

Sistemas Distribuídos

Aula 19

Aula passada

- Redes sem fio
- Coordenando acesso ao meio
- Eleição de líder

Aula de hoje

- Sistema transacional
- ACID
- Exemplos
- *2-Phase Locking*

Sistema Transacional

- Considere sistema com muitos objetos (dados)
 - possivelmente distribuídos em diferentes comp.
- Valor de todos objetos define *estado global*
- Operação de leitura e escrita simultânea dos objetos (*multithreaded* ou *multiprocessed*)
 - chamada de transações
- Transação pode ser complexa, diversas partes
 - ler diversos valores, escrever diversos valores condicionadamente

Como garantir bom funcionamento?



Propriedades das Transações

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

Propriedades das Transações

- *Isolation*: transação executa como se fosse a única no sistema (ao ler/escrever dados). *Serializable*, *thread-safe*: transações ocorrem uma depois da outra.
- *Durability*: ao concluir uma transação (*commit*), sistema passa a novo estado global que permanece, independente de eventos externos (ex. falta de energia)
- Modelo clássico (70s, por Jim Gray) e mais utilizado por SGBDs

Como garantir ACID?

Exemplo Bancário

- Três funções: 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 $\text{saldo}(x)=100$, $\text{saldo}(y)=\text{saldo}(z)=0$
- Duas transações $T1 = \text{transf}(x,y,60)$, $T2 = \text{transf}(x,z,70)$, simultâneas
- Sobre ACID, qual é o resultado?
- Temos uma condição de corrida?

```
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  
}
```

ACID não Vem de Graça

- O que acontece se não fizermos nada?
- Duas retiradas simultâneas (*já vimos este filme*)
 - $\text{saldo}(x)=40$ ou 30 , $\text{saldo}(y)=60$, $\text{saldo}(z)=70$
- Viola Isolation(I) e consistência(C): soma dos saldos deve ser constante

```
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  
}
```

Garantindo ACID



- Como garantir ACID?

- ***Locks to the rescue!***

- 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  
}
```

- Funciona? Garante ACID?

- Sim, mas muito ineficiente!

- Permite apenas uma transferência no sistema de cada vez

- t é variável global

- muito ineficiente!

- Como melhorar?

Tentativa 1

- Usar um lock por 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  
}
```

- Funciona? Garante ACID?
- Não, não oferece consistência
 - entre `release(c1)` e `acquire(c2)`, soma dos saldos não está preservada
- Como fazer funcionar?

Tentativa 2

- Usar um lock por cada conta!
- Pegar locks em sequência, liberar em sequência, após 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  
}
```

- Funciona?
- Não, podemos ter deadlock
- $\text{saldo}(x) = \text{saldo}(y) = 100$
- $T1 = \text{transf}(x, y, 30),$
 $T2 = \text{transf}(y, x, 20)$
- O que pode acontecer?
- Como fazer funcionar?

Tentativa 3

- Usar um lock por 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  
}
```

- saldo(x)=saldo(y)=100
- T1= transf(x,y,30),
T2= transf(y,x,20)
- O que vai acontecer?
- Funciona!

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 é liberado
 - Fase 2 (*Shrinking*): locks são liberados, nenhum é 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
- Strong Strict Two Phase Locking (SS2PL)
 - fase 1 igual, fase 2: liberar todos os read e write locks apenas ao final da transação
 - mais comum, oferece melhores propriedades

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)

Exemplo

- SS2PL implementado dividindo transação em duas fases
- Melhor desempenho, facilita gerenciamento

```
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)

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  
}
```

• **Deadlock!**

- Duas operações distintas que usam os mesmos locks
 - ex. `transf(c1,c2,v1)` e `transf(c1,c3,v2)`
- Deadlock ao executarem `commit()`: não conseguem locks de escrita
 - lock de leitura está preso
- 2PL (ou SS2PL) não resolvem deadlocks
- Usar outra técnica (ex. temporizadores)