Schedulers Otimistas

- Técnicas pessimistas
  - *overhead* no processamento de transações
    - executam verificações e ações antes de qualquer operação no BD para garantir a serializabilidade (solicitação de bloqueio, teste de TS)
- Técnicas otimistas
  - não realizam nenhuma verificação durante o processamento da transação
    - pressupõem nenhuma ou pouca interferência
    - verificações de violação de serializabilidade feitos somente ao final de cada transação
    - técnica mais conhecida: Técnica de Validação

Scheduler Baseado em Validação

- Técnica na qual atualizações de uma transação *Tx* são feitas sobre cópias locais dos dados
- Quando *Tx* solicita *commit* é feita a sua validação
  - Tx violou a serializabilidade?
    - SIM: *Tx* é abortada e reiniciada posteriormente
    - NÃO: atualiza o BD a partir das cópias dos dados e encerra *Tx*

Técnica de Validação

- Cada transação *Tx* passa por 3 fases:
  1. Leitura
     - *Tx* lê dados de transações *committed* do BD e atualiza dados em cópias locais
  2. Validação
     - análise da manutenção da serializabilidade de conflito caso as atualizações de *Tx* sejam efetivadas no BD
  3. Escrita
     - se fase de Validação for OK, aplica-se as atualizações de *Tx* no BD e *Tx* encerra com sucesso; caso contrário, *Tx* é abortada

Técnica de Validação

- Duas listas de dados são mantidas para *Tx*
  - lista-READ(*Tx*) : conjunto de dados que *Tx* leu
  - lista-WRITE(*Tx*): conjunto de dados que *Tx* atualizou
- Três *timestamps* são definidos para *Tx*
  - *TS-Start(Tx)*: início da fase de leitura de *Tx*
  - *TS-Validation(Tx)*: início da fase de validação de *Tx*
  - *TS-Finish(Tx)*: término da fase de escrita de *Tx*

Funcionamento da Técnica

- Durante a fase de Leitura
  - *Tx* lê / atualiza dados; lista-READ(*Tx*) e list-WRITE(*Tx*) vão sendo alimentadas
- Durante a fase de Validação
  - três condições são testadas entre *Tx* e toda transação *Ty* que já encerrou com sucesso ou está sofrendo validação
    - se alguma das condições for VERDADEIRA para toda *Ty*
      - *Tx* passa para a fase de Escrita e encerra com sucesso
    - caso contrário
      - há interferência entre *Tx* e *Ty*
      - *Tx* é abortada e suas cópias locais são descartadas

Condições para Validação de *Tx*

- Condição 1
  - *TS-Finish(Ty) < TS-Start(Tx)*
    - se *Ty* encerrou suas atualizações antes de *Tx* iniciar, então *Tx* não interfere em *Ty*
- Exemplo
  - $H_{V\text{-}C1}$ = s1 r1(A) s2 r2(B) w1(A) v1 c1 w2(A) v2 c2 s*x* r*x*(A) s3 r3(Z) w*x*(A) v*x* c*x* . . .

  TS-Finish(T1) < *TS-Start(Tx)* E
  TS-Finish(T2) < *TS-Start(Tx)*

Condições para Validação de *Tx*

- Condição 2
  - *TS-Start(Tx) < TS-Finish(Ty) < TS-Validation(Tx)* E lista-READ(*Tx*) ∩ lista-WRITE(*Ty*) = ϕ
    - *Ty* encerrou durante a execução de *Tx* e *Tx* não leu nenhum dado que possa ter sido atualizado por *Ty* (não há risco de *Ty* ter interferido nos dados lidos por *Tx*)
- Exemplo
  - $H_{V\text{-}C2}$ = s1 r1(C) s2 r2(B) w1(C) v1 c1 s*x* r*x*(B) r*x*(C) w2(A) v2 c2 w*x*(B) s3 r3(Z) v*x* c*x* . . .
    - *T1* atende condição 1 em relação à *Tx*
    - *T2* atende condição 2 em relação à *Tx*:
      - lista-READ(*Tx*) = {B, C}
      - lista-WRITE(*T2*) = {A}

