Recuperação de Falhas

Banco de Dados II

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM

Recuperação de Falhas

- A recuperação de falhas de transações consiste em um processo que garante o retorno do banco de dados ao seu estado consistente mais recente, antes da ocorrência de uma falha.
  - Geralmente, as técnicas utilizam o arquivo de *log*.
- As possíveis falhas são categorizadas em:
  - Falha catastrófica: recarrega a cópia (*backup*) mais recente do banco de dados e reconstrói um estado consistente do mesmo refazendo operações das transações já confirmadas a partir do arquivo de *log*.
  - Falha não catastrófica: reverte as alterações que causaram a inconsistência desfazendo operações de transações não confirmadas ou refazendo operações de transações confirmadas.

Recuperação de Falhas

- As principais técnicas para recuperação de falhas não catastróficas são:
  - atualização postergada (algoritmo NO-UNDO / REDO);
  - atualização imediata (algoritmo UNDO / REDO).

- Uma técnica de atualização postergada apenas pode ser empregada se a atualização física do banco de dados no disco só ocorre quando uma transação é confirmada.

- Uma técnica de atualização imediata permite a atualização física do banco de dados por operações de uma transação antes que a mesma atinja seu ponto de confirmação.

Caching de Blocos de Disco

- O processo de recuperação de falhas está estreitamente ligado a funções do sistema operacional.
  - Normalmente, uma ou mais páginas de disco, que incluam os itens de dados a serem atualizados, são ocultadas (*cached*) nos *buffers* da memória principal e lá são atualizadas, antes de serem gravadas de volta no disco.

Caching de Blocos de Disco

- Geralmente, uma coleção de *buffers* em memória (*caches* do SGBD) é mantida sob o controle do SGBD.
  - Quando o SGBD solicita uma ação em um item, ele primeiro verifica se o mesmo encontra-se em alguma página de disco em *cache*.
  - Caso não se encontre o item em *cache*, o mesmo é localizado em disco e as páginas de disco apropriadas são copiadas em *cache*.
  - Para substituição de páginas de disco em *cache*, estratégias do sistema operacional são usadas: *least recently used* (LRU), *first in first out* (FIFO).

Registro Adiantado em *Log*

- Ao descarregar um *buffer* modificado para o disco, a estratégia **atualização in loco** geralmente é utilizada.
  - Esta estratégia grava o *buffer* de volta para a mesma localização no disco, sobrescrevendo os valores antigos de quaisquer itens de dado alterados no disco.

- Torna-se necessário, então, utilizar um *log* para recuperação.
  - O mecanismo de recuperação deve garantir que a imagem anterior do item de dado seja registrada na entrada apropriada do *log* e que tal entrada seja descarregada no disco, antes que a imagem anterior seja sobrescrita pela imagem posterior no banco de dados do disco.
  - Este processo é conhecido como **registro adiantado em *log*** (***write-ahead logging***): gravação do *log* antes que a atualização seja efetuada no banco de dados.

Registro Adiantado em *Log*

- Algumas informações necessárias no *log* para desfazer e refazer operações são:
  - Para refazer (REDO), uma entrada no *log* para gravação deve incluir o novo valor (imagem posterior) do item gravado.
  - Para desfazer (UNDO), uma entrada no *log* para gravação deve incluir o valor antigo (imagem anterior) do item gravado.
  - Em um algoritmo UNDO / REDO, ambos os tipos de entrada de *log* são combinados.
  - Em técnicas de controle de concorrência que não previnem *rollback* em cascata, entradas de leitura de itens de dado (*read_item*) também devem ser registradas no *log* para técnicas de recuperação que envolvam UNDO.

Registro Adiantado em *Log*

- Os termos *steal*/*no-steal* (roubar/não-roubar) e *force*/*no-force* (forçar/não-forçar) são usados para especificar quando uma página do bd pode ser gravada em disco a partir do *cache*:
  - *Steal*: uma página do *cache* atualizada pode ser gravada em disco antes do *commit* da transação.
  - *No-steal*: uma página do *cache* atualizada não pode ser gravada em disco antes do *commit* da transação.
  - *Force*: todas as páginas atualizadas por uma transação são imediatamente gravadas em disco quando a transação atinge seu ponto de confirmação.
  - *No-force*: as páginas atualizadas por uma transação *não* são necessariamente gravadas em disco quando a transação atinge seu ponto de confirmação.

