Transações de SGBDs

Ramon Duarte de Melo

Universidade Federal do Rio de Janeiro

ramonduarte@poli.ufrj.br

17 de novembro de 2018

Sumário

Glossário

Recurso

No entanto, dentro do contexto da disciplina de *Bancos de Dados*, consideraremos como um subconjunto qualquer de dados (um registro, ou uma coluna, ou mesmo toda a base de dados).

Cadeado (ou *lock*)

Estrutura de dados implementada atomicamente pelo sistema operacional que oferece controle de acesso concorrente a um determinado recurso através da exclusão mútua.

Requisição de cadeado

Feitas pela aplicação ao gerenciador de concorrência (*lock manager*). A transação fica bloqueada até que receba o sinal de cadeado concedido.

Cadeado binário

Impede que qualquer outra aplicação tenha acesso concorrente ao recurso. Não são efetivamente utilizados em sistemas onde múltiplas aplicações executem simultaneamente.

Cadeado exclusivo (*lock X*)

Permite a livre leitura e/ou escrita em modo exclusivo.

Cadeado compartilhado (*lock S*)

Permite somente a leitura, em modo concorrente a outras transações de leitura.

Deadlock

Situação em que conflitos de acesso previnem as transações envolvidas de chegarem a um acordo em que ambas possam prosseguir.

Granularidade

Porção da base de dados que um único recurso representa.

Serialização

Característica de um escalonamento reprodutível num cenário hipotético em que cada transação é mutuamente exclusiva e executada uma após a outra.

Recuperação

Característica de um escalonamento que garante que nenhuma de suas transações exija um *rollback* uma vez que concluam com sucesso.

Recuperação estrita

Característica de um escalonamento recuperável que prevê uma ordem estrita de *rollbacks* para transações abortadas.

Protocolos baseados em cadeados

Para esta apresentação, consideraremos somente o conjunto de controles de concorrência baseados em cadeados (*lock-based*).

Os conjuntos de controles de concorrência baseados em *timestamps*, híbridos ou otimistas (com validação pós-execução) serão desconsiderados.

Nestes protocolos, uma aplicação que deseja executar uma transação requisita uma permissão de leitura (*lock S*) ou leitura e escrita (*lock X*) ao gerenciador de concorrência e aguarda numa fila.

A concessão ocorrerá quando o cadeado solicitado for compatível com os demais cadeados (ativos e pendentes).

Um sistema adequadamente projetado deve respeitar o princípio de justiça do acesso aos recursos, isto é, toda transação cujo acesso ao recurso seja permitido deve eventualmente obter o cadeado solicitado.

Matriz de compatibilidade de cadeados

	lock S	**lock X**
lock S	✓	×
lock X	×	×

Tabela: Cadeados somente-leitura toleram acesso concorrente.

Upgrades e *downgrades*

Upgrade: *lock S* para *lock X*

Só é possível se não houver nenhuma outra transação com um cadeado exclusivo no recurso. Caso contrário, deverá entrar na fila e esperar que os cadeados exclusivos anteriores sejam liberados.

Downgrade: *lock X* para *lock S*

Não é necessária nenhuma checagem, pois se assume que o cadeado exclusivo seja único. A conversão é simples e pode ser realizada imediatamente.

O uso de cadeados por si só não é suficiente para garantir serialização.

Example (Operação de leitura simples)

```
resource Q, R;
if (this->getSLock(Q)) {
    this->read(Q);
    this->release(Q); }
// !!! DANGER ZONE !!!
if (this->getSLock(R)) {
    this->read(R);
    this->release(R); }
this->compute(Q + R);
```

No exemplo acima, se Q ou R forem atualizados na transição entre a obtenção dos cadeados, o resultado será distinto do esperado pelo desenvolvedor.

2PL: *Two-Phase Locking*

Este protocolo garante a produção de escalonamentos serializáveis (baseados na ordem dos *lock points*), que são as aquisições do último cadeado) e é composto de duas fases sequenciais:

Fase de Crescimento

Etapa em que as transações requerem os cadeados ou *upgrades*. Nesta fase, as transações não podem liberar cadeados ou fazer *downgrade* de sua possessão.

2PL: *Two-Phase Locking*

Fase de Retração

Etapa em que as transações liberam os cadeados ou fazem *downgrades*. Nesta fase, as transações não podem requerer novos cadeados ou *upgrades*.

