CI218: Recuperação de Falhas

Profa. Carmem Hara

Departamento de Informática/UFPR

19 de janeiro de 2023

Gerenciador de Recuperação

- Responsável por garantir a atomicidade e durabilidade das transações
- **Atomicidade:** desfaz ações de transações não efetivadas
- **Durabilidade:** garante que ações de transações efetivadas não sejam perdidas por falhas do sistema ou de mídia

Tipos de Falhas

- Falha de transação:
  - Erro lógico: entrada errada ou condição para execução não satisfeita (p.ex saldo insuficiente)
  - Erro do sistema: Exemplos: timeout, deadlock
- Falha do sistema: defeito do hardware, software ou sistema operacional
- Falha de disco: o disco perde o conteúdo devido a uma falha do dispositivo ou falha na transferência dos dados.

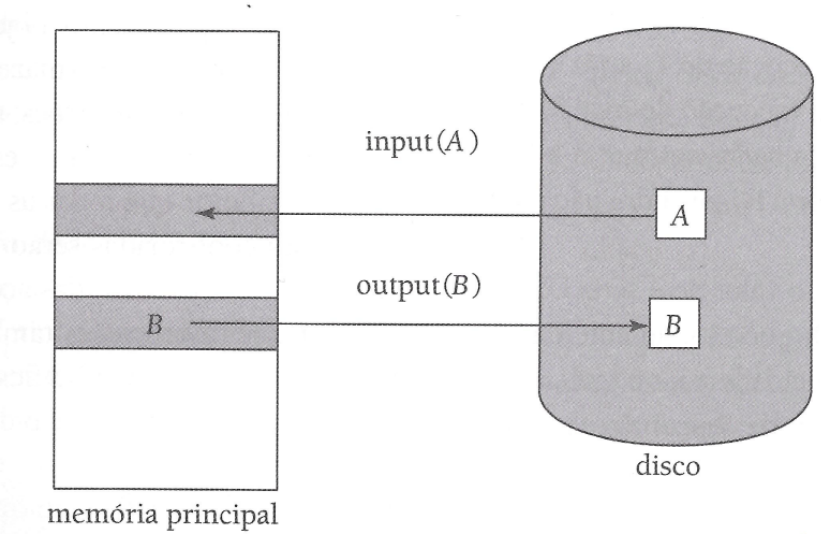
Recuperação de falhas

- ações durante o processamento da transação: p.ex. escrita das ações em logs e cópias
- ações após a falha para recuperar o conteúdo do banco de dados para um estado consistente e garantir a atomicidade e durabilidade

Tipos de armazenamento

- **Volátil:** não sobrevive a quedas no sistema
- **Não volátil:** Ex: discos e fitas magnéticas
- **Estável:** que "nunca"e' perdida
  - replicação em vários meios de armazenamento não volatil: Ex: disco espelhado - cópia de todos os blocos em discos separados
  - cópia dos dados em site remoto

Acesso aos Dados



- `read(x)`:
  1. se o bloco Bx (onde reside x) nao está na memoria principal então execute `input(Bx)`
  2. xi:= valor de x
- `write(x)`:
  1. se o bloco Bx nao está na memoria principal então execute `input(Bx)`
  2. x:= recebe o valor de xi

Sobre a Gravação de Dados

2 questões importantes:

- **Roubo:** permitir que outra transação utilize um frame do pool de buffers, forçando a gravação de uma página com dados alterados em disco antes da transação ser efetivada.
- **Imposição:** forçar a gravação de todas as alterações em disco assim que uma transação for efetivada

A recuperação é mais fácil com uma estratégia **sem roubo** e **com imposição**.

Recuperação baseada em Log

O **log** é uma sequência de **registros de log**, que guardam todas as ações de atualização no banco de dados.
Registros de log:

- `<Ti, start>`
  inicio da transação `Ti`
- `<Ti, X, val-ant, val-novo>`
  item de dado `X` foi atualizado de `val-ant` para `val-novo` pela transação `Ti`
- `<Ti, commit>`
  transação `Ti` foi efetivada
- `<Ti, abort>`
  trasação `Ti` foi cancelada

Modificação do Banco de Dados

- A gravação do registro de log deve ser feita ANTES da modificação do banco de dados
- A atualização de item de dado envolve as seguintes etapas:
  1. a transação modifica dados em uma área privada da memória principal
  2. os dados são modificados no bloco que contém o dado e que está no *buffer* em memória principal
  3. é feito `output(B)` para gravar o bloco em disco
- Os itens 2 e 3 compõem a ação de *modificar o banco de dados*
- **Modificação adiada:** a modificação do banco de dados quando a transação for efetivada
- **Modificação imediata:** a modificação é executada enquanto a transação está ativa

Operações de Recuperação

- `redo(Ti)`:
  para cada registro `<Ti, X, V1, V2>` modificar o valor de `X` no banco de dados para `V2`
- `undo(Ti)`:
  1. varrer o log **do contrário** e para cada `<Ti, X, V1, V2>`: escrever no log `<Ti, X, V1>` (registro de compensação)
  2. modificar o valor de `X` no banco de dados para `V1`
  3. quando encontrar `<Ti, start>` escrever no log `<Ti, abort>`