Registro Adiantado em *Log*

- Os SGBDs geralmente empregam uma estratégia roubada/não forçada no processo de recuperação de falhas.
  - A vantagem de ser roubada (*steal*) é que se evita a necessidade de um espaço muito grande de *buffer* para o armazenamento em memória de todas as páginas atualizadas.
  - A vantagem de não ser forçada (*no-force*) é que uma página atualizada por uma transação confirmada poderá ainda estar no *buffer* quando outra transação necessitar de atualização, eliminando o custo de I/O para ler novamente essa página no disco.

Checkpoint

- Periodicamente, o sistema operacional força a gravação em disco de todos os *buffers* do SGBD que tenham sido alterados.
  - Quando isso ocorre, um registro chamado *checkpoint* (ponto de verificação) é gravado em *log*.
  - O intervalo entre *checkpoints* pode ser medido em tempo (minutos) ou em número de transações confirmadas desde o último *checkpoint*.

- Vantagem: todas as transações que registraram um [*commit*, T] no *log* antes de um [*checkpoint*] não precisam ser refeitas em caso de colapso do sistema.

Checkpoint

- Ações seguidas em um processo de *checkpoint* são:
  1. Suspender temporariamente a execução de transações.
  2. Gravar à força todos os *buffers* de memória que tenham sido modificados para o disco.
  3. Gravar um registro [*checkpoint*] no *log* ,forçando a gravação do mesmo no disco.
  4. Retomar as transações em execução.

Fuzzy checkpoint

- Para não atrasar o processamento de transações em função da ação 1 do processo de *checkpoint*, a técnica *fuzzy checkpoint* permite a retomada do processamento antes da conclusão da ação 2.
  - Para isso, a técnica grava o registro de *checkpoint* mas mantém um apontador para o *checkpoint* anterior (*checkpoint* válido) até que a ação 2 seja concluída.

Atualização Postergada

- A técnica de recuperação baseada na atualização postergada adia qualquer atualização no banco de dados até que a transação complete sua execução com êxito e atinja seu ponto de confirmação (abordagem *no-steal*).
  - Durante a execução da transação, as atualizações são registradas somente no *log* e nos *buffers* do *cache*.
  - Se uma transação falha antes de atingir seu ponto de confirmação, não há necessidade de desfazer nenhuma operação, porque a transação não afetou, de modo algum, o banco de dados armazenado em disco.
- Depois que a transação atinge seu ponto de confirmação e o *log* é forçosamente gravado no disco, as atualizações são registradas no banco de dados (registro adiantado em log).
- Na prática, é inviável devido ao tamanho do *cache* necessário para manipular grandes transações.

Atualização Postergada

- A técnica é conhecida como algoritmo de recuperação NO-UNDO / REDO (NÃO-DESFAZER / REFAZER).
  - Não é necessário desfazer porque o banco de dados não é atualizado antes do *commit*.
  - Pode ser necessário refazer se o sistema falhar depois do *commit*, porém antes que todas as suas alterações tenham sido registradas no banco de dados.
    - As operações da transação são refeitas a partir das entradas do *log*.

