Transações Distribuídas

Transação

- Sequência de instruções claramente delimitada cuja execução aparenta ser instantânea e indivisível
- Tipicamente limitada através de instruções específicas:
  - “begin-transaction”
  - “end-transaction/commit/abort”

Transação

```
transferir(contaA, contaB, montante)
{
  beginTransaction;
  bancoA.lerSaldo(contaA, saldoA);
  bancoB.lerSaldo(contaB, saldoB);
  bancoA.atualizarSaldo(contaA, saldoA-montante);
  bancoB.atualizarSaldo(contaB, saldoB+montante);
  commit;
}
```

Atomicidade

- A Transação ou se executa na totalidade, ou não produz qualquer efeito, tal como se nunca tivesse executado
- Uma Transação que se executa e tem efeito diz-se *confirmada* (committed)
- Uma Transação que não produz efeitos diz-se que *abortou*

Em que casos uma transação aborta?

- A pedido do programa
  - Quando, perante certa condição, o programa chama *abort*

ou

- Devido a problemas que possam ocorrer durante a execução da transação:
  - Ocorreu uma falha (da máquina, da base de dados, etc) após a transação iniciar
  - [e mais…]

Atomicidade

```
transferir(contaA, contaB, montante)
{
  beginTransaction;
  bancoA.lerSaldo(contaA, saldoA);
  bancoB.lerSaldo(contaB, saldoB);
  if (saldoA < montante) {
  abort;
  } else {
       bancoA.atualizarSaldo(contaA, saldoA-montante);
       bancoB.atualizarSaldo(contaB, saldoB+montante);
       commit;
  }
}
```

Isolamento

- Define qual é o comportamento esperado do sistema quando são executadas de forma concorrente transacções que acedem aos mesmos dados.

Isolamento

- Têm sido propostas diferentes formas de isolamento:
  - Serializabilidade Estrita
  - Serializabilidade
  - Isolamento Instantâneo (“snapshot isolation”)
  - Coerência transaccional causal
  - …

- Nesta cadeira não temos a oportunidade de estudar todos estes modelos em pormenor.

Serializabilidade

- O resultado da execução concorrente de um conjunto de transacções deve ser equivalente ao resultado da execução destas transacções, uma de cada vez, por uma ordem série

… O que é que isto nos lembra?

Transacções e Bases de Dados

- As transacções podem ser usadas como alternativa ao controlo de concorrência através de objectos de sincronização como os trincos, semáforos, etc.

- Uma das mais importantes aplicações onde a noção de transação é mais usada é nas bases de dados

Transacções e Bases de Dados

- No contexto das bases de dados as transacções oferecem duas propriedades adicionais:
- *Coerência:* a execução de uma transação não resulta na violação dos invariantes que caracterizam o modelo de dados
- *Durabilidade:* os resultados da transação ficam registados de forma persistente na base de dados
- Assim as bases de dados oferecem um conjunto de propriedades designadas por ACID: **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

Como garantir a Atomicidade e o Isolamento em base de dados?

Nos slides seguintes, abordamos algumas (entre muitas) destas técnicas, de forma apenas superficial

Atomicidade: utilização do “redo log”

- Os resultados da transação não são aplicados durante a execução, mas apenas quando a transação confirma
- Antes de aplicar os resultados, estes são escritos para um “log” persistente, conjuntamente com a informaçao de que a transação foi confirmada
- Os resultados começam a ser aplicados apenas depois da persistência do “log” ter sido garantida
- Se ocorrer uma falha que interrompa a aplicação dos resultados, quando a máquina recupera, consulta o “log” para acabar o trabalho que foi interromopido

Isolamento: strict 2-phase locking

- Método pessimista de controlo de concorrência, baseado na obtenção automática de trincos sobre os dados, à medida que a transação executa
- Cada objecto tem associados trincos de leitura e trincos de escrita
- Quando uma transação acede a um objecto para **leitura** pela primeira vez tenta obter um **trinco de leitura** nesse objecto
- Quando uma transação acede a um objecto para **escrita** pela primeira vez tenta obter um **trinco de escrita** nesse objecto

Isolamento: strict 2-phase locking

- Este mecanismo pode gerar situações de interbloqueio
  - Existem algoritmos para detetar o interbloqueio (o livro descreve alguns destes mecanismos)
  - Em muitos casos o interbloqueio é resolvido simplemente através de timeouts: uma transação que esteja bloqueada num trinco há demasiado tempo aborta, libertando os trincos que já possui

Isolamento: strict 2-phase locking

- É também possível prevenir o interbloqueio, atribuindo “prioridades” às transacões:
  - Um transação menos prioritária bloqueia-se se outra transação mais prioritária detiver o trinco
  - Uma transação mais prioritária que pretenda um trinco, já detido por uma transação menos prioritária, força esta última a abortar para não ter de se bloquear
  - Isto evita o interbloqueio mas pode gerar situações em que transacções são abortadas desnecessariamente
- Nota: muitas vezes a prioridade é uma estampilha temporal (física ou lógica), em que a transação mais “antiga” tem mais prioridade.

