Lista 5 - Sistemas Distribuídos

Thiago Guimarães - DRE: 118053123

2021.1

1

Um sistema transacional deve oferecer ACID para garantir previsibilidade, corretude e um sistema de fácil utilização e manutenção.

1.1 Atomicidade

Significa que cada transação é atômica, ou seja, ou é realizada totalmente ou é abortada. Se abortada, não ocorrem alterações no estado global.

1.2 Consistência

Cada transação mantém propriedades do estado global. Um exemplo seria a soma total dos saldos de uma transferência entre 2 contas.

1.3 Isolamento

Cada transação roda de forma isolada. Ou seja, cada transação roda inteiramente (ou não roda), até que outra possa então rodar.

1.4 Durabilidade

Ao commitar uma transação, o sistema transiciona para um novo estado global que é persistido, independente de eventos externos.

2

O que pode acontecer com esta implementação é uma falha no caso quando acontece outra transferência entre c2 e c1, podendo ocorrer um deadlock. Se a transf(c1, c2, v) pegar o lock de c1 e a transf(c2, c1, v) pegar o lock de c2, os dois ficam esperando a liberação do lock da outra conta. Para corrigir a implementação, poderíamos ordenar as contas de uma forma arbitrária e consistente e obter os locks nesta ordem, impedindo a ocorrência de deadlocks.

3

A técnica de 2PL tem como objetivo controlar a concorrência, visando possibilitar a atomicidade e isolamento das transações do sistema. Para fazer isso, a técnica cria 2 tipos de locks para cada objeto, um de leitura e um de escrita. Read locks travam apenas write locks novos de surgirem, enquanto write locks impedem qualquer outro lock, seja write ou read. Ademais, o 2PL utiliza 2 fases para cada uma das suas transações: na primeira fase (expansion), os locks necessários são adquiridos. Na segunda fase (shrinking), os locks utilizados são liberados.

4

Um exemplo seriam 2 transferências bancárias ocorrendo de forma simultânea - sendo cada transação composta por retirada e depósito. No caso em que ocorra uma transação de transferência entre c1 e c2, enviando um valor v1 para c2 e uma outra enviando um valor v2 para c1. Se na primeira transferência for adquirido o lock de c1 e na sequência a segunda transferência adquirir o lock de c2, como mencionado na questão 2. Assim, as duas transferências aguardarão os locks, impedindo que os dois votem, colocando o coordenador em wait eterno, formalizando um deadlock.

5

5.1

Quando um processo participante falha no INIT, significa que este não chegou a votar nem em abort e nem em commit. O coordenador terá seu temporizador estourado enquanto o processo participante não responde, enviando assim uma mensagem de abort.

5.2

O processo, ao falhar em READY, já votou em commit. O que acontece para recuperação neste caso é que quando o processo se recuperar, verificará o log e voltará para seu último estado. Ao voltar, perguntará o que aconteceu com a transação na qual ele falhou anteriormente.

5.3

Quando o coordenador falha em WAIT, podemos ter qualquer combinação de votos, dado que a falha ocorreu na espera do coordenador. Os processos que votarem em commit ficarão aguardado o coordenador voltar da sua falha e retomar o processo. Assim, fica claro a natureza bloqueante no coordenador do 2PC.

6

6.1 read-write

Este tipo de conflito acontece quando temos um ou mais processos lendo um dado em uma réplica e outro processo escrevendo neste dado em outra réplica.

6.2 write-write

Este tipo de conflito acontece quando dois ou mais processos em réplicas diferentes escrevem no mesmo dado.

7

7.1

Respeita o modelo de consistência sequencial.

A ordenação poderia ser: R(x,0), W(x,1) e R(x,1)

7.2

Não respeita o modelo de consistência sequencial.

Não há ordenação possível, dado que o valor de X só pode ser 1 após o W(x,1), de forma que a ordem R(x, 1), R(x, 0) é impossível.

7.3

Respeita o modelo de consistência sequencial.

A ordenação poderia ser: W(x,1) R(x,1), W(x,2) e R(x,2)

7.4

Respeita o modelo de consistência sequencial.

A ordenação poderia ser: W(x,2), R(x,2), W(x,1) e R(x,1)

7.5

Não respeita o modelo de consistência sequencial.

Nenhuma ordenação é possível, dado que os processos P3 e P4 geram uma contradição entre si na ordem, o que impossibilita a consistência temporal.

7.6

Respeita o modelo de consistência sequencial.

A ordenação poderia ser: R(y,0), R(x,0), W(x,1), R(x,1), R(y,0), R(x,1), R(y,0), W(y,1), R(y,1), R(x,1)

7.7

Não respeita o modelo de consistência sequencial.