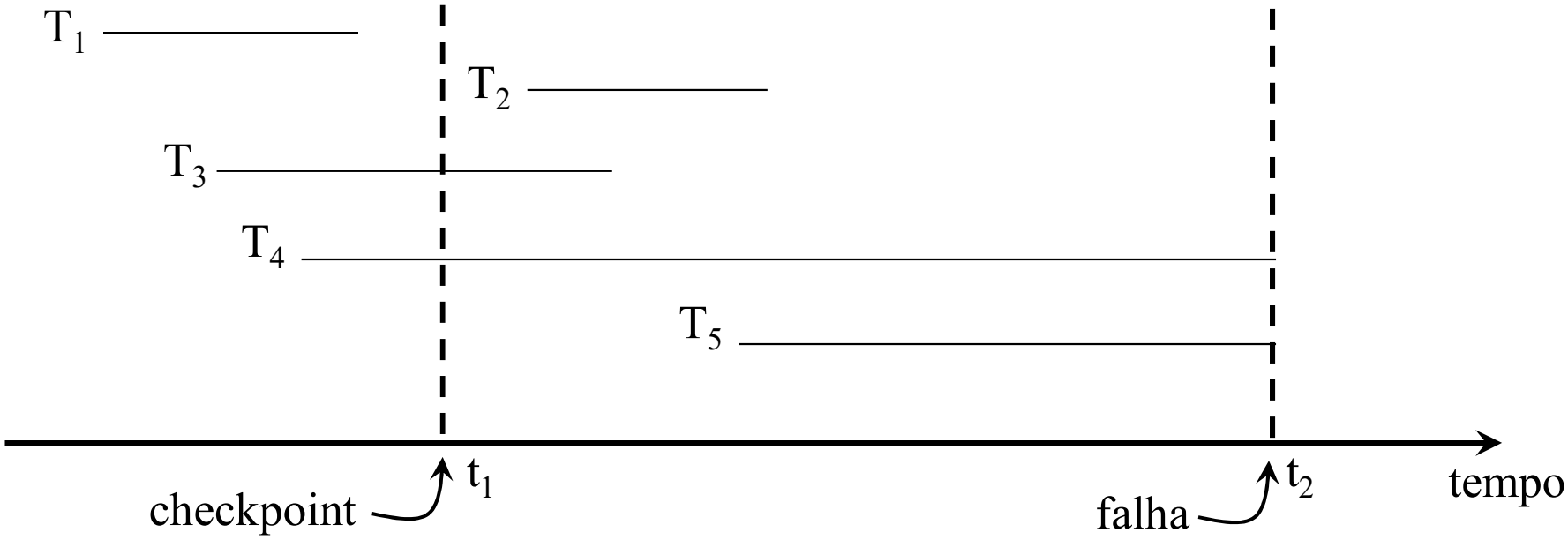
Atualização Postergada

- Procedimento para recuperação no momento de uma falha:
  - Todas as operações de escrita (*write_item*) das transações confirmadas devem ser refeitas, a partir do *log* na ordem em que foram gravadas, utilizando o procedimento REDO.
  - As transações que estavam ativas e não chegaram ao *commit* são efetivamente canceladas e devem ser resubmetidas.

- Procedimento REDO:
  - Examinar as entradas de *log* do tipo [*write_item, T, X, novo_valor*] e ajustar o valor do item X no banco de dados para *novo_valor* (imagem posterior).

Atualização Postergada

- Considere o seguinte exemplo:



Como atua a técnica de recuperação baseada na atualização postergada?

Atualização Postergada

- Considerações do exemplo:
  - Quando o *checkpoint* foi executado no momento t_1, apenas a transação T_1 havia chegado ao *commit*.
  - Antes da falha do sistema no momento t_2, T_2 e T_3 já haviam sido confirmadas; T_4 e T_5 não.

- De acordo com a técnica, tem-se:
  - não há necessidade de refazer as operações de T_1 já que foi confirmada antes do *checkpoint* no tempo t_1;
  - as operações *write_item* de T_2 e T_3 devem ser refeitas porque ambas atingiram seus pontos de confirmação depois do *checkpoint* no tempo t_1 (o *log* já foi gravado em disco);
  - as transações T_4 e T_5 são ignoradas (nenhuma de suas operações foi registrada no banco de dados) e devem ser resubmetidas.

Atualização Postergada

- A operação REDO pode se tornar mais eficiente.
  - Quando um item de dado X for atualizado mais de uma vez por transações confirmadas desde o último *checkpoint*, só é necessário refazer a última atualização de X a partir do *log* (as outras atualizações seriam sobrescritas pelo último REDO).
    - Nesse caso, o processo inicia-se a partir do final do *log*.
    - Sempre que um item for refeito, ele é acrescentado a uma lista de itens refeitos.
    - Antes de refazer a operação aplicada a um item, a lista é verificada: se o item aparecer na lista, não é novamente refeito, uma vez que seu último valor já foi recuperado.

