Controle Distribuído da Concorrência

Aluno: Walter Travassos Sarinho
wts@cin.ufpe.br
Orientadora: Bernadette Farias Lóscio
bfl@cin.ufpe.br
Centro de Informática (CIn)
Pós-Graduação em Ciência da Computação
Universidade Federal de Pernambuco (UFPE)

Controle Distribuído da Concorrência

- REVISÃO E INTRODUÇÃO
- SERIALIZABILIDADE
- ALGORITMOS LOCKING-BASED
- ALGORITMOS TIMESTAMP-BASED
- GERENCIAMENTO DE IMPASSES
- CONSIDERAÇÕES

Revisão de Conceitos

Revisão e Introdução

Revisão de Conceitos

Transação
Operação

Uma transação Ti é uma ordenação parcial sobre suas operações e a condição de término.

Denota-se $O_{ij}(x)$ alguma operação O_j da transação T_i sobre uma entidade do banco de dados x.

Revisão de Conceitos

Operação em conflito.

Duas operações $O_i(x)$ e $O_j(x)$ são conflitantes se
(1) pertencem a transações diferentes,
(2) ambas acessam o mesmo item de dados e
(3) pelo menos uma delas é uma operação Write_item.

Revisão de Conceitos

ACID

Atomicidade – "tudo ou nada".
Consistência – mapeia um estado consistente do banco de dados em outro.
Isolamento – acesso ao banco de forma isolada.
Durabilidade – resultados permanentes no banco de dados.

Revisão de Conceitos

Transações Planas

Possui um único ponto de início (Begin_transaction) e um único ponto de término (End_transaction).

Revisão de Conceitos

Transações Aninhadas

Uma transação inclui outras transações entre seus pontos de início e consolidação. Transações embutidas em outras costumam ser chamadas de subtransações.

Introdução

- Controle de concorrência em SGBD distribuído assegura a consistência em ambiente distribuído e multiusuário.
- Objetiva encontrar um equilíbrio adequado entre a **consistência do BD** e um **nível elevado de concorrência**.
- Para toda essa apresentação, considerar que o sistema distribuído é totalmente confiável e não experimenta nenhuma falha.

Serializabilidade

Serializabilidade

A serializabilização de escalonamentos é usado para identificar quais escalonamentos estão corretos quando as execuções da transação tiverem intercalação de suas operações nos escalonamentos.

Serializabilidade

- Se em um escalonamento S as operações não estão intercaladas (ou seja, as operações de cada transação ocorrem consecutivamente) dizemos que o escalonamento é serial.
- A execução serial de um conjunto de transações mantém a consistência do banco de dados, porém pode acarretar estados de inatividade da CPU, desperdiçando processamento.

Serializabilidade

A execução concorrente de transações deve deixar o banco de dados num estado que possa ser alcançado por uma execução sequencial em alguma ordem.

Caso essa situação seja alcançado serão resolvidos problemas como os de **atualizações perdidas** (assegurando o isolamento e não permitindo que os resultados incompletos sejam acessados por outras transações)

Serializabilidade

- Um escalonamento S (ou Schedule, também chamado de Histórico) é definidos sobre um conjunto de transações T = {T_1, T_2,..., T_n} e especifica uma ordem intercalada de execução dessas operações de transações.
- Um escalonamento pode ser especificado como uma ordem parcial sobre T.

Serializabilidade

Formalizando um escalonamento completo

1. O domínio da relação será a união dos domínios individuais.
2. A relação de ordenação é um superconjunto das relações de ordenação de transações individuais
3. Manter a ordem de execução entre operações conflitantes.

Serializabilidade

Exemplo de escalonamento completo

T_1: Read(x) $x \leftarrow x+1$ Write(x) Commit

T_2: Read(x) $x \leftarrow x+1$ Write(x) Commit

$$\Sigma_1 = \{R_1(x), W_1(x), C_1\}$$
$$\Sigma_2 = \{R_2(x), W_2(x), C_2\}$$

Thus

$$\Sigma_T = \Sigma_1 \cup \Sigma_2 = \{R_1(x), W_1(x), C_1, R_2(x), W_2(x), C_2\}$$

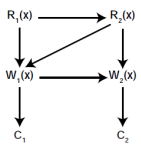
and

$$\prec_H = \{(R_1,R_2),(R_1,W_1),(R_1,C_1),(R_1,W_2),(R_1,C_2),(R_2,W_1),(R_2,C_1),(R_2,W_2),(R_2,C_2),(W_1,C_1),(W_1,W_2),(W_1,C_2),(C_1,W_2),(C_1,C_2),(W_2,C_2)\}$$

