Sistemas Distribuídos
Aula 16

Roteiro

- Sistema transacional
- ACID
- Exemplos
- *2-Phase Locking* (2PL)
- Exemplo e deadlocks

Sistema Transacional

- Considere sistema que armazena *objetos*
  - objeto = elemento fundamental de dado
- Valor de todos objetos define *estado global* do sistema
- Operação de leitura e escrita concorrente nos objetos (*multi-threaded* ou *multi-processed*)
  - operações são denominadas *transações*
- Transação pode ser complexa
  - composição de transações elementares
  - ex. ler diversos valores, escrever diversos valores de forma condicional

Exemplo: Sistema Bancário

- Objeto = valor do saldo de uma conta
  - saldo é o dado elementar
- Estado global = saldo de todas as contas
- Transações elementares de leitura e escrita nos saldos
- Transações complexas manipulando diferentes contas de forma condicional

```
se c1.saldo > 1000
  c1.saldo -= 500
  se c2.saldo > 1000
    c2.saldo -= 500
    c3.saldo += 1000
  senao
    c3.saldo += 500
```

Como garantir um bom funcionamento?

Propriedades das Transações

- **Ideia:** construir sistema cujas transações oferecem determinadas propriedades
  - facilita uso do sistema, oferece previsibilidade, garante corretude
- **ACID**: *Atomicity, Consistency, Isolation, Durability*
- ***A****tomicity*: transação é realizada por inteiro ou é abortada por inteiro. Em caso de abortada, estado global não é modificado
- ***C****onsistency*: cada transação preserva propriedades do estado global (propriedades dependem do sistema)
  - ex. sistema bancário preserva soma dos saldos

Propriedades das Transações

- ***I****solation*: transação é realizada como se fosse única no sistema. *Serializable, thread-safe*: transações ocorrem uma depois da outra
  - *parecem ocorrer*, pois podem ser independentes e ocorrerem concorrentemente
- ***D****urability*: ao concluir uma transação (*commit*), sistema transiciona para novo estado global que permanece, independente de eventos externos
  - ex. falta de energia
- Modelo clássico (anos 70, por Jim Gray) e ainda muito utilizado por SGBDs

Como garantir ACID?

Exemplo Bancário

- Três funções (transações): retirada, depósito, e transferência
- Transferência usa retirada e depósito
  - transação composta de outras transações

```
retirada(conta, valor) {
    saldo = get_saldo(conta)
    saldo = saldo – valor
    se (saldo > 0)
        put_saldo(conta, saldo)
        retorna saldo
    retorna -1
}
```

```
deposito(conta, valor) {
    saldo = get_saldo(conta)
    saldo = saldo + valor
    put_saldo(conta, saldo)
    retorna saldo
}
```

```
transferencia(c1, c2, v) {
    se (retirada(c1,v) >= 0)
        deposito(c2,v)
        retorna 0
    retorna -1
}
```

Exemplo Bancário

- Considere saldo(x)=100, saldo(y)=saldo(z)=0
- Duas transações T1= transf(x,y,60), T2= transf(x,z,70), executadas de forma simultâneas
- Possíveis resultados da condição de corrida?
- T1 ocorre por inteiro antes de T2:
  - saldo(x)=40, saldo(y)=60, saldo(z)=0
- T2 ocorre por inteiro antes de T1:
  - saldo(x)=30, saldo(y)=0, saldo(z)=70
- T1 e T2 ocorrem simultaneamente:
  - saldo(x)=30, saldo(y)= 60, saldo(z)=70
  - saldo(x)=40, saldo(y)= 60, saldo(z)=70
- Viola **I**solation e **C**onsistency: soma dos saldos deve ser constante

Banco não gosta!

Garantindo ACID

- Como garantir ACID?
- ***Locks to the rescue!***

- Ideia 0: Locks na função de transferência

```
transferencia(c1, c2, v) {
    acquire(t)
    se (retirada(c1,v) >= 0)
        deposito(c2,v)
        release(t)
        retorna 0
    release(t)
    retorna -1
}
```

- Garante ACID para o exemplo anterior?
- Sim, mas muito ineficiente
- Permite apenas uma transferência no sistema de cada vez
  - *t* é variável global
- Não permite transferência entre contas não relacionadas

Tentativa 1

- Ideia melhor: Criar um lock para cada conta

```
transferencia(c1, c2, v) {
   acquire(c1)
   se (retirada(c1,v) >= 0)
      release(c1)
      acquire(c2)
      deposito(c2,v)
      release(c2)
      retorna 0
   release(c1)
   retorna -1
}
```

- Garante ACID?
- Não, não oferece consistência
  - entre release(c1) e acquire(c2), soma dos saldos não está preservada
  - c1 e c2 podem ser manipuladas em outras transações!
- Como fazer funcionar?