Ainda assim, do desenvolvedor é requerida a inserção manual das instruções de *locking*.

Aquisição automática de cadeados

Example (Operação de leitura do recurso R)

```
resource R;
if (this->getLock(R)) {
    this->read(R);
} else {
    if !(isXLocked(R)) {
        this->getSLock(R);
        this->read(R);
    }
}
this->commit();
this->release(R);
```

Example (Operação de escrita do recurso R)

```
resource R;
if (this->getXLock(R)) {
    this->write(R);
} else {
    if !(xLocked(R)) {
        if (this->hasSLock(R)) {
            this->upgradeToXLock(R);
        } else {
            this->getXLock(R);
        }
        this->read(R);
    }
}
this->commit();
this->release(R);
```

Implementação

O cérebro do 2PL é um processo chamado "gerenciador de concorrência" ("*lock manager*"), para o qual as aplicações requerem cadeados e enviam *releases*) antes e depois de executarem transações.

O gerenciador de concorrência deve responder a todas as aplicações com uma destas orientações:

- Confirmação do cadeado: a transação está autorizada a prosseguir.
- *Rollback*: a transação não poderá prosseguir e a aplicação deverá desfazê-la.

Tabela de cadeados

Para tanto, o gerenciador mantém uma tabela de cadeados (*lock table*), uma estrutura de dados especial que registra cadeados garantidos, negados e pendentes.

A tabela é mantida em memória física (RAM) como uma tabela *hash* indexada pelo nome do recurso cujo acesso foi requisitado. Nela, são gravados também o identificador da transação, o tipo de cadeado requisitado e um ponteiro para a próxima requisição da fila.

Tabela de cadeados

ID transação	Recurso ID	tipo de lock	ponteiro próx.
T1	R122	S	0x009F03BC
T2	T12	S	0x00AF13CD
T1	C2349	S	0x00795CF1
T3	R122	X	0x00A01403

Tabela: Representação de uma tabela de cadeados.

Para economizar espaço, o padrão é guardar somente o estado de recursos cujos cadeados têm requisições ativas ou pendentes. Assume-se que todo recurso ausente da tabela está disponivel.

Tabela de cadeados (*lock table*)

Organização das requisições de cadeados

1. Novas requisições entram no fim da fila de cada recurso e são concedidas quando forem compatíveis com as demais.
2. Liberações (*releases*) de cadeados removem a requisição da fila e as requisições pendentes são reavaliadas.
3. Se um *rollback* ou *abort* é emitido, todo cadeado concedido e/ou pendente da transação é descartado.

2PL: Vantagens

A principal vantagem do 2PL (e a razão por ser a estratégia mais comum em sistemas distribuídos) é produzir um escalonamento que é garantido de ser serializável, isto é, reprodutível como se cada transação ocorresse uma após a outra.

Example

Exemplo 1

2PL: Vantagens

Example

Exemplo 2

2PL: Desafios

Nem todos os cenários de conflito podem ser serializados através do 2PL.

Example

Exemplo 3

Example

Exemplo 4

2PL: Desafios

Ainda assim, 2PL oferece uma alternativa de serialização de conflitos mesmo na ausência de maiores informações.

- Caso haja uma transação Ti que não use 2PL, é possível coordenar as transações Tj de forma que, para cada par (Ti, Tj), não haja conflitos de serialização.

2PL: Desafios

2PL não necessariamente previne a existência de *deadlocks*. Um exemplo clássico é o *lock* cruzado, em que múltiplas transações necessitam do mesmo conjunto de recursos e cada uma segura o cadeado de um subconjunto dele, impedindo as demais de prosseguirem.

Example

Exemplo 6

2PL: Desafios

2PL prevê apenas uma forma de resolução de tais conflitos: *rollback* de ao menos uma das transações e descarte de suas permissões especiais.

Example

Exemplo 7

2PL: Desafios

Por conta disto, há a possibilidade de "*starvation*", cenário em que uma transação que necessita de acesso de escrita é repetidamente desfeita por conta da sequência de permissões de leitura concedidas sobre o mesmo recurso a outras transações.

Example

Exemplo 8

2PL: Estratégias

Caso ocorram *deadlocks*, é papel do gerenciador de concorrência garantir a integridade, a consistência e a recuperação do sistema.