Condições para Validação de *Tx*

- Condição 3
  - *TS-Validation(Ty) < TS-Validation(Tx) E*
    lista-READ(*Tx*) ∩ lista-WRITE(*Ty*) = ϕ E
    lista-WRITE(*Tx*) ∩ lista-READ(*Ty*) = ϕ E
    lista-WRITE(*Tx*) ∩ lista-WRITE(*Ty*) = ϕ
    - *Ty* já estava em validação, mas não há operações em conflito entre ela e *Tx*
- Exemplo
  - $H_{VAL\text{-}C3}$ = s1 r1(C) s2 r2(B) w1(C) v1 c1 s*x* r*x*(B) s3 r3(C) r*x*(C) w2(A) v2 c2 w3(Y) w3(Z) v3 w*x*(B) v*x* c*x* . . .
    - *T1* atende condição 1 em relação à *Tx*
    - *T2* atende condição 2 em relação à *Tx*
    - *T3* atende condição 3 em relação à *Tx*:
      - lista-READ(*T3*) = {C} lista-WRITE(*T3*) = {Y, Z}
      - lista-READ(*Tx*) = {B, C} lista-WRITE(*Tx*) = {B}

Scheduler Baseado em Validação

- Vantagens
  - reduz o *overhead* durante a execução de *Tx*
  - evita aborto em cascata
    - *Tx* não grava no BD antes de suas atualizações serem validadas em memória
      - se *Tx* interfere em outra *Ty committed* ou em validação, suas atualizações são descartadas
- Desvantagem
  - se houve interferência entre *Tx* e outras transações (isso não é esperado pois a técnica é otimista), isso é descoberto somente ao final da execução de *Tx* (na validação) e só após essa validação *Tx* pode ser reiniciada

Exercícios 6

1. Apresente um escalonamento que não seja serializável por validação para as transações abaixo:

   T1: r(Y) w(Y) w(Z)

   T2: r(X) r(T) w(T)

   T3: r(Z) w(Z)

   T4: r(X) w(X)
2. Um *scheduler* baseado em validação garante um escalonamento passível de recuperação pelo *recovery*?

Bloqueios e Granularidade

- Grânulo
  - porção do BD
    - atributo, tupla, tabela, bloco, ...
  - níveis de granularidade
    - granularidade fina
      - porção pequena do BD ⇒ muitos itens de dados
      - maior número de itens de dados a serem bloqueados e controlados pelo *scheduler*
      - maior concorrência
    - granularidade grossa
      - porção grande do BD ⇒ menos itens de dados
      - menor número de itens de dados a serem bloqueados e controlados pelo *scheduler*
      - menor concorrência

Bloqueios e Granularidade

- Na prática, transações podem realizar bloqueios em vários níveis de granularidade
  - *Tx* atualiza uma tupla; *Ty* atualiza toda uma tabela
- Algumas questões
  - se *Ty* quer atualizar toda uma tabela, *Ty* deve bloquear TODAS as tuplas?
  - se *Tx* bloqueou uma tupla da tabela *T* (bloqueio fino) e *Ty* quer bloquear *T* (bloqueio grosso), como *Ty* sabe que *Tx* mantém um bloqueio fino?
- Solução
  - gerenciar bloqueios por níveis de granularidade
    - além do uso de bloqueios S e X, uso de bloqueios de intenção

Bloqueios de Intenção

- Indicam, em grânulos mais grossos, que *Tx* está bloqueando algum dado em um grânulo mais fino
  - vê o BD como uma árvore de grânulos
- Tipos de bloqueios de intenção
  - IS (*Intention-Shared*)
    - indica que um ou mais bloqueios compartilhados serão solicitados em nodos descendentes
  - IX (*Intention-eXclusive*)
    - indica que um ou mais bloqueios exclusivos serão solicitados em nodos descendentes
  - SIX (*Shared-Intention-eXclusive*)
    - bloqueia o nodo corrente no modo compartilhado, porém um ou mais bloqueios exclusivos serão solicitados em nodos descendentes