Exemplo

- Transação T0: transfere R$50 de A para B

T0:

```
read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);
```

Log:

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
```

- Transação T1: saque de R$100 de C

T1:

```
read(C);
C:=C-100;
write(C);
```

Log:

```
<T1, start>
<T1, C, 700, 600>
<T1, commit>
```

Modificação Adiada

Em um escalonamento serial:

- `write(X)`: escreve no log `<Ti, X, val-novo>`
- `commit`:
  1. escreve no log `<Ti, commit>`
  2. escreve registros de log em armazenamento estável
  3. o BD é modificado
- Recuperação de falhas:
  se o log contiver `<Ti,start>` e `<Ti,commit>` então executar `redo(Ti)`

Modificação Adiada - Falhas

```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
-- Falha --
```

Recuperação

```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 600>
-- Falha --
```

Recuperação

```
redo(T0);
```

```
<T0, start>
<T0, A, 950>
<T0, B, 2050>
<T0, commit>
<T1, start>
<T1, C, 600>
<T1, commit>
-- Falha --
```

Recuperação

```
redo(T0);
redo(T1);
```

Modificação Imediata

Em um escalonamento serial:

- `write(X)`:
  1. escreve no log `<Ti, X, val-antigo, val-novo>`
  2. o BD é modificado
- `commit`: escreve no log `<Ti, commit>`
- antes de ser executado um `output(B)`, todos os registros de log das modificações feitas no bloco B devem estar em armazenamento estável.
- Recuperação de falhas:
  - se o log contiver `<Ti,start>` e `<Ti,commit>` ou `<Ti, abort>` então executar `redo(Ti)`
  - se o log contiver `<Ti,start>` mas não contiver `<Ti,commit>` e nem `<Ti, abort>` então executar `undo(Ti)`

Modificação Imediata - Falhas

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
-- Falha --
```

Recuperação

```
undo(T0);
```

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, A, 1000>
<T0, B, 2000>
<T0, abort>
```

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 700, 600>
-- Falha --
```

Recuperação

```
redo(T0);
undo(T1);
```

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 700, 600>
<T1, C, 700>
<T1, abort>
```

```
<T0, start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0, commit>
<T1, start>
<T1, C, 700, 600>
<T1, commit>
-- Falha --
```

Recuperação

```
redo(T0);
redo(T1);
```

Checkpoint - Ponto de Salvamento

- Utilizado para reduzir a quantidade de transações a serem recuperadas depois de uma falha
- Ações:
  1. não permitir nenhuma atualização durante o checkpoint
  2. escrita em armazenamento estável de todos os registros de log na memória principal
  3. escrita em disco de todos os blocos de buffer modificados
  4. escrita em armazenamento estável de um registro de log `<checkpoint L>`, onde `L` é a lista de transações ativas no momento do checkpoint

Escalonamentos Não Seriais

- o gerenciador de concorrência deve garantir que não haja leitura suja ou rollback em cascata (utilizando por exemplo 2PL rigoroso)

Escalonamentos Não Seriais - Recuperação de falha

1. encontrar o último `<checkpoint L>`
2. iniciar o *undo-list* com `L`
3. **Fase de redo:** para cada registro de log a partir do checkpoint:
   - `<Ti, X, V1, V2>`: escrever V2 em X
   - `<Ti, start>`: incluir `Ti` em *undo-list*
   - `<Ti, commit>`: remover `Ti` de *undo-list*
   - `<Ti, abort>`: remover `Ti` de *undo-list*
4. **Fase de undo:** varrer o log do final para o início e para operações de transações no *undo-list*:
   - `<Ti, X, V1, V2>`:
     1. escrever no log `<Ti, X, V1>`
     2. escrever V1 em X
   - `<Ti, start>`:
     1. escrever no log `<Ti, abort>`
     2. remover `Ti` de *undo-list*
5. a recuperação termina quando o *undo-list* estiver vazio

Exemplo

```
    LOG                          Recuperação
----------------------    ------------------------------------------
<T0, start>               |   FASE DE REDO          ^  FASE DE UNDO
<T0, B, 2000, 2050>       |                         |
<T1, start>               V                         |
<checkpoint, [T0, T1]>       undo-list=[T0,T1]
<T1, C, 700, 600>            escreve 600 em C
<T1, commit>                 undo-list=[T0]
<T2, start>                  undo-list=[T0,T2]
<T2, A, 500, 400>            escreve 400 em A
<T2, A, 400, 200>            escreve 200 em A
<T0, B, 2000>                escreve 2000 em B
<T0, abort>                  undo-list=[T2]
-- Falha --                                        undo-list=[T2]
                                                   <T2, A, 400>
                                                   <T2, A, 500>
                                                   <T2, abort>
                                                   undo-list=[]
```