Universidade do Minho

Departamento de Informática

Tolerância a Faltas

Guião Laboratorial 2
Replicação de Quórums

Ana Rita Peixoto PG46988
André Gonçalves PG46525
Henrique Neto PG47238
Márcia Teixeira A80943

18 de março de 2022

Necessidade de *Locks* para a operação *CAS*

O problema apresentado surge no contexto da replicação de dados recorrendo a várias réplicas, e visa encontrar um mecanismo que permita manter a consistência no sistema, ou seja, cada nodo deve ter a mesma visão válida dos dados num dado momento.

No contexto deste problema, os dados são armazenados em vários nodos, onde cada um possui um dicionário que mapeia *key* → (*value*, *timestamp*). Esta estrutura vai ser atualizada à medida que os nodos recebem pedidos de *Write* ou de *CAS*, e os seus valores podem ser acedidos com pedidos de *Read*. Ora, dado que se trata de uma estrutura mutável e que existem várias réplicas a serem acedidas por vários clientes, existem problemas que podem surgir devido à concorrência associada aos vários pedidos.

O primeiro problema surge quando ocorrem múltiplas leituras a um valor durante a sua escrita. Devido às latências existentes nas mensagens, é possível que durante uma escrita os quórums de leitura não tenham convergido no mesmo valor, sendo que alguns já apresentam o valor novo, enquanto que os restantes ainda apresentam o antigo. Desta forma o sistema não responderá de forma determinística aos pedidos de leitura, podendo este apresentar um de dois valores de forma completamente aleatória durante este período.

O segundo problema surge quando fazemos várias escritas concorrentemente. Precedendo a uma escrita, existe sempre um período de reconhecimento, em que o processo calcula a versão atual (*timestamp*), para desta forma poder atualizar o chave com o valor e a versão nova (*timestamp+1*). Quando temos várias escritas em simultâneo, é possível que os vários processos de escrita obtenham o mesmo valor para a versão atual, o que resulta na sua tentativa de escrever cada um dos seus valores com a mesma versão. Sendo que as mensagens da mesma escrita tem latências diferentes, é provável que cada nó receberá as mensagens por ordem diferente e desta forma escreverá os valores por ordens diferentes, resultando assim em inconsistências nos valores entre os nós e por sua vez chaves com valores não determinísticos. Uma abordagem para evitar tais inconsistências poderia passar por expandir a ordem global introduzida pelas versões com base em fatores adicionais ao contexto, como priorizar um processo em vez de outro, ou com base no objeto que está a ser escrito. No entanto, no contexto do protocolo *lin-kv* em que o *maelstrom* opera, não existem dados suficientes para optar por abordagens análogas às referidas anteriormente.

Assim sendo, foi necessário tratar o problema de forma diferente, recorrendo a mecanismos de controlo de concorrência, salientando a exclusão mútua. Desta forma, foram implementados *locks* que foram descritos num objeto (*Lock*). Este mecanismo permite a cada nodo anunciar aos restantes a intenção de trabalhar com uma determinada chave, impedindo acessos concorrentes e, consequentemente, inconsistências no sistema distribuído.

O mecanismo de bloqueio implementado segue a lógica de um *Read Write Lock*, na medida em que permite a obtenção de vários *locks* de leitura em simultâneo e apenas permite a obtenção de um *lock* de escrita de cada vez. Desta forma, uma *key* pode ser acedida por múltiplos nodos no caso da leitura (*Read*), mas apenas pode ser acedida por um nodo de cada vez quando se trata da escrita (*Write* ou *CAS*).

A partir do mecanismo explicitado anteriormente são solucionados problemas de concorrência que se poderiam refletir em inconsistências para as operações *Read*, *Write* e *CAS*. Cada operação passou assim a englobar duas fases diferentes de processamento (1 - Pedir acesso a um quórum para ler os *value* e *timestamp* associados a uma chave; 2 - efectuar a operação e libertar os *locks* associados à chave).

