# Políticas e consentimentos

**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

**Trabalho Prático 1**

Serviço Integrado de Pagamento em Contexto de Mobilidade

Desafio de Coordenação de Transações Distribuídas

Infra-Estruturas de Sistemas Distribuídos

Luis Campanela, n.º 8600

Bruno Costa, n.º 36868

Rodrigo Pina, n.º 44178

Docente: Prof. Luís Osório

**6 de Junho de 2021**

# Introdução

o Apresentação sumária do problema

o Apresentação sumária da abordagem

o Estrutura do documento

No âmbito do desafio de coordenação CMSP, foi, entre outros, abordado o problema das transacções entre os serviços cliente e os recursos (vectores), respeitando as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

Para garantia da propriedade Atomicidade, foi decidido criar um elemento Transaction Manager (TM) e, para garantia das propriedades Consistência e Isolamento, um elemento Lock Manager (TPLM). O acrónimo TPLM refere-se à característica Two-Phase do Lock Manager que será explicada mais à frente.

A propriedade Durabilidade, considera-se assegurada, apesar da implementação fazer o tratamento dos dados em memória. Na prática, para garantia da Durabilidade, era necessário introduzir um mecanismo de escrita dos dados em disco, o que não faz parte do âmbito deste trabalho.

A validação das propriedades é feita através da garantia do invariante, somatório do conteúdo dos diversos vectores de cada um dos serviços.

Nos capítulos seguintes, será feita a análise e discussão do problema e a descrição da abordagem de resolução.

# Estado do conhecimento e análise e discussão do problema

o Desenvolvimento de sistemas informáticos na base de elementos Service (SOA)

o Coordenação de transações distribuídas

o Descrição de quadros tecnológicos utilizados

Desenvolvimento de sistemas informáticos com elementos distribuídos e abordagem das dificuldades inerentes à concretização de diferentes mecanismos de interacção. No quadro das arquitecturas orientadas a serviços (SOA), a complexidade dos sistemas distribuídos pode ser reduzida com a modularidade e autonomia das entidades computacionais e, se estas tiverem um acomplamento fraco, contribuem para a robustez do sistema como um todo.

No entanto, as SOA colocam novos desafios em termos de coordenação de sistemas distribuídos, uma vez que aumentando o número de elementos componentes, aumenta a complexidade da sua coordenação.

# Demonstrador centrado na coordenação

* 1. o Descrição do demonstrador



*Fig. X Acesso de múltiplos clientes a um vector*



*Fig. X Acesso de múltiplos clientes a múltiplos vectores*

# Transaction Manager (TM)

Na abordagem inicial à questão da Atomicidade (Prática 1), como se tratava do acesso a um único vector, a garantia do invariante foi efectuada com o controlo do número de escritas que tinham existido no momento do cálculo do somatório. Fazendo o cálculo só quando havia um número de escritas par, ou seja quando não se verificava (nRequests % 2 != 0), garantia-se que já tinham havido as duas escritas (e, por isso, também as duas leituras) que compõem uma transacção atómica sobre o vector.

public static int sum() {

synchronized(lock) {

while(nRequests % 2 != 0) {

try {

lock.wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

return getSum();

}

}

Neste caso, em que há vários vectores sujeitos a operações de leitura e escrita, a estratégia terá que ser diferente porque a transacção tem que ser vista como atómica globalmente (para o conjunto de vectores) e pode provocar, indiferentemente, um número par ou ímpar de escritas em cada um dos vectores. Assim, desde logo, surgiu a necessidade de ter uma entidade computacional autónoma que agisse como moderadora, fazendo esse controlo. A essa entidade, chamou-se Transaction Manager, e considerou-se que teria que implementar um algoritmo do tipo two-phase commit (2PC), um dos algoritmos mais comuns, para garantir *atomic commitment* através de múltiplos nós [Gray, 1978].



*Fig. X Protocolo 2PC*

No 2PC, há uma fase inicial (*prepare*) antes da segunda fase (*commit*). Quando o cliente termina as operações e está preparado para finalizar a transacção, envia uma instrução de *commit* ao TM (coordinator). Este, começa por enviar uma mensagem de *prepare* a cada um dos nós (vectores), que respondem indicando se estão prontos para fazer *commit* da transacção. O Transaction Manager, finalmente, decide se envia a decisão de commit para todos os nós ou, caso algum dos nós não esteja preparado ou não responda, dá a indicação para abortar a transacção (roll-back).

*Transaction Manager na perspectiva do modelo X/OPEN: two-phase commit with presumed rollback.*

# Operações

getLocks()

unlock()

# Two-Phase Lock Manager

Garante a concorrência no acesso aos recursos.

Two-phase lock (2PL)

Os read locks (shared locks) ou write locks (exclusive locks).

Recebe os pedidos de locks dos clientes (getLocks) e atribui-os, se estiverem disponíveis. Mantém o registo dos locks atribuídos e também dos pedidos ainda não satisfeitos (locks pendentes).

Quando há a libertação de um lock num recurso, a lista de locks pendentes é consultada, para verificar se existem pedidos de lock para esse recurso. Se existir, o lock é atribuído e o cliente que o pediu é notificado.

# Operações

# Resource Managers (RM)

Os RM são responsáveis por receber as instruções dadas pelos clientes (leitura e escrita nos vectores) e executarem-nas quando o TM dá essa indicação, no processo de 2PC (two-phase commit).

# Verificação do invariante

# Clientes

Um cliente delimita um conjunto de operações que vai ser executado, tendo em conta as propriedades ACID. Um cliente tem que garantir que é o único que detém os locks para os elementos do vector em que pretende efectuar operações de read ou write.

O cliente inicia a transacção no TM (tx\_begin) e recebe um ID. Pede os locks ao LM (getLocks) e, assim que os obtém, inicia as operações nos vectores. Quando termina, dá indicação de commit para o TM (tx\_commit) e, após obter confirmação, dá indicação de libertação dos locks ao LM (unlock).

Se existir uma falha, o cliente pode dar uma indicação de tx\_rollback, ao TM.

Para garantir a consistência, o commit tem que ser feito antes de libertar os locks, estes têm que ser mantidos até que haja confirmação de realização das operações. A evolução de estado tem que ser sempre consistente.

# Interface entre componentes

A comunicação entre o TM e o LM pode ser vantajosa em termos de reforçar a tolerância a falhas. Por exemplo, caso a comunicação com um dos clientes com o TM falhe e este detecte essa falha ou receba uma indicação de um cliente para abortar a transacção, o TM pode informar o LM que libertará os recursos atribuídos à transacção daquele cliente. Caso contrário, o LM ficará com o espaço ocupado com os recursos da transacção.

# Callback

Por exemplo, quando um cliente falha, todos os serviços que tinham alguma forma de associação, podem libertar os recursos respectivos.

Alguns frameworks implementam funcionalidades, como callback automático no caso de um cliente cair. Pode fazer parte do runtime.

Invocação mútua?

Invocação circular?

Estamos a usar o Framework OSGi -> Callbacks para o JAX-WS, possibilidade de ter chamada assíncrona.

Usando serviços remotos baseados em JAX-WS, como é que se reflecte? A abstracção está na implementação do remoting. O modelo programático é o do OSGi, não é o do JAX-WS.

# Conclusões

* 1. o Resumo do que foi discutido e realizado
  2. o Dificuldades e aspetos a melhorar

Adequação ao cenário CMSP

# Bibliografia

[Gray, 1978] Jim N. Gray. Notes on data base operating systems. Springer, 1978.

[Kleppman, 2021] Martin Kleppman. Distributed Systems Notes. University of Cambridge, 2020/21.