CEFET/RJ

Sistema de Recuperação

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

FUNDAMENTOS

Classificação de Falhas

Sistemas de banco de dados enfrentam diversos tipos de falhas que podem comprometer a integridade dos dados. Compreender essas classificações é fundamental para implementar mecanismos de recuperação eficazes.

Falha de Transação

Erros lógicos: A transação não pode ser concluída devido a alguma condição de erro interno, como violação de restrições ou divisão por zero.

Erros de sistema: O sistema de banco de dados precisa encerrar uma transação ativa devido a condições como deadlock ou timeout de recursos.

Falha do Sistema

Uma queda de energia ou outra falha de hardware ou software causa a parada do sistema (crash). Opera sob a **suposição de parada segura (fail-stop)**: assume-se que o conteúdo do armazenamento não-volátil não é corrompido pela falha.

Os sistemas de banco de dados possuem numerosas verificações de integridade para prevenir corrupção dos dados em disco.

Falha de Disco

Um crash de cabeça de leitura ou falha similar do disco destrói todo ou parte do armazenamento em disco. A destruição é considerada detectável através de checksums utilizados pelos drives de disco.

CONCEITOS ESSENCIAIS

Algoritmos de Recuperação

Considere uma transação *Ti* que transfere $50 da conta *A* para a conta *B*. Esta operação requer duas atualizações: subtrair 50 de A e adicionar 50 a B. O desafio surge quando uma falha ocorre após uma modificação ser feita, mas antes que ambas sejam completadas.

Dilema da Modificação

Modificar o banco de dados sem garantir que a transação será confirmada pode deixar o banco em estado inconsistente. Por outro lado, não modificar pode resultar em perda de atualizações se a falha ocorrer logo após o commit.

Estrutura dos Algoritmos

Os algoritmos de recuperação possuem duas partes fundamentais que trabalham em conjunto para garantir a integridade dos dados.

1

Durante Processamento Normal

Ações tomadas durante o processamento normal das transações para garantir que informação suficiente existe para recuperação.

2

Após Falha

Ações tomadas após uma falha para recuperar o conteúdo do banco de dados a um estado que garanta atomicidade, consistência e durabilidade.

ARMAZENAMENTO

Estrutura de Armazenamento

Os sistemas de banco de dados utilizam diferentes tipos de armazenamento, cada um com características específicas de persistência e vulnerabilidade a falhas. Compreender essas distinções é crucial para o design de sistemas robustos.

Armazenamento Volátil

Não sobrevive a crashes do sistema

- Memória principal (RAM)
- Memória cache
- Registradores de CPU

Dados perdidos em caso de falha de energia ou crash.

Armazenamento Não-Volátil

Sobrevive a crashes do sistema

- Discos magnéticos (HDD)
- Unidades de estado sólido (SSD)
- Fita magnética
- Memória flash
- RAM não-volátil

Ainda pode falhar, perdendo dados em algumas situações.

Armazenamento Estável

Forma mítica que sobrevive a todas as falhas

Aproximado mantendo múltiplas cópias em mídias não-voláteis distintas. Representa o ideal teórico de persistência absoluta.

Consulte o livro-texto para detalhes sobre implementação de armazenamento estável.

Implementação de Armazenamento Estável

A implementação prática de armazenamento estável envolve estratégias sofisticadas de replicação e verificação. O objetivo é aproximar-se do ideal teórico de armazenamento que sobrevive a todas as falhas.

Múltiplas Cópias

Manter múltiplas cópias de cada bloco em discos separados. As cópias podem estar em sites remotos para proteção contra desastres como incêndio ou inundação.

Falha Parcial

O bloco de destino pode conter informação incorreta se a transferência for interrompida.

Falha Total

O bloco de destino nunca foi atualizado se a operação falhar completamente.

Protocolo de Escrita Segura

Para proteger a mídia de armazenamento durante a transferência de dados:

1. Escrever a informação no primeiro bloco físico
2. Quando a primeira escrita for concluída com sucesso, escrever a mesma informação no segundo bloco físico
3. A saída é considerada completa apenas após a segunda escrita ser concluída com sucesso

GERENCIAMENTO DE MEMÓRIA

Acesso a Dados

O gerenciamento eficiente de dados entre diferentes níveis de armazenamento é fundamental para o desempenho e a confiabilidade do sistema de banco de dados. Compreender a distinção entre blocos físicos e blocos de buffer é essencial.

Blocos Físicos

São aqueles blocos que residem permanentemente no disco. Representam o armazenamento persistente de longo prazo dos dados do banco de dados.

Blocos de Buffer

São os blocos que residem temporariamente na memória principal. Fornecem acesso rápido aos dados durante o processamento de transações.