Serializabilidade

T_1: Read(x) $x \leftarrow x+1$ Write(x) Commit

T_2: Read(x) $x \leftarrow x+1$ Write(x) Commit



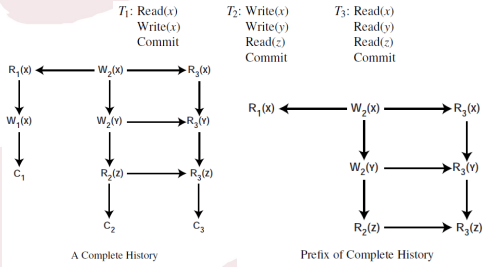
$$H_T^c = \{R_1(x), R_2(x), W_1(x), C_1, W_2(x), C_2\}$$

Serializabilidade

- Um escalonamento também pode ser definido como um prefixo de um escalonamento completo.
- Utilizar esse prefixo permite lidar com escalonamentos incompletos.
- A serializabilidade lida **apenas** com operações de transações que entram em conflito, e não com todas as operações.
- Ao introduzir falhas, deve-se ser capaz de lidar com escalonamentos incompletos, e é isso que o prefixo permite fazer.

Serializabilidade

Escalonamento incompleto



Serializabilidade

Considerando as três transações do exemplo anterior, temos o escalonamento a seguir:

$$H = \{\underbrace{W_2(x), W_2(y), R_2(z)}_{T_2}, \underbrace{R_1(x), W_1(x)}_{T_1}, \underbrace{R_3(x), R_3(y), R_3(z)}_{T_3}\}$$

É serial pois todas as operações de T2 são executadas antes das de T1, e todas de T1 são executadas antes de T3.

$$T_2 \rightarrow T_1 \rightarrow T_3 \qquad T_2 \prec_H T_1 \prec_H T_3.$$

Serializabilidade

Equivalência de Conflitos

Dois escalonamentos definidos em cima de um mesmo conjuntos de transações T são ditos equivalentes se para cada par de operações conflitantes, uma operação O_{ij} que será executada numa T_1 é a mesma que será executada numa T_2 que ocorre após T_1. Essa situação é o que chamamos de equivalência de conflito.

Serializabilidade

Formalizando a serializabilidade

Um escalonamento S é serializável se, e somente se, ele é equivalente de conflitos a um escalonamento serial.

Serializabilidade

Qual a diferença entre o escalonamento serial e o serializável?

Serializabilidade

A serializabilidade se estende de maneira direta aos bancos de dados distribuídos não replicados (ou particionados).

Escalonamento em cada site -> escalonamento local

Se o BD não for replicado e cada escalonamento local for serializável, sua união, chamada escalonamento global, também será serializável.

Serializabilidade

T1: Read(x)
x<- x + 5
Write(x)
Commit

T2: Read(x)
x <- x * 10
Write(x)
Commit

S1 = {$R_1(x)$, $W_1(x)$, C_1, $R_2(x)$, $W_2(x)$,C_2 } **60**
S2 = {$R_2(x)$, $W_2(x)$, C_2, $R_1(x)$, $W_1(x)$, C_1 } **15**

Suponha que antes das transações o valor de (x) seja 1, qual será o valor final nos dois sites?

Serializabilidade

- O exemplo anterior viola a consistência mútua dos dois bancos de dados locais.
- A consistência mútua exige que todos os valores de todos os itens de dados replicados sejam idênticos.

Para resolver esse problema:

(1) Cada escalonamento local deve ser serializável.

(2) Duas operações conflitantes devem estar na mesma ordem relativa em todos os escalonamentos locais que aparecem juntas.

Taxonomia dos Mecanismos de Concorrência

Existem diversos modos de abordagens de controle de concorrência.

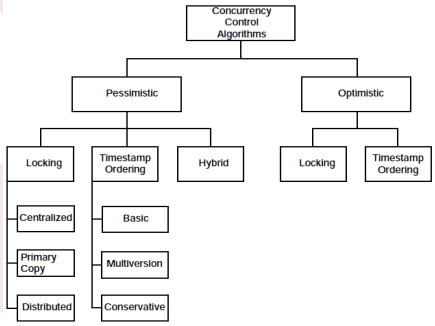
- Distribuição do banco de dados – total ou parcialmente replicado
- Topologia da Rede – estrela, circular, com capacidade de difusão (broadcasting)
- Primitiva de sincronização – timestamp ordering ou locking-based

Taxonomia dos Mecanismos de Concorrência

As primitivas de sincronização podem ser usadas em algoritmos com dois pontos de vista

- Pessimistas – muitas transações entrarão em conflito, portanto a sincronização da execução concorrente ocorre mais cedo em seu ciclo de execução.
- Otimistas – poucas transações entrarão em conflito, portanto a sincronização de execução concorrente ocorre até seu término.

