

PADI-DSTM

Abstract

O PADI-DSTM é um sistema distribuído que permite gerir objectos que residem em memória e são partilhados por programas transaccionais que correm em máquinas diferentes.

1. Introdução

Este artigo elaborado no âmbito da disciplina de Plataformas para Aplicações Distribuídas na Internet, tem como objectivo fazer uma apresentação e descrição da solução escolhida para a implementação do projecto PADI-DSTM.

Inicialmente iremos abordar a nossa solução e compará-la com algumas alternativas que considerámos, apresentando uma vista global da sua arquitectura. De seguida falaremos das estruturas de dados e algoritmos escolhidos. Por fim, iremos abordar o trabalho a ser realizado posteriormente - tolerância a faltas - e iremos fazer uma pequena conclusão.

2 Solução

2.1 2PC vs S2PL vs Timestamps

Inicialmente considerámos seguir uma abordagem optimista. No entanto, como não sabemos qual é a relação existente entre o número de leituras e escritas, podemos admitir que existe igual número de leituras e escritas. Neste caso o número de conflitos existentes poderá ser grande.

Considerando que isto seria um potencial bottleneck de performance no nosso sistema, optámos por abandonar as soluções que usam timestamps ou two-phase-commit (2PC). Para além disto outro argumento contra o 2PC é a possibilidade das transacções poderem abortar (se surgir algum conflito) depois de já terem realizado algum trabalho, o que implicaria refazer-se esse trabalho.

Assim, escolhemos utilizar *Strict Two Phase Locking* ou *S2PL*

2.2 Replicação activa vs replicação passiva

Depois de escolhermos qual o protocolo a usar, deparámo-nos com a escolha entre replicação activa e passiva. Inicialmente considerámos seguir replicação activa com um protocolo de *Quorum Consensus*. No entanto, por esta precisar de três servidores (em vez dos dois necessários para a replicação passiva) e tendo também em conta que teríamos que enviar e receber mais mensagens do que as necessárias ao usar replicação passiva, optámos por esta última opção.

2.3 Arquitectura

3 Estruturas de Dados

3.1 Master

Esta classe é o gestor do sistema de memória distribuída e é responsável por armazenar os dados globais do sistema.

É importante salientar que a lista *Servidores* tem como finalidade garantir que no caso em que o servidor primário for substituído, o endereço registado é actualizado e os clientes continuam a conseguir aceder aos *PadInts* sem perturbações. Para além disto, os *TID* são atribuídos de forma sequencial e única.

De seguida apresenta-se a estrutura interna desta classe.

Variável	Descrição
TID	Último <i>Transaction ID</i> atribuído
Servidores	Estrutura que mapeia o identificador de cada servidor primário com o seu endereço

Table 1. Atributos da classe Master

3.2 Servidor

O conjunto das instâncias da classe Servidor representa a memória distribuída onde são armazenados os *PadInts*. De seguida apresentam-se os atributos desta classe.

Variável	Descrição
Pedidos	Lista de pedidos feitos ao servidor quando o servidor entra no modo <i>Freeze</i>
Réplica	Endereço do outro servidor
Valor original	Estrutura que mapeia <i>UIDs</i> em <i>PadInts</i>

Table 2. Atributos da classe Servidor

A localização de cada *PadInt* depende do número total de servidores primários, ou seja, é dada por:

$$UID \bmod N^{\circ} \text{ de Servidores}$$

3.3 PadInt

Esta classe representa o objecto gerido pelo *PADI-DSTM* que guarda um inteiro. Esta classe é composta por:

Variável	Descrição
UID	Identificador do inteiro que representa
Valor actual	Valor no momento actual da transacção
Valor original	Valor no início da transacção
Temporizador	Usado na detecção de deadlocks
Promoção	Referência para a próxima transacção a ser promovida
Leitores	Fila de transacções com locks de leitura atribuídos
Escritor	Transacção com lock de escrita atribuído
Leitores à espera	Fila de transacções com locks de leitura à espera
Escritores à espera	Fila de transacções com locks de escrita à espera

Table 3. Atributos da classe PadInt

3.4 Lock

Um *Lock* é apenas o *TID*.

3.5 Biblioteca

Esta é a classe que representa a biblioteca usada pelos clientes para comunicar com o sistema de memória distribuída. Esta classe tem a seguinte estrutura:

Variável	Descrição
Nº de Servidores	Número total de servidores primários existentes
TID	Transacção atribuída pelo <i>Master</i>
(UID, Servidor)	Lista de associações entre <i>UIDs</i> e o seu respectivo <i>Servidor</i>
Cache de Servidores	Estrutura que mapeia <i>UID</i> no <i>Servidor</i> em que o <i>PadInt</i> está armazenado

Table 4. Atributos da classe Biblioteca

4 Algoritmos propostos

4.1 Locking

4.2 Commit

Quando é invocado o método *TxCommit*, são percorridos os pares (*UID*, *servidor*) na estrutura descrita na Tabela 4, enviando a cada servidor primário um pedido para que faça commit de todos os *PadInts* que foram acedidos para leitura ou escrita durante o decorrer da transacção atual.

Ao receber o pedido de commit, o servidor primário percorre a lista de identificadores de *PadInts* envolvidos no commit e, para cada um deles, verifica se o *TID* recebido como argumento pertence a alguma das seguintes variáveis da classe *PadInt* ilustradas na Tabela 3:

- Leitores;
- Escritor;
- Leitores à espera;
- Escritores à espera;
- Promoção.

Se *TID* que identifica a transacção estiver contido nalguma das variáveis acima descritas, este é removido dessa variável. É também criado e guardado um par (*TID*, *Boolean*¹). Este par é usado se eventualmente o servidor primário entrar no estado de *Fail*, o secundário assumir o papel de primário e a *Lib* lhe pedir para responder a um commit/abort de uma transacção terminada, mas à qual o primário (agora em modo *Fail*), nunca chegou a enviar uma mensagem de *ack*.

Se o *TID* não pertence a nenhuma das variáveis anteriormente descritas e o *TID* da transacção a ser tratada é o mesmo que foi registado da última vez que se guardou um par (*tid*, valor final atribuído a um *PadInt*) no final da última transacção, então o valor registado no par como resultado final da transacção é devolvido como retorno à *lib* e o par é apagado.

¹O boolean indica se o commit teve sucesso ou não

4.3 Detecção de deadlocks

Para a detecção de deadlocks usámos um temporizador para cada *PadInt* existente no servidor. Este temporizador é activado quando se coloca algum pedido em espera, seja para promoção de lock ou para obter lock de escrita/leitura. Quando o temporizador expira é feito abort da transação² (ou transacções no caso de existirem locks de leitura atribuídos) que possuía o lock e escolhe-se um pedido dos que estão em espera para ser executado pela seguinte ordem: pedido de promoção, pedido de lock de escrita e finalmente pedido de lock de leitura.

4.4 Recuperação de aborts

5 Tolerância a Faltas

5.1 Fail

5.2 Freeze

6 Conclusão

²Admitindo que existem mais transacções à espera de serem promovidas ou para lerem/escreverem