Concorrência

Sistemas Distribuídos e Tolerância a Falhas

Índice

- Consistência Atómica e sequencial
- “Serializability“
- Controle de concorrência
- One-copy Serializability

Exclusão Mútua

- ✓Técnica usada em programação concorrente.
  - Evita que 2 processos ou threads acedam simultâneamente a um recurso partilhado (recurso denominado secção critica).
    - » Exemplos: Semáforos binários(0 ou 1).
    - » Problemas: Deadlock, starvation.

Consistência Atómica

- Comportamento de uma memória física única.
  - Consideremos que cada acesso individual à memória é uma operação atómica única.
  - Pode ser descrita por:
    - » escrita e leitura estão ordenadas pela ordem física (real).
    - » Naturalmente , a escrita no tempo t e visualizada no tempo t’ ,onde t’ > t.

Consistência Atómica (cont…)

- Quando existem vários processadores, estes podem ter vários níveis de cache, logo, pode haver o problema de “coerência de cache”.
    - A forma utilizada para resolver este problema é, esconder a existência de caches múltiplas aos utilizadores.
    - Assegurando que o sistema tem um comportamento igual como se não existissem caches. (Assegurar a consistência atómica).

Consistência Sequencial

- É um modelo mais fraco.
  - Permite aos processos ler dados das caches desde que:
    - A resultante sequência de memória seja equivalente a execuções de escrita em série .
    - Nota: não força a escrita e leitura pela ordem física.

Consistência Sequencial (cont…)

- Exemplo:
    - Num processo é permitida a leitura de valores fora de prazo na cache mesmo se a escrita já foi transferida para outro processo .

Consistência Sequêncial (cont…)

- Vantagens:
    - Pode ser implementada sem a serialização das operações de memória.
    - Apenas é necessário serializar as operações de escrita.
    - As de leitura basta ordená-las em relação às de escrita.
    - As transacções procedem em paralelo para acessos não conflituosos a diferentes estruturas de dados.

Serializability

- Exclusão Mútua:
  - Funciona por reforçar a ordem em série das sequencias de operações.
    - » Um processo adquire o lock
    - » Acede aos dados
    - » E liberta o lock
    - » Depois outro processo adquire o lock
    - » …..

Serializability (cont…)

- Exclusão mútua:
  - Não é eficaz para programações de larga escala.
    - Exemplo : Uma base de dados com milhares de dados ,programas complexos, escritos por diferentes pessoas que acedem a várias partes da base de dados.
      - » É bastante difícil definir quais as secções criticas.

Serializability (cont…)

- É necessário um mecanismo que :
  - Assegure que estas sequências de operações (transacções) sejam executadas de forma que a sua saída seja a equivalente a uma execução sequêncial sem forçar a execução sequêncial.
  - Isto é ,se 2 transacções acedem a dados diferentes deveriam poder executar em paralelo…. Uma vez que resultaria numa melhor performance.

Controle de Concorrência

- Quando o sistema de gestão de ficheiros é distribuído o problema do acesso partilhado e do controlo da concorrência é mais difícil de resolver.
- Os métodos dividem-se em :
    - Pessimistas: Tentam esconder que existem várias cópias e procuram emular da melhor forma possível o sistema centralizado
    - Optimistas: Tentam não esconder totalmente a replicação e o “caching” em troca de um maior desempenho, adaptação à escala e flexibilidade.

Controle de Concorrência(cont...)

- Garante os resultados correctos em operações concorrentes.
- Ao mesmo tempo que adiciona rapidez na execução.
- Pode ser feito por:
    - Locking: Um gerente centralizado ou distribuído registra todos os *locks* e rejeita pedidos de *lock* em objectos já alocados a outros processos.

Controle de concorrência (cont…)

- Dada uma base de dados com 4 inteiros A,B,C,D
  - Se as transacções são executadas por threads concorrentes é necessário adicionar primitivas de sincronização para assegurar os resultados correctos.

```
Transaçcão mvAtoB(int x)
is
  A= A-x;
  B=B+x;
End
```

```
Transaçcão mvCtoC(int x)
is
  C= C-x;
  D=D+x;
End
```

```
Transacção
sum All is
  x=0;
  x=x+A;
  x=x+B;
  x=x+C;
  x=x+D;
Print(x);
End;
```

Locking

- *lock* para escrita deve ser exclusivo, mas *lock* para leitura pode ser compartilhado.
- Quanto menor a granularidade do *lock* maior a chance de paralelismo, mas também maior é a chance de *deadlock.*
- *Strict two-phase locking*: a fase *shrinking* ocorre "instantaneamente" (previne *cascade aborts*).

Controle de concorrência (cont…)

- Uma das obter exclusão mútua na base de dados é criar um semáforo global.
    - Cada transacção executa um wait no semáforo antes de aceder qualquer variável, e uma operação de signal após o último acesso (forçava a execução em série) .
        - Desvantagem: como mvAtoB e mvCtoD acedem variáveis diferentes deveriam executar em paralelo.

Controle de concorrência (cont…)

- Controle de concorrência global

```
Transaçcão mvAtoB(int x) is
Wait(db-lock)
  A= A-x;
  B=B+x;
Signal(db-lock)
End
```

```
Transaçcão mvCtoC(int x) is
Wait(db-lock)
  C= C-x;
  D=D+x;
Signal(db-lock)
End
```

```
Transacção sumAll is
Wait(db-lock)
  x=0;
  x=x+A;
  x=x+B;
  x=x+C;
  x=x+D;
Signal(db-lock)
Print(x);
End;
```

Controle de concorrência (cont…)

- Associar o método da exclusão mútua do semáforo a cada variável da base de dados :
  - Aumenta o grau de concorrência
  - Assegura que as execuções sejam serializáveis.
  - Enquanto deixa que cada transacção faça o lock apenas ás variáveis que ela acessa.

Controle de concorrência (cont…)

- É a forma mais simples e mais optimizada porque:
- A transacção sumAll é bloqueada quando existe um update a uma das variáveis.
- mvAtoB e mvCtoD podem ser executadas em paralelo (usam semáforos diferentes).

```
Transaçcão mvAtoB(int x) is
Wait(lock-A);wait(lock-B)
  A= A-x;
  B=B+x;
Signal(lock-A); Signal(lock-B);

End
```

```
Transaçcão mvCtoC(int x) is
Wait(lock-C);wait(lock-D)
 C= C-x;
  D=D+x;
Signal(lock-C); Signal(lock-D);
End
```

```
Transacção sumAll is
Wait(lock-A);wait(lock-B)
Wait(lock-C);wait(lock-D);
  x=0;
  x=x+A;
  x=x+B;
  x=x+C;
  x=x+D;
Signal(lock-A);
Signal(lock-B);
Signal(lock-C);
Signal(lock-D);
Print(x);
End;
```

One-Copy Seriability

- “sistemas com equivalência a uma só copia”

- cada execução concorrente deve
  equivaler a uma execução sequencial.
- Garante-se a consistência porque se propagam os resultados entre as várias cópias.
- One-Copy equivalence :o conjunto de réplicas de memória deve-se comportar como se de uma se tratasse no que diz respeito a acesso a dados.
- Combinar One-Copy equivalence e o critério de seriabilidade dá lugar ao One-Copy Seriability.

One-Copy Seriability (cont…)

- Se todas as cópias de memória se encontram acessíveis e disponíveis podem-se sincronizar de forma a garantir que o critério de consistência não é violado.
- Para garantir o.c. seriability em várias partições da rede é necessário:
    - Garantir os updates no máximo em uma partição, mas nunca em partições concorrentes.