Operações de Movimentação

As movimentações de blocos entre o disco e a memória principal são iniciadas através de duas operações fundamentais:

`input(B)`

Transfere o bloco físico *B* para a memória principal, tornando-o disponível para leitura e modificação.

`output(B)`

Transfere o bloco de buffer *B* para o disco, substituindo o bloco físico apropriado. Esta operação torna as modificações persistentes.

Simplificação assumida: Cada item de dados cabe em um único bloco e é armazenado dentro dele. Esta suposição simplifica o gerenciamento de dados e as operações de recuperação.

Recuperação e Atomicidade

Para garantir atomicidade apesar das falhas, primeiro enviamos informações descrevendo as modificações para armazenamento estável sem modificar o próprio banco de dados. Esta abordagem fundamental separa o registro das intenções da execução real das mudanças.

Mecanismos Baseados em Log

Estudamos **mecanismos de recuperação baseados em log** em detalhes. Esta é a abordagem mais amplamente utilizada na indústria devido à sua eficiência e robustez.

Conceitos-Chave

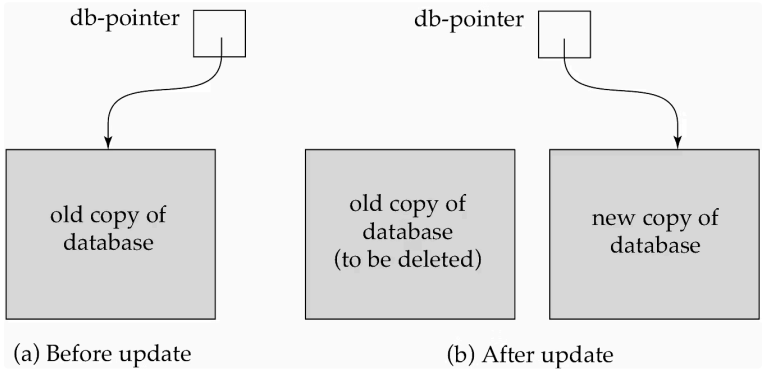
Primeiro apresentamos os conceitos fundamentais que sustentam a recuperação baseada em log.

Algoritmo de Recuperação

Então apresentamos o algoritmo real de recuperação com todos os detalhes de implementação.

Alternativas

Shadow-copy e Shadow-paging: Alternativa menos utilizada para recuperação.



Exemplo de shadow-copy

LOGS

Recuperação Baseada em Log

Um **log** é uma sequência de **registros de log** que mantém informações sobre atividades de atualização no banco de dados. O log é mantido em armazenamento estável para garantir sua persistência mesmo em caso de falhas.

1

<Ti start>

Quando a transação *Ti* inicia, ela se registra escrevendo um registro de log de início.

2

<Ti, X, V1, V2>

Antes de *Ti* executar `write(X)`, um registro de log é escrito, onde *V1* é o **valor antigo** de *X* antes da escrita, e *V2* é o **valor novo** a ser escrito em *X*.

3

<Ti commit>

Quando *Ti* termina sua última instrução, o registro de log de commit é escrito, marcando o sucesso da transação.

Duas Abordagens Utilizando Logs

Modificação Imediata do Banco de Dados

Permite atualizações antes do commit da transação.

Modificação Diferida do Banco de Dados

Adia atualizações até o commit da transação.

Modificação Imediata do Banco de Dados

O esquema de **modificação imediata** permite que atualizações de uma transação não confirmada sejam feitas no buffer, ou no próprio disco, antes que a transação seja confirmada. Esta abordagem oferece flexibilidade, mas requer cuidados especiais para garantir a recuperabilidade.

Regra de Escrita Antecipada do Log (WAL)

O registro de log de atualização **deve ser escrito antes** que o item do banco de dados seja escrito. Assumimos que o registro de log é enviado diretamente para armazenamento estável.

Nota: Veremos posteriormente como adiar a saída de registros de log até certo ponto para melhorar o desempenho.

Flexibilidade de Saída

A saída de blocos atualizados para o disco pode ocorrer a qualquer momento antes ou após o commit da transação. A ordem em que os blocos são enviados pode ser diferente da ordem em que foram escritos.

Esquema de Modificação Diferida

O esquema de **modificação diferida** realiza atualizações no buffer/disco somente no momento do commit da transação.

- **Vantagem:** Simplifica alguns aspectos da recuperação, pois transações não confirmadas não afetam o banco de dados
- **Desvantagem:** Tem o overhead de armazenar cópia local das modificações até o commit

CONFIRMAÇÃO

Commit de Transação

Uma transação é considerada confirmada (committed) quando seu registro de log de commit é enviado para armazenamento estável. Este é o ponto crítico que determina se as modificações da transação se tornam permanentes e visíveis para outras transações.