Taxonomia dos Mecanismos de Concorrência



Algoritmos de Controle de Concorrência

Locking-Based

Locking-Based

- Controle de concorrência baseado em bloqueio assegura que os dados compartilhados por operações conflitantes sejam acessados por uma única operação de cada vez.
- Isso é conseguido pelo associação de um "bloqueio" a cada unidade de bloqueio.
- Esse bloqueio é definido por uma transação antes de ser acessado e é redefinido no final de seu uso.

Locking-Based

- Existem dois modos de bloqueio:
- Bloqueio de leitura (rl – read lock)
- Bloqueio de gravação (wr – write lock)

	$rl_i(x)$	$wl_i(x)$
$rl_j(x)$	compatible	not compatible
$wl_j(x)$	not compatible	not compatible

Locking-Based

Comunicação

- Em sistemas baseados em bloqueio o escalonador é um **gerenciador de bloqueio** (LM – lock manager).
- O **gerenciador de transações** repassa ao **gerenciador de bloqueio** a operação do banco de dados (leitura ou gravação) e as informações associadas (item acessado, identificador da transação).

Locking-Based

- O **gerenciador de bloqueio** verifica se a unidade de bloqueio que contém o item de dados já está bloqueada. Se já estiver, e se o modo de bloqueio existente for incompatível com o da transação atual, a operação será adiada. Caso contrário, o bloqueio será estabelecido no modo desejado e a operação do BD será repassada ao processador de dados para acesso ao BD real.

Locking-Based

Bloqueio de 2 Fases – 2PL (*two-phase locking*)

- Considere a seguinte transação e seu escalonamento H.

T_1: Read(x)	T_2: Read(x)
$x \leftarrow x+1$	$x \leftarrow x*2$
Write(x)	Write(x)
Read(y)	Read(y)
$y \leftarrow y-1$	$y \leftarrow y*2$
Write(y)	Write(y)
Commit	Commit

$$H = \{wl_1(x), R_1(x), W_1(x), lr_1(x), wl_2(x), R_2(x), w_2(x), lr_2(x), wl_2(y), R_2(y), W_2(y), lr_2(y), wl_1(y), R_1(y), W_1(y), lr_1(y)\}$$

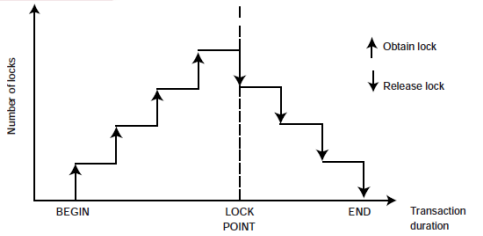
Locking-Based

- A regra de bloqueio de duas fases estabelece que nenhuma transação deve solicitar um bloqueio após liberar um de seus bloqueios.
- Como alternativa, uma transação não deve liberar um bloqueio até ter certeza de que não solicitará outro bloqueio.

Locking-Based

- Algoritmos 2PL executam transações em duas fases.
- Cada transação tem uma fase de crescimento na qual ela obtém bloqueios e acessa itens de dados e,
- Uma fase de contração durante a qual ela libera bloqueios.
- O ponto de bloqueio é o momento em que a transação conseguiu todos os seus bloqueios, mas ainda não começou a liberar nenhum deles.
- Teorema: qualquer escalonamento gerado por um algoritmo que obedece a regra 2PL é serializável.

Locking-Based

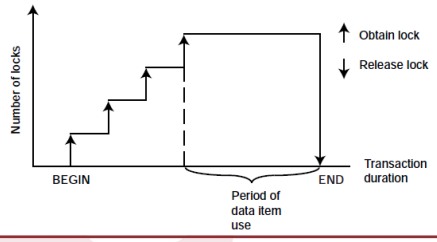


Locking-Based

- É difícil implementar a liberação de bloqueios em cascata pois o gerenciador de bloqueio tem que saber que a transação obteve todos os seus bloqueios e não precisará bloquear outro item de dados.
- O gerenciador precisa também saber que a transação não precisa mais de acesso ao item de dados em questão.
- Se a transação abortar após liberar um bloqueio, pode fazer com que outras transações sejam abortadas que também tenham acessado o item de dados desbloqueado.