Atualização Imediata

- A técnica de recuperação baseada em atualização imediata faz com que o banco de dados seja ou não "imediatamente" atualizado, quando uma transação emite um comando de atualização, sem ter que esperar que a mesma atinja seu ponto de confirmação.
  - O processo de registro adiado em *log* deve ser usado para permitir recuperação em caso de falha.

Atualização Imediata

- Existem duas categorias de algoritmos, a saber:
  - Algoritmo de recuperação UNDO / NO-REDO
    - Não há necessidade de refazer operações se a técnica de recuperação garantir que todas as atualizações de uma transação sejam registradas no banco de dados em disco antes do *commit* da transação.
  - Algoritmo de recuperação UNDO / REDO
    - Pode ser necessário desfazer e refazer operações se a técnica de recuperação permitir que a transação chegue ao seu ponto de confirmação antes que todas as alterações sejam gravadas no banco de dados.

Atualização Imediata

- O sistema deve manter duas listas de transações: uma lista de transações confirmadas desde o último *checkpoint* (**lista de *commit***) e uma lista de transações ativas (**lista ativa**).

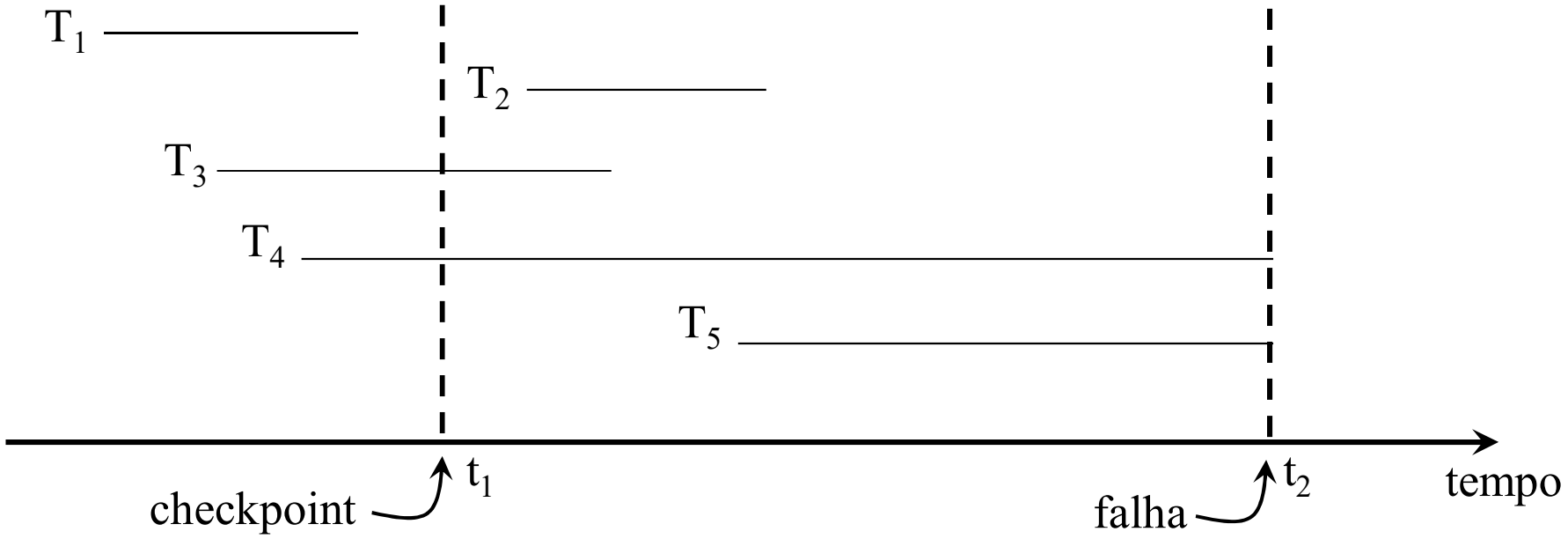
- Procedimento para recuperação no momento de uma falha:
  - Todas as operações de escrita (*write_item*) das transações ativas devem ser desfeitas, a partir do *log* na ordem inversa em que foram gravadas, utilizando o procedimento UNDO.
    - Tais transações são efetivamente canceladas e resubmetidas.
  - Todas as operações de escrita (*write_item*) das transações confirmadas devem ser refeitas, a partir do *log* na ordem em que foram gravadas, utilizando o procedimento REDO.
    - Para melhorar eficiência, pode-se partir do final do *log*.