Exemplo

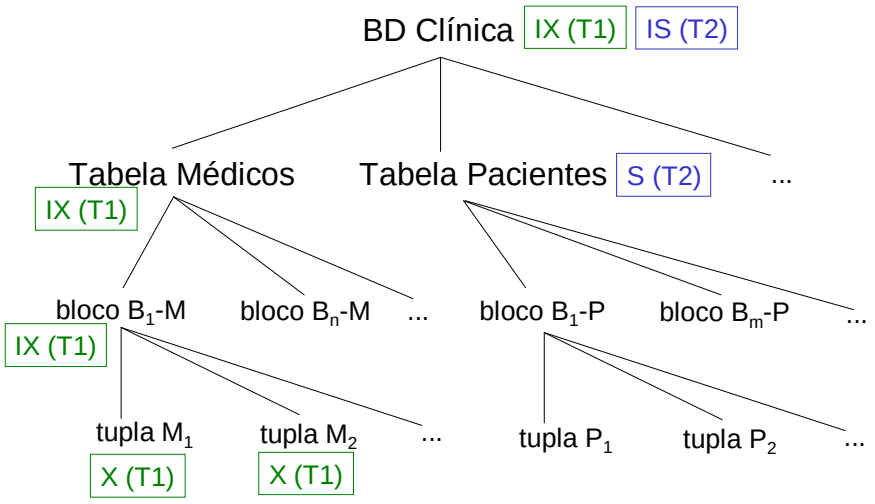

Tabela de Compatibilidade de Bloqueios

	IS	IX	S	SIX	X
IS	verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso
IX	verdadeiro	verdadeiro	falso	falso	falso
S	verdadeiro	falso	verdadeiro	falso	falso
SIX	verdadeiro	falso	falso	falso	falso
X	falso	falso	falso	falso	falso

Técnica de Bloqueio de Várias Granularidades

- Protocolo que atende às seguintes regras:
  1. A tabela de compatibilidade de bloqueios deve ser respeitada
  2. A raiz da árvore deve ser bloqueada em primeiro lugar, em qualquer modo
  3. Um nodo *N* pode ser bloqueado por *Tx* no modo S ou IS se o nodo pai de *N* já estiver bloqueado por *Tx* no modo IS ou IX
  4. Um nodo *N* pode ser bloqueado por *Tx* no modo X, IX ou SIX se o nodo pai de *N* já estiver bloqueado por *Tx* no modo IX ou SIX
  5. *Tx* pode bloquear um nodo se não tiver desbloqueado nenhum nodo (é 2PL!)
  6. *Tx* pode desbloquear um nodo *N* se nenhum dos filhos de *N* estiver bloqueado por *Tx*

Técnica de Bloqueio de Várias Granularidades

- Serializabilidade é garantida
  - segue-se um protocolo 2PL
- Obtenção de bloqueios é *top-down*
- Liberação de bloqueios é *bottom-up*
- Vantagens
  - reduz o *overhead* na imposição de bloqueios
  - adequada a qualquer tipo de transação
    - alcance de dados pequeno, médio ou grande
- Desvantagens
  - maior controle e registro de bloqueios
  - não está livre de *deadlock*

Exemplo

- T1: deseja atualizar os dados do médico com CRM 100 (está no bloco B_1-M) e do paciente com CPF 200 (está no bloco B_2-P)
- T2: deseja atualizar os médicos ortopedistas (estão no bloco B_{10}-M)
- T3: deseja ler os dados do médico com CRM 50 (está no bloco B_1-M) e todos os dados de pacientes
- Escalonamento (apenas os bloqueios são mostrados)

$H_{2PL\text{-}VG}$ = lix1(BD) lix1(Médicos) lix2(BD) lis3(BD) lis3(Médicos) lis3(Médicos.BlocoB_1-M) lix1(Médicos.BlocoB_1-M) lx1(Médicos[CRM=100]) lix2(Médicos) lx2(Médicos.BlocoB_{10}-M) ls3(Médicos[CRM=50]) lix1(Pacientes) lix1(Pacientes.BlocoB_2-P) lx1(Pacientes[CPF=200]) u1(Pacientes[CPF=200]) u1(Pacientes.BlocoB_2-P) u1(Pacientes) ls3(Pacientes) u2(Médicos.BlocoB_{10}-M) u2(Médicos) u2(BD) u1(Médicos[CRM=100]) u1(Médicos.BlocoB_1-M) u1(Médicos) u1(BD) u3(Médicos[CRM=50]) u3(Médicos.BlocoB_1-M) u3(Médicos) u3(Pacientes) u3(BD)

Exercícios 7

1. Apresente um escalonamento concorrente 2PL de várias granularidades (considerando os níveis: BD-Tabela-Tupla) para as transações abaixo:

   T1: r(Médicos[CRM=100]) w(Médicos[CRM=100])
   w(Pacientes[CPF=101])
   T2: r(Médicos) r(Pacientes[CPF=200])
   w(Pacientes[CPF=200])
   T3: r(Pacientes[CPF=101]) w(Pacientes[CPF=111])
   T4: r(Médicos)
   w(Médicos[especialidade = 'ortopedia'])

   Obs.: o médico com CRM=100 é ortopedista.