Novamente, nenhuma ordenação é possível, pois não há combinação que siga a lógica do modelo de consistência sequencial.

8

8.1

Respeita o modelo de consistência causal.

A ordenação poderia ser: R(x,0), W(x,1) e R(x,1)

8.2

Não respeita o modelo de consistência causal.

Não há ordenação possível, dado que o valor de X só pode ser 1 após o W(x,1), de forma que a ordem R(x, 1), R(x, 0) é impossível.

8.3

Respeita o modelo de consistência causal.

A ordenação poderia ser: W(x,1) R(x,1), W(x,2) e R(x,2)

8.4

Respeita o modelo de consistência causal.

A ordenação poderia ser: W(x,2), R(x,2), W(x,1) e R(x,1)

8.5

Respeita o modelo de consistência causal.

A ordenação poderia ser algo do tipo: CONCORRENTEMENTE(W(x, 1), W(x, 2)), CONCORRENTEMENTE((R(x,1), R(x,2))

8.6

Respeita o modelo de consistência causal.

A ordenação poderia ser: R(y,0), R(x,0), W(x,1), R(x,1), R(y,0), R(x,1), R(y,0), W(y,1), R(y,1), R(x,1)

8.7

Não respeita o modelo de consistência sequencial.

Novamente, nenhuma ordenação é possível, pois não há combinação que siga a lógica do modelo de consistência sequencial.

9

Availability é referente a fração do tempo em que o sistema está operacional, enquanto Reliability se refere ao intervalo de tempo até que ocorra uma falha. O Reliability tem relação direta com a métrica MTTF (mean time to failure), enquanto o Availability tem com a métrica A = MTTF/(MTTF+MTTR), sendo MTTR o tempo médio de reparo.

Um exemplo é um HD que demora, em média, 2 anos para ocorrer uma falha e demora, em média, 5 dias para ser consertado.

Assim, teríamos: MTTF = 730 dias MTTR = 5 dias A = MTTF/(MTTF+MTTR) = 730/735 = 0.99319727891

10

10.1

Reliability Recovery = MTTF/(MTTF + MTTR)

Reliability Recovery = $21915/(21915 + 32) = 0.9985$

$0.9999 = 1 - (1 - 0.9985)^k$

$0.9999 = 1 - (1 - 0.9985)^k$

$k = ln(0.0001)/ln(0.0015) = 1.1465$

Aproximando, teríamos que ter pelo menos 2 componentes para ter uma Reliability Recovery acima de 0.9985. Para ter uma mais próxima, teríamos que ter 1 componente, porém teríamos uma Reliability Recovery menor do que 0.9985.

10.2

$q = 1 - p^k$

Com k = 2:

$q = 1 - (0.9985)^2$

$q = 1 - 0.99700225$

$q = 0.003$

Teríamos então 0,3%

11

Assumindo a seguinte configuração e nomenclaturas:

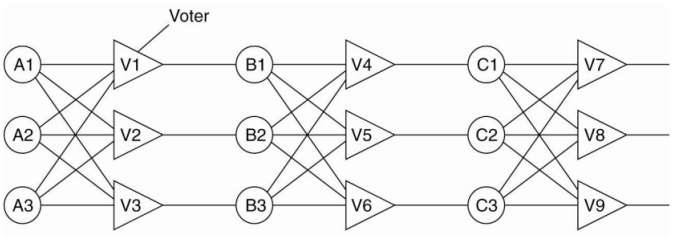

Figura 1: Configuração

11.1

O funcionamento do sistema será afetado a depender da configuração do sistema. Se um componente e um votador falham em linhas diferentes porém na mesma coluna, o sistema não funcionará.

11.2

Se dois votadores na mesma coluna falharem, o sistema não funcionará. Isto se dá pois o dado chegará em B3 mas não chegará nem em B2 nem em B1, só recebendo a informação de B1.

11.3

O sistema funcionará. Isto pois mesmo com dois votadores falhando, como estão com o mesmo valor de saída, este valor não será perdido e o sistema funcionará tranquilamente.

12

Crash failures consistem em erros em que o sistema para de funcionar, de modo que logo antes ele estava funcionando corretamente e ele simplesmente parou. Enquanto isso, falhas bizantinas possuem um comportamento mais randômico, podendo enviar qualquer mensagem ou realizar qualquer ação a qualquer momento, sendo assim muito mais difícil de lidar com este tipo de falha.

13

Assumindo que o coordenador seja o participante em falha bizantina, este enviará dados diferentes para os outros participantes, de modo que estes processos enviarão também estes dados recebidos para os outros dois. Assim, como cada participante recebeu um valor diferente, nunca haverá uma maioria, implicando que nunca haverá um consenso.