Atualização Imediata

- Procedimento UNDO:
  - Examinar as entradas de *log* do tipo [*write_item, T, X, valor_antigo, novo_valor*] e ajustar o valor do item X no banco de dados para *valor_antigo* (imagem anterior).
    - O processo de desfazer as operações *write_item* de uma ou mais transações do *log* deve acontecer na ordem inversa da ordem em que as operações foram gravadas no *log*.
- Procedimento REDO:
  - Examinar as entradas de *log* do tipo [*write_item, T, X, valor_antigo, novo_valor*] e ajustar o valor do item X no banco de dados para *novo_valor* (imagem posterior).

Atualização Imediata

- Considere o seguinte exemplo:



Como atua a técnica de recuperação baseada na atualização imediata?

Atualização Imediata

- Considerações do exemplo:
  - Quando o *checkpoint* foi executado no momento t_1, apenas a transação T_1 havia chegado ao *commit*.
  - Antes da falha do sistema no momento t_2, T_2 e T_3 já haviam sido confirmadas; T_4 e T_5 não.

- De acordo com a técnica, tem-se:
  - não há necessidade de refazer as operações de T_1 já que foi confirmada antes do *checkpoint* no tempo t_1;
  - as operações *write_item* de T_2 e T_3 devem ser refeitas porque ambas atingiram seus pontos de confirmação depois do *checkpoint* no tempo t_1 (o *log* já foi gravado em disco);
  - as operações *write_item* de T_4 e T_5 devem ser desfeitas e resubmetidas porque ambas não atingiram seus pontos de confirmação antes da falha.

Recuperação em Sistemas de Vários BDs

- Em um sistema composto por vários bancos de dados (homogêneos ou heterogêneos), uma única transação pode acessar dados em cada um deles.
  - Essas transações são chamadas de transações distribuídas.

- Para manter a atomicidade de uma transação distribuída, é necessário ter um mecanismo de recuperação de falhas em dois níveis: global (gerenciador global) e local (gerenciador de cada banco de dados envolvido).
  - Os gerenciadores global e local seguem, normalmente, o protocolo de confirmação em duas fases (*two-phase commit*) que envolve as fases de preparação e de confirmação propriamente dita.

Recuperação em Sistemas de Vários BDs

- Fase de preparação do protocolo *two-phase commit*:
  - Quando todos os bancos de dados sinalizam ao gerenciador global que a parte da transação distribuída de cada um foi confirmada, então o gerenciador global envia um comando de preparação para confirmação para todos os gerenciadores locais envolvidos na transação.
  - Cada gerenciador local, então, faz o que for necessário para recuperação local (registro de *log*).
    - Quando cada gerenciador local completa a fase de preparação, ele retorna sucesso ou falha de preparação para o gerenciador global.
    - Se o gerenciador global não receber uma resposta de algum gerenciador local dentro de um certo intervalo de tempo (*timeout*), ele supõe uma resposta de falha.

Recuperação em Sistemas de Vários BDs

- Fase de confirmação do protocolo *two-phase commit*:
  - Se o gerenciador global receber preparação com sucesso de todos os gerenciadores locais, ele envia comandos *commit* para cada um dos gerenciadores locais.
    - Os gerenciadores locais podem completar a confirmação.
    - O gerenciador global pode notificar sucesso à aplicação.
  - Se algum dos gerenciadores locais relatar uma falha na preparação, o gerenciador global envia comandos *rollback* para cada um dos gerenciadores locais e indica uma falha para a aplicação.

- Essência do protocolo: todos os bancos de dados concluem o efeito da transação ou nenhum deles conclui.