Uma das consequências a serem evitadas é o *rollback* em cascata.
Algumas estratégias para evitá-lo são:

Normal ou padrão

Cada transação deve requerer seus cadeados quando necessitar dos recursos bloqueados e liberá-los logo em seguida. O mais simples de implementar, mas não garante um escalonamento livre de *deadlocks*, ou um escalonamento estrito de recuperação.

2PL: Estratégias

Conservadora ou estática

Cada transação deve requerer os cadeados antes de executar e liberá-los logo em seguida. A implementação exige conhecimento prévio do plano de execução, mas garante um escalonamento livre de *deadlocks*. Ainda assim, não garante que ele seja estrito de recuperação, por permitir leituras sujas (dirty reads).

2PL: Estratégias

Estrita ou restrita

Cada transação deve requerer seus cadeados exclusivos antes de iniciar a execução e segurá-los até que complete com sucesso ou aborte. A implementação exige conhecimento prévio do plano de execução, mas garante um escalonamento estrito de recuperação. *Deadlocks* ainda podem ocorrer. Apesar disso, é uma das estratégias mais comuns (não só em SGBDs), graças aos algoritmos de detecção e tratamento de *deadlocks*. Exemplos de SGBDs que o utilizam incluem o *SQL Server* (Microsoft) e o *DB2* (IBM).

2PL: Estratégias

Rigorosa

Cada transação deve requerer todos os seus cadeados (exclusivos ou commpartilhados) antes de iniciar a execução e segurá-los até que complete ou aborte. As transações passam a ser serializadas pela ordem de commit. A implementação exige conhecimento prévio do plano de execução, mas garante um escalonamento livre de *deadlocks* e estrito de recuperação. Embora seja a implementação menos prática de todas, é uma estratégia bastante empregada por SGBDs.

Demais estratégias de prevenção de *deadlocks*

Outras estratégias de prevenção de *deadlocks* incluem:

Predeclaração

Cada transação deve requerer todos os cadeados necessários antes mesmo de iniciar sua execução.

Ordenação parcial

Cada transação só pode enviar requisições numa ordem parcial (menos estrita que a ordenação total para evitar *overhead*).

Demais estratégias de prevenção de *deadlocks*

Timestamps não-preemptivos (*wait-die*))

Transações antigas podem esperar recursos das posteriores, mas transações recentes são sempre desfeitas. O risco de *starvation* é alto porque uma transação pode tomar *rollback* repetidamente.

Timestamps preemptivos (*wound-wait*))

Transações forçam *rollback* das posteriores, mas as recentes podem esperar. O risco de *starvation* é significativamente menor.

Demais estratégias de prevenção de *deadlocks*

Em ambos os casos de estratégias baseadas em *timestamps*, a *timestamp* original da transação é mantida caso ela seja executada novamente, de forma a prevenir *starvation*.

Timeout

Se o cadeado não for concedido dentro de um certo intervalo de tempo, a transação é desfeita. Simples de implementar, porém a determinação do intervalo adequado é crucial para seu bom funcionamento.

Detecção de *deadlocks*

Os algoritmos mais eficazes de detecção de *deadlocks* são os baseados no problema de escalonamento de tarefas (*jobshop scheduling problem* ou *JSSP*).

Detecção de *deadlocks*

O problema pode ser descrito por um conjunto de n transações fJig1≤j≤n que requerem um conjunto de m recursos fMrg1≤r≤m.

Cada transação tem uma sequência própria de recursos a serem processados.

O processamento da transação Jj na máquina Mr é chamada de operação Ojr.

A operação Ojr requer o uso exclusivo de Mr por uma duração ininterrupta pjr, que é seu tempo de processamento.

Detecção de *deadlocks*

Um escalonamento eficaz é um conjunto de operações sequenciais fcjrg1≤j≤n;1≤r≤m que satisfazem estes critérios.

Caso haja um ciclo no escalonamento, isto é, ¡se existe um ciclo¿, então há um estado de *deadlock* presente no escalonamento que precisa ser endereçado.

O algoritmo deve buscar por ciclos frequentemente, para evitar *rollbacks* em cascata.

Recuperação de *deadlocks*

Quando um *deadlock* é detectado, alguma transação deverá ser desfeita (*rolled back*) para quebrá-lo. A seleção da vítima deve ser feita de forma a minimizar o prejuízo de tempo.