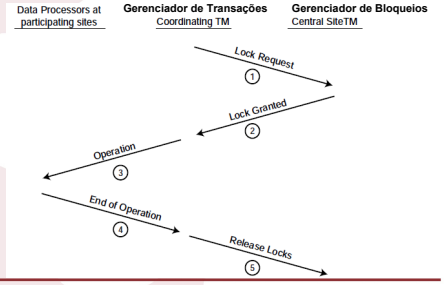
Locking-Based

A maioria dos escalonadores 2PL implementam o bloqueio de 2 fases estrito (*strict two-phase locking*).



Locking-Based

2PL Centralizado



Locking-Based

2PL de Cópia Primária

- É uma extensão direta do 2PL Centralizado
- Implementa gerenciadores de bloqueio em vários sites e cada um irá administrar um dado conjunto de unidades de bloqueio.
- Mudanças mínimas em relação ao C2PL.

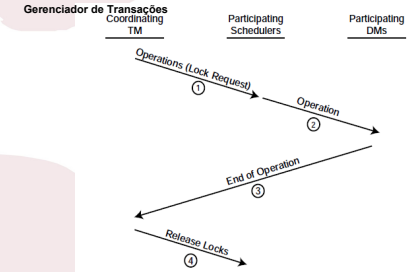
Locking-Based

2PL Distribuído

- Espera a disponibilidade de gerenciadores de bloqueio em cada site.
- Se o BD não for replicado, o 2PL distribuído irá degenerar no algoritmo de 2PL de cópia primária.
- Caso sejam replicados, será implementado o protocolo ROWA

Locking-Based

2PL Distribuído



Algoritmos de Controle de Concorrência

Timestamp Ordering - TO

Timestamp Ordering

- Algoritmos de controle de concorrência do tipo *timestamp* selecionam uma ordem de serialização e executam as transações de acordo com ela.
- Para estabelecer essa ordem, o gerenciador de transações atribui a cada transação um *timestamp* (timbre de hora).
- Gerado pelo Gerenciador de Transações, o *timestamp* é um identificador simples que permite a unicidade, exclusividade e o caráter monotônico de cada operação.

Timestamp Ordering

- Cada nova operação é comparada com operações conflitantes que já tenham sido escalonadas.
- Se a nova operação for mais nova que as operações conflitantes já escalonada, será aceita,
- Do contrário será rejeitada obrigando a transação reiniciar com um novo *timestamp*.
- Um escalonador de TO tem a garantia de gerar escalonamentos serializáveis.
- Além do contador local, o tempo também pode ser usado para definir o *timestamp*.

Timestamp Ordering

Algoritmo Básico de TO

- É uma implementação direta da regra de TO.
- O gerenciador de transações de coordenação atribui o *timestamp* a cada transação, determina os sites em que cada item de dados está armazenado e envia as operações relevantes a esses sites.

Timestamp Ordering

Como as transações nunca esperam enquanto mantêm direitos de acesso aos itens de dados, o algoritmo básico de TO nunca provoca impasses. No entanto, o preço para se livrar de impasses é a reinicialização potencial de uma transação várias vezes.

Timestamp Ordering

Algoritmo de TO conservador (*Conservative*)

- Problema das reinicializações em sites comparativamente inativos em relação a outros.
- Ao invés de usar um contador central (dispendioso), cada gerenciador de transações pode enviar suas operações remotas aos gerenciadores de transações de outros sites.
- Assim, qualquer gerenciador que possuir um contador inferior ao divulgado, ajusta seu próprio a um valor de uma unidade maior que o valor de entrada.

Timestamp Ordering

- O algoritmo básico de TO tenta executar uma operação logo que ela é aceita (agressivo / progressivo).
- Algoritmos conservadores atrasam cada operação até ter certeza que não chegará nenhuma operação com um *timestamp* menor.
- As operações de cada transação são inseridas em buffers até ser possível estabelecer uma relação, tal que, não seja detectada possíveis rejeições, e elas possam ser executadas nessa ordem.

Timestamp Ordering

Algoritmo de TO de várias versões (*Multiversion*)

- Teve foco maior em bancos de dados centralizados.
- As atualizações não modificam o banco de dados. Cada operação de gravação cria uma nova versão do item de dados.
- Cada versão é marcada pelo *timestamp* da transação que o cria.
- Troca espaço de armazenamento pelo tempo.
- Para economizar espaço, as versões podem ser purgadas de tempos em tempos.