Na figura 1 é possível observar o comportamento do algoritmo sem considerar a utilização de mecanismos de controlo de concorrência. Como podemos observar, é recebido um pedido *CAS* no nodo n2 e, de seguida, o nodo n0 recebe um pedido de *write*. Como não foram considerados *locks* neste exemplo, tanto a operação *Write* como *CAS* podem prosseguir a sua execução, começando de forma análoga e iniciando-se com um pedido "do_read"para o quórum respetivo. Esta mensagem é respondida com uma do tipo "read_done"que informa o nodo do *value* e da respetiva versão da *key*. Como podemos observar, as duas operações pretendem manipular o *value* da mesma *key*, e tanto o *Write* (mensagens delimitadas a cor de laranja) como o *CAS* (mensagens delimitadas a verde) recebem a mesma resposta, indicando-lhes a

mesma versão do registo. Este acontecimento é problemático na medida em que irá permitir as escritas para as duas operações de forma inconsistente, visto que o *Write* irá escrever o par (*value*: 2, *timestamp*: 4) e o *CAS* irá escrever o par (*value*: 1, *timestamp*: 4) nos seus quórums.

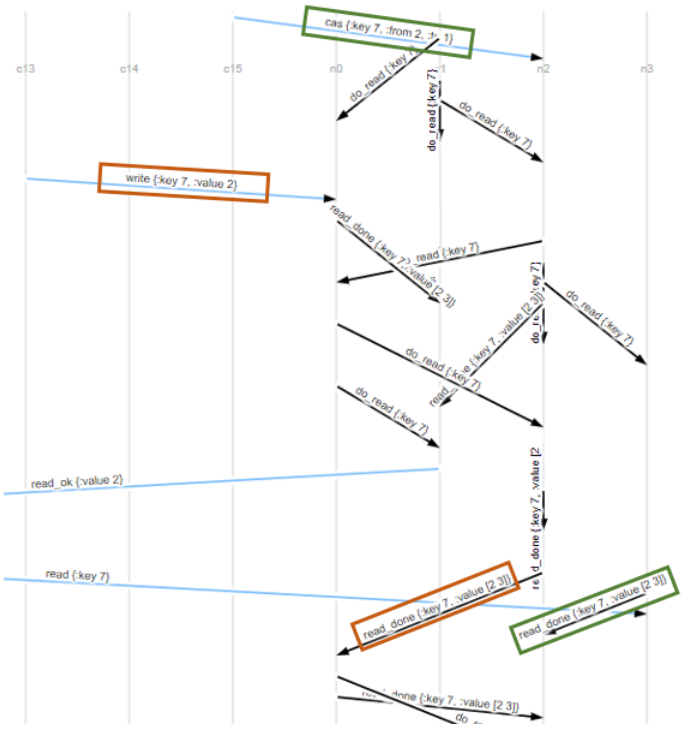

Figura 1: Exemplo da não utilização de *locks* para a operação *CAS*

Consequentemente, os acessos posteriores a esta *key*, seja por operações *read*, *write* ou *CAS*, vão originar resultados incoerentes, dado que para a mesma versão existem diferentes *value*, dependendo do nodo em questão. Este acontecimento é visível na figura 2, em que conseguimos observar que o pedido de *CAS* recebeu mensagens (delimitadas a azul) provenientes de diferentes nodos, que indicam *values* diferentes para a mesma versão (*timestamp*), que ocorreu devido ao exemplo da figura 1.

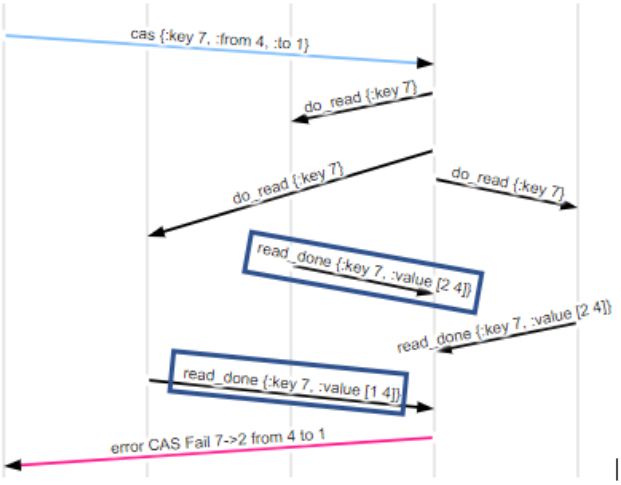

Figura 2: Consequência da não utilização de *locks* na operação *CAS*