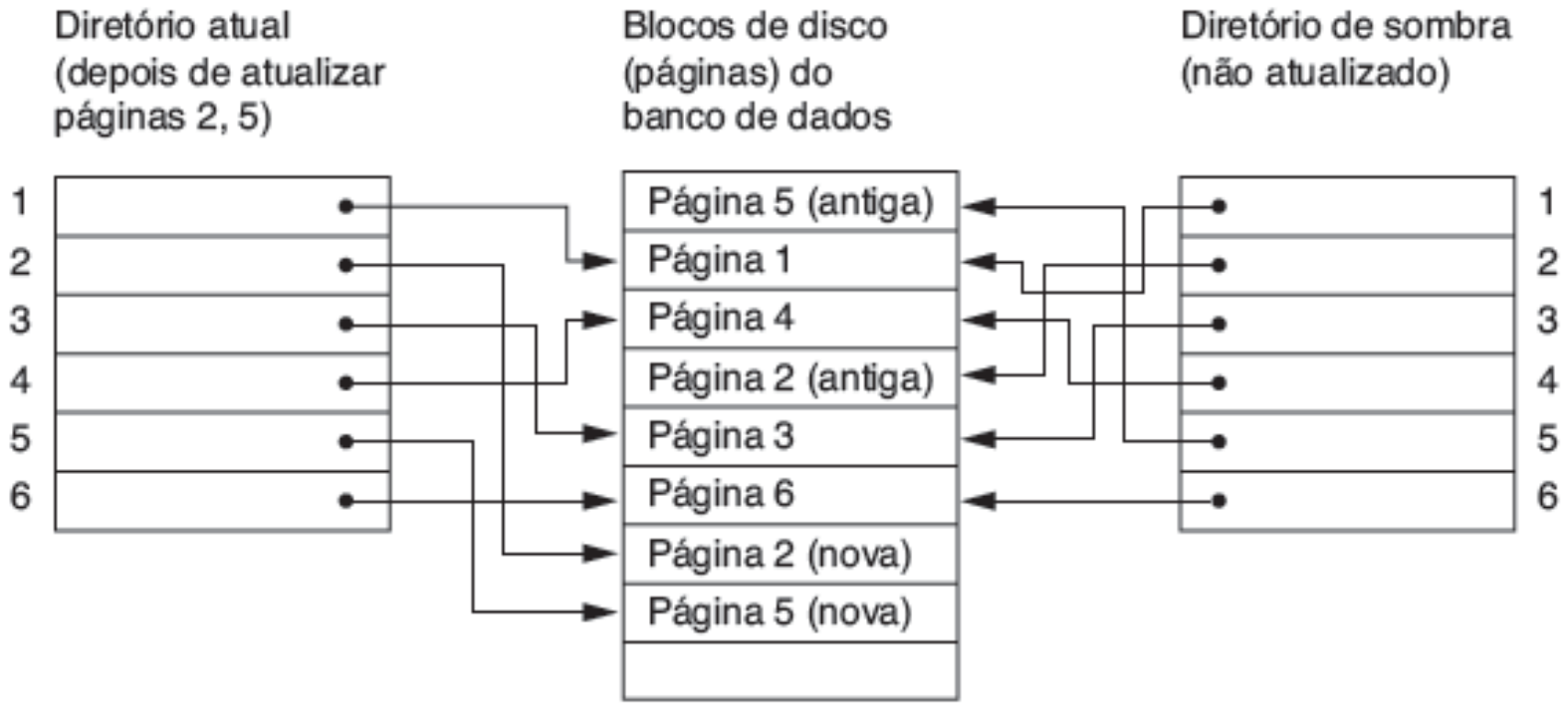
- Para a fase de REDO, M = 1 (menor LSN na tabela de Páginas Sujas)
- Começa-se no registro de *log* 1 e prossegue-se com o REDO das atualizações (registros com LSN 1, 2, 6 e 7)
- Os registros com LSN 1, 2, 6 e 7 correspondem a modificações nas páginas C, B, A e C, respectivamente
- Os LSNs na tabela de Páginas Sujas para C, B e A são 1, 2 e 6, respectivamente
- Portanto, em cada uma das modificações, o *LSN_atual* >= LSN associado à pagina modificada na tabela de Páginas Sujas. Assim, essas páginas serão lidas do disco novamente e as atualizações serão reaplicadas com base no *log* (supondo que os LSNs reais armazenados nessas páginas sejam menores que a entrada de *log* correspondente)

Exemplo de Recuperação em ARIES (Parte 4)

- Depois da recuperação das páginas com o REDO, começa a **fase do UNDO**
- Pela tabela de Transações, o UNDO só é aplicado à transação ativa T3
- O UNDO de T3 inicia no registro 6 (último atualização para T3) e prossegue de trás para frente no *log*
- A cadeia inversa de atualizações para T3 é então seguida e desfeita. No caso do exemplo, essa cadeia contém somente o registro 6.

Paginação de sombra (só uma pincelada)

Exemplo (monousuário)



Referências Bibliográficas

- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010. – Capítulo 23