Tentativa 2

- Ideia melhor: Criar um lock para cada conta
- Pegar locks em sequência, liberar em sequência, após conclusão da transação

```
transferencia(c1, c2, v) {
   acquire(c1)
   se (retirada(c1,v) >= 0)
      acquire(c2)
      deposito(c2,v)
      release(c1)
      release(c2)
      retorna 0
   release(c1)
   retorna -1
}
```

- Não funciona sempre
- saldo(x)=saldo(y)=100
- T1= transf(x,y,30),
  T2= transf(y,x,20)
- T1 faz o acquire de x,
  T2 faz o acquire de y
- T1 faz o acquire de y (bloqueia),
  T2 faz o acquire de x (bloqueia)
- **Deadlock!**

Tentativa 3

- Ideia melhor: Criar um lock para cada conta
- Pegar locks em sequência, liberar em sequência, após transação
- Pegar locks em alguma ordenação global

```
transferencia(c1, c2, v) {
   acquire(min(c1,c2))
   acquire(max(c1,c2))
   se (retirada(c1,v) >= 0)
      deposito(c2,v)
      release(c1)
      release(c2)
      retorna 0
   release(c1)
   release(c2)
   retorna -1
}
```

- Pega primeiro o lock da menor conta, depois o lock da maior
- saldo(x)=saldo(y)=100
- T1= transf(x,y,30),
  T2= transf(y,x,20)
- Supor que x < y
- T1 faz o acquire de x,
  T2 faz o acquire de x (bloqueia)
- T1 faz o acquire de y, e libera os dois locks
- **Funcionou!**

Two Phase Locking (2PL)

- Garantir ACID de forma geral não é fácil
- 2PL: mecanismo para controle de concorrência
  - garante atomicidade (e outras coisas)
- Utiliza dois tipos de lock para *cada objeto* (dado)
  - *read locks, write locks*
  - read lock permite outro read lock, mas não write lock
  - write lock não permite nenhum outro lock
- Duas fases, para cada transação:
  - Fase 1 (*Expanding*): locks são adquiridos, nenhum lock é liberado
  - Fase 2 (*Shrinking*): locks são liberados, nenhum é lock adquirido

Two Phase Locking (2PL)

- Duas variações do 2PL
- Strict Two Phase Locking (S2PL)
  - fase 1 igual, fase 2: liberar todos os write locks apenas ao final da transação (read locks podem ser liberados aos poucos)
- Strong Strict Two Phase Locking (SS2PL)
  - fase 1 igual, fase 2: liberar todos os read e write locks apenas ao final da transação
- Variações oferecem melhores propriedades, mas reduzem concorrência
  - ex. reduzem chance de *deadlocks*

Duas Fases da Transação

- SS2PL implementado dividindo transação em duas fases
- **Fase 1: Preparação**
  - adquirir locks de leitura e determinar tudo que é necessário para alterar estado global
  - gerar lista com mudanças no estado global
- **Fase 2: Commit ou Abort**
  - adquirir locks de escrita
  - atualizar estado global (**commit**)
  - abortar a transação sem modificar estado global, se algum imprevisto ocorreu (**abort**)
  - liberar todos os locks (leitura e escrita)
- Facilita gerenciamento dos locks e execução das transações elementares

Exemplo

- **acquire**, **commit** e **abort** são funções do sistema transacional

```
transferencia(c1, c2, v) {
   L={c1,c2}  // locks
   U={}       // updates
   acquire(L)
   s1 = get_saldo(c1)
   s2 = get_saldo(c2)
   se (s1 - v >= 0)
      s1 = s1 – v
      s2 = s2 + v
      U ={“put_saldo(c1,s1)”,
          “put_saldo(c2,s2)”}
      commit(U,L)
      retorna 0
   abort(L)
   retorna -1
}
```

- **acquire(L)**: obtém todos os locks de leitura em L (em alguma ordenação global)

- **commit(U,L)**: obtém todos os locks de escrita em L (em alguma ordenação global), realiza todas as operações em U, e libera todos os locks em L (leitura e escrita)

- **abort(L):** libera todos os locks em L (leitura e escrita) sem modificar estado global

Ainda Problemas

- O que pode acontecer com este código?

```
transferencia(c1, c2, v) {
   L={c1,c2}  // locks
   U={}       // updates
   acquire(L)
   s1 = get_saldo(c1)
   s2 = get_saldo(c2)
   se (s1 - v >= 0)
      s1 = s1 – v
      s2 = s2 + v
      U ={“put_saldo(c1,s1)”,
          “put_saldo(c2,s2)”}
      commit(U,L)
      retorna 0
   abort(L)
   retorna -1
}
```

- Duas operações distintas que usam os mesmos locks
  - T1 = transf(c1,c2,v1),
  - T2 = transf(c1,c3,v2)
- Ao executarem commit() não conseguem lock de escrita
  - lock de leitura de c1 está preso na outra transação
- **Deadlock!**

Resolvendo Deadlocks

- 2PL (ou SS2PL) não resolvem deadlocks
  - mecanismo para oferecer ACID
  - reduz a chance de deadlocks
- Sistema transacional precisa detectar a presença de deadlocks
  - ex. ciclos no grafo direcionado de dependências de locks entre transações em execução
  - ex. temporizadores para cada transação
- Abortar transação para quebrar deadlock
  - libera locks da transação, permitindo a execução de outras transações