Em que casos uma transação aborta?

- A pedido do programa
  - Quando, perante certa condição, o programa chama *abort*

ou

- Devido a problemas que possam ocorrer durante a execução da transação:
  - Ocorreu uma falha (da máquina, da base de dados, etc) após a transação iniciar
  - Em sistemas com strict 2-phase locking: um trinco (lock) obtido pela transação foi roubado por outra transação para prevenir algum *deadlock*
  - [e mais…]

Transacções distribuídas

- Aplicam-se em sistemas que possuem dados distribuídos por várias máquinas.
  - Por exemplo, transferir fundos do bancoA para o bancoB, em que cada banco tem a sua base de dados.

Atomicidade e confirmação atómica

- Para garantir a atomicidade da Transação distribuída é necessário garantir que os resultados se aplicam em todos os nós ou não se aplicam em nenhum dos nós
- Isto involve a coordenação entre os nós que participam na Transação
- Este problema designa-se pelo problema da “confirmação atómica distribuída”
- A solução mais simples para este problema é conhecida para confirmação (atómica distribuída) em 2-fases (“*2-phase commit*”)
  - Não confundir com o “*2-phase locking*”, que é independente disto.

Confirmação em duas fases

- Um dos participantes é eleito como coordenador
- O coordenador envia uma mensagem denominada “prepare” para todos os participantes.
- Se o participante pode confirmar a transação (a transação executou sem abortar, e o “redo log” já está guardado em memória persistente) o participante acrescenta ao “log” um registo a indicar que vota favoravelmente, e envia “ok” ao coordenador.
- Se o participante não puder confimar a Transação, o participante acrescenta ao “log” um registo a indicar que vota desfavoravelmente, e envia “not-ok” ao coordenador.

Confirmação em duas fases

- Se o coordenador receber ”ok” de todos os participante, decide confirmar a Transação:
  - Acrescenta ao seu “log” uma entrada indicado que a Transação foi confirmada
  - Avisa os outros participantes deste facto (estes também registam o resultado no seu próprio “log”).
- Se o coordenador receber ”not-ok” de pelo menos um participante decide abortar a Transação:
  - Acrescenta ao seu “log” uma entrada indicado que a Transação foi abortada
  - Avisa os outros participantes deste facto (estes também registam o resultado no seu próprio “log”)

Confirmação em duas fases

- Se um dos participantes não responder ao coordenador, este pode tentar contacar o participante de novo ou, se suspeitar que este participante falhou, abortar a Transação
- Neste caso a falha do participante é equivalente a um “not-ok”.

Confirmação em duas fases

- Este algoritmo é bloqueante se o coordenador falha:
  - Um participante, depois de responder “ok” já não pode mudar de opinião e tem de esperar pela decisão do coordenador
    - Os objectos necessitam de permanecer protegidos pelos trincos até a transação terminar.
  - Se o coodenador falhar, esses objectos ficam bloqueados até o coordenador recuperar

Confirmação atómica tolerante a faltas

- Uma maneira de obter uma versão tolerante a faltas da confirmação atómica consiste em usar o **consenso** como módulo.
- Esta versão inicia-se da mesma forma que o algortimo de 2-phase commit
  - O coordenador despoleta o processo de verificação em todos os participantes
- Mas neste algoritmo, os participantes enviam o “ok” ou “not-ok” para todos os outros participantes (e não apenas para o coordenador)
- Todos os participantes coleccionam as respostas de todos os participantes e depois invocam o consenso (ver próximo slide)

Confirmação atómica tolerante a faltas

- Se um participante recebe ok de todos os participantes, inicia o consenso propondo “commit”
- Se um participante recebe pelo menos um “not-ok”, inicia o consenso propondo “abort”
- Se um participante suspeita que outro participante falhou, inicia o consenso propondo “abort”
- Todos os participantes confirmam ou abortam a transação de acordo com o resultado do consenso

Confirmação atómica tolerante a faltas

- Nota:
  - Se um participante propõe “abort” para consenso e outro participante a propõe “commit”, qualquer um destes resultados é um output válido do consenso.
- Questão:
  - Em que cenário podemos ter um participante a propor “abort” para consenso e outro participante a propor “commit”?
  - Porque é que neste caso, qualquer uma das decisões é válida?