Gerenciamento de Impasses

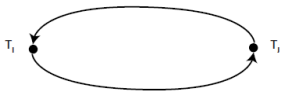
Deadlock Management

Gerenciamento de Impasses

- Um impasse pode ocorrer porque as transações esperam uma pela outra. De modo informal, uma situação de impasse é um conjunto de solicitações que jamais poderão ser concedidas pelo mecanismo de controle de concorrência.
- Um impasse é um fenômeno permanente, não termina a menos que ocorra uma intervenção externa.

Gerenciamento de Impasses

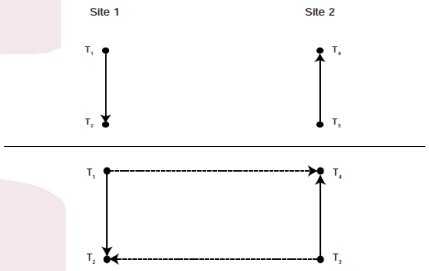
- Wait-for Graph – representa a espera entre as transações.
- Cada nó representa uma transação concorrente.
- O arco representa a espera entre da liberação do bloqueio sobre alguma entidade.



Gerenciamento de Impasses

- A formação do WFG é mais complicada em sistemas distribuídos, pois duas transações que participam de uma condição de impasse podem estar em execução em sites diferentes (impasse global).
- LWFG – grafo de espera local em cada site.
- GWFG – grafo de espera global, união de todos LWFGs.

Gerenciamento de Impasses



Gerenciamento de Impasses

Detecção e Resolução de Impasses:

- Em geral a resolução é feita pela seleção de uma ou mais transações vitimas que serão apropriadas antecipadamente e abortadas para romper os ciclos no GWFG.
- No entanto alguns fatores devem ser levados em consideração...

Gerenciamento de Impasses

1. A quantidade de esforço já investido na transação
2. O custo de se abortar a transação
3. A quantidade de esforço necessária para concluir
4. O número de ciclos que contém a transação (melhor abortar as que possuem mais de um ciclo).

Existem três métodos para detectar impasses distribuídos: centralizada, distribuída e hierárquica.

Gerenciamento de Impasses

Detecção Centralizada de Impasses

- Um site é designado como o detector de impasses para todo o sistema. Ele recebe todos os LWFG do sistema e monta o GWFG.

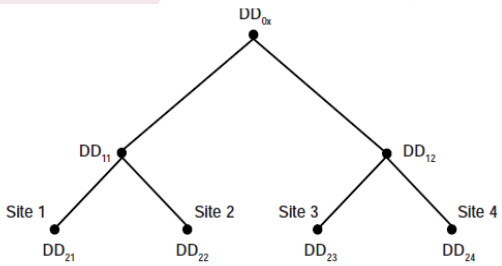
Qual consequência o **tempo de intervalo** entre os recebimentos de LWFGs pelo site responsável refletem no sistema?

Gerenciamento de Impasses

Detecção Hierárquica de Impasses

- É construída uma hierarquia de detectores de impasses.
- Impasses locais são detectados por esse site com o uso do LWFG.
- Cada site envia seu LWFG ao detector de impasses do próximo nível.
- Portanto, quando ocorrem impasses entre sites diferentes, eles são descobertos pelo detector de nível acima .

Gerenciamento de Impasses



Gerenciamento de Impasses

Detecção Hierárquica de Impasses

Consequências: Reduz a dependência do site central no entanto sua implementação é consideravelmente mais complicada.

Gerenciamento de Impasses

Detecção Distribuída de Impasses

- Delegam a responsabilidade de detectar impasses a sites individuais
- Detectores de impasses comunicam seu LWFGs entre si (apenas os ciclos de impasses potenciais são transmitidos).

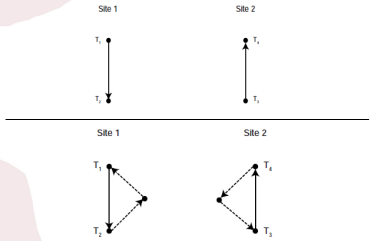
Gerenciamento de Impasses

Detecção Distribuída de Impasses

O LWFG é formado e modificado da seguinte maneira:

1. Tendo em vista que cada site recebe os ciclos de impasses potenciais de outros sites, essas arestas são acrescentadas aos LWFGs.
2. Arestas no LWFG que mostram transações locais esperando por transações de outros sites são unidas com as arestas dos LWFGs que mostram que as transações remotas estão esperando pelas locais.

Gerenciamento de Impasses




Centro de Informática
end_transaction
Universidade Federal de Pernambuco
Banco de Dados Distribuídos e Móveis – 2012.1
cin.ufpe.br
67