1

Pré-requisito para Commit

Todos os registros de log anteriores da transação **devem ter sido enviados** para armazenamento estável antes que o registro de commit possa ser escrito. Esta ordem garante a recuperabilidade completa.

2

Escritas Pendentes no Buffer

As escritas realizadas por uma transação podem ainda estar no buffer quando a transação confirma, e podem ser enviadas para o disco posteriormente. O importante é que o log garanta que essas escritas possam ser refeitas se necessário.

Ponto-chave: A durabilidade não depende de todas as modificações estarem fisicamente no disco no momento do commit, mas sim de todos os registros de log estarem em armazenamento estável. Isso permite otimizações significativas de desempenho.

Controle de Concorrência e Recuperação

Com transações concorrentes, todas as transações compartilham um único buffer de disco e um único log. Um bloco de buffer pode ter itens de dados atualizados por uma ou mais transações, criando desafios únicos para o sistema de recuperação.

Suposição Fundamental

Assumimos que se uma transação Ti modificou um item, nenhuma outra transação pode modificar o mesmo item até que Ti tenha confirmado ou abortado.

Em outras palavras, as atualizações de transações não confirmadas não devem ser visíveis para outras transações.

1

Por Que Esta Restrição?

Problema do cascateamento: Como realizar undo se *T1* atualiza A, depois *T2* atualiza A e confirma, e finalmente *T1* precisa abortar? Seria necessário desfazer T2, causando um efeito cascata indesejável.

2

Implementação Prática

Pode ser garantido obtendo **locks exclusivos** em itens atualizados e mantendo os locks até o final da transação. Esta é a abordagem de **bloqueio estrito em duas fases (strict two-phase locking)**.

3

Intercalação no Log

Os registros de log de diferentes transações podem ser intercalados no log, refletindo a execução concorrente. O sistema de recuperação deve ser capaz de processar essa intercalação corretamente.

OPERAÇÕES

Operações de Undo e Redo

As operações de undo e redo são fundamentais para restaurar o banco de dados a um estado consistente após falhas. Cada uma serve um propósito específico no processo de recuperação.

`undo(Ti)` - Desfazer Transação

Restaura o valor de todos os itens de dados atualizados por *Ti* para seus **valores antigos**, percorrendo o log de trás para frente a partir do último registro de log para *Ti*.

- Cada vez que um item de dados X é restaurado para seu valor antigo V, um registro de log especial `<Ti, X, V>` é escrito
- Quando o undo de uma transação é completo, um registro de log `<Ti abort>` é escrito

Usado para: Transações que não confirmaram antes de uma falha.

`redo(Ti)` - Refazer Transação

Define o valor de todos os itens de dados atualizados por *Ti* para os **valores novos**, percorrendo o log para frente a partir do primeiro registro de log para *Ti*.

- Nenhum registro de log é feito neste caso
- As operações são idempotentes (podem ser repetidas com segurança)

Usado para: Transações que confirmaram mas cujas atualizações podem não ter sido escritas no disco.

Propriedade importante: Ambas as operações são idempotentes, o que significa que podem ser executadas múltiplas vezes com o mesmo resultado. Isso é crucial para a robustez do sistema de recuperação.

OTIMIZAÇÃO

Checkpoints

Refazer ou desfazer todas as transações registradas no log pode ser muito lento, especialmente em sistemas que executam por longos períodos. O checkpointing otimiza significativamente o processo de recuperação.

Problemas da Recuperação sem Checkpoints

Processamento Demorado

Processar o log inteiro consome muito tempo se o sistema executou por um longo período, potencialmente horas ou dias de registros.

Redo Desnecessário

Poderíamos desnecessariamente refazer transações que já enviaram suas atualizações para o banco de dados no disco.

Procedimento de Checkpointing

Otimizar o processo de recuperação realizando **checkpointing** periodicamente:

1

Enviar Registros de Log

Enviar todos os registros de log atualmente residindo na memória principal para armazenamento estável.

2

Enviar Blocos Modificados

Enviar todos os blocos de buffer modificados para o disco, garantindo que todas as modificações pendentes sejam persistidas.

3

Escrever Registro de Checkpoint

Escrever um registro de log `<checkpoint L>` para armazenamento estável, onde *L* é uma lista de todas as transações ativas no momento do checkpoint.

4

Pausar Atualizações

Todas as atualizações são interrompidas durante o checkpointing para garantir consistência. Isso cria um ponto conhecido e seguro no log.

Benefício principal: Durante a recuperação, apenas transações que estavam ativas no último checkpoint (ou que iniciaram depois) precisam ser consideradas, reduzindo drasticamente o tempo de recuperação.

Referências

Elmasri & Navathe

Fundamentals of Database Systems

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

Korth, Sudarshan & Silberschatz

Database System Concepts

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

Özsu & Valduriez

Principles of Distributed Database Systems

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.

Nota: Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.