A implementação mais simples é a do *rollback* total, que aborta a transação e a desfaz completamente. Porém, esta operação pode ser muito custosa.

Uma implementação mais complicada, porém mais eficiente, é o *rollback* parcial até que o *deadlock* seja desfeito.

Se a mesma transação é sempre escolhida como vítima, há o risco de *starvation*.

Granularidade múltipla

Idealmente, um SGBD deve oferecer às aplicações métodos de acesso indexado através de uma interface flexível de controle de acesso concorrente aos recursos.

Uma das dimensões desta flexibilidade se dá através da granularidade dos cadeados, permitindo que pequenas porções de dados sejam aninhadas em porções maiores.

Acesso indexado

A hierarquia de indexação do acesso a estas granularidades pode ser representada graficamente como uma árvore:

Acesso indexado

Quando uma transação requisita um cadeado explicitamente, o gerenciador de concorrência implicitamente aplica o cadeado a todos os recursos filhos daquele índice hierárquico.

A granularidade do bloqueio, portanto, pode ocorrer em índices de:

Granularidade fina

Próxima às folhas da árvore, permitem maior grau de concorrência mas aumentam o custo computacional do gerenciamento (*overhead*).

Granularidade grossa

Próxima à raiz da árvore, simplificam o gerenciamento mas reduzem a capacidade de acessos concorrentes.

Acesso indexado

Os índices da árvore de granularidade são:

1. base de dados
2. setor
3. arquivo
4. registro

Tipos de intenção de cadeados

A partir do momento em que a granularidade múltipla passa a ser uma possibilidade oferecida pelo SGBD, os protocolos baseados em cadeados precisam passar a suportar três tipos adicionais de cadeados:

lock IS (intenção de compartilhamento)

Requisita somente acessos concorrentes de leitura aos nós filhos do recurso.

lock IX (intenção de exclusividade)

Requisita acessos de escrita e leitura aos nós filhos do recurso. A distinção será feita numa granularidade mais fina.

Tipos de intenção de cadeados

lock SIX (compartilhado com intenção de exclusividade)

Requer somente acesso de leitura aos nós filhos do recurso, mas acessos de escrita adicionais serão requeridos em granularidade mais fina.

A vantagem de introduzir estes cadeados adicionais é a diminuição no tempo de resposta (*overhead*) do gerenciamento de concorrência, já que eliminam a necessidade de checar todos os filhos do nó requisitado.

Matriz atualizada e compatibilidade de cadeados

	IS	IX
S	**SIX**	**X**
IS	✓	✓
✓	✓	×
IX	✓	✓
×	×	×
S	✓	×
✓	×	×
SIX	✓	×
×	×	×
X	×	×
×	×	×

Tabela: Cadeados adicionais oferecem opções intermediárias de controle de concorrência.

Esquematização dos cadeados sob granularidade múltipla

Uma transação pode obter cadeado de qualquer nó da árvore, desde que:

1. As regras da matriz atualizada de compatibilidade de cadeados sejam observadas.
2. A raiz da árvore seja a primeira a ser reservada, e possa sê-la em qualquer modo.
3. Um nó pode ser requisitado em nodo S ou IS somente se o nó pai dele estiver sob um cadeado IX ou IS pertencente à mesma transação.

Esquematização dos cadeados sob granularidade múltipla

4. Um nó pode ser requisitado em nodo X, SIX ou IX somente se o nó pai dele estiver sob um cadeado IX ou SIX pertencente à mesma transação.
5. Um nó só pode ser requisitado se a transação não houver liberado nenhum nó anterior (ou seja, se está na fase de crescimento do 2PL).
6. Um nó só pode ser liberado se nenhum de seus nós filhos estiver sob cadeado da mesma transação.

Esquematização dos cadeados sob granularidade múltipla

A fase de crescimento deve ocorrer sempre no sentido raiz $\Rightarrow$ *folhas da árvore*, *enquanto a fase de retração deve ocorrer sempre no sentido oposto (folhas → raiz)*.

Se houver cadeados demais num mesmo nível da árvore, o gerenciador de concorrência poderá elevá-los a um nó de maior hierarquia. Este procedimento é chamado de escalação da granularidade do cadeado e deverá obedecer à matriz atualizada de compatibilidade.

Referências bibliográficas

Title of the publication *Journal Name* 12(3), 45 – 678.