Concorrência e Paralelismo 2016-17 - HW 3 - Hashmap concurrente

Aluno: Gonçalo Sousa Mendes, n.º 42082, Turno p.3
Prof: João Lourenço
Dept. de Informática
FCT/NOVA
24 de Dezembro de 2016

1 Introdução

Neste trabalho era necessário implementar e avaliar várias estrategias de *loocking* com diferentes granularidades.

2 Implementação: Validação e Estratégias usadas

Neste fase foi necessário descobrir quais as invariantes do *hashmap* e da aplicação. Por base, verificava-se se todos os *id's* existentes no *set* da *main* também existiam no *hashmap*, foi adiciono o inverso, isto é, se todos os *id's* existentes no *hashmap* também existiam no *set* (para isso, foi implementado um iterador no *hashmap*). Umas das propriedades importante deste tipo de estrutura de dados é a individualidade de cada elemento na estrutura, isto é, não existem repetições de chaves, pelo que foi adicionado uma verificação para esta invariante. Para se verificar a consistência do *hashmap* as operações realizadas sobre esta estrutura eram também feitas num *set* que se encontrava na *main*, pela classe *worker*, no entanto, aqui encontrava-se um problema, pois entre o acesso ao *hashmap* e ao *set*, existia a possibilidade de uma outra *thread* realizar operações sobre o *set*. Pode-se facilmente imaginar uma situação de remoção, onde uma *thread* faz a remoção no *hashmap*, entretanto outra *thread* faz uma inserção com o mesmo *id* na tabela, que aceita por não existir esta chave, mas quando tenta fazer a inserção no *set*, este retorna falso por ainda ter lá este elemento. Face a este situação foi adicionada outra verificação, sempre que se faz uma operação sobre o *set*, testa-se o sucesso da mesma, caso falhe, estamos perante uma inconsistência. É tambem verificado se o tamanho do *hashmap* é igual ao do *set*.
Foram usadas quatro estrategias de *locking* duas de granularidade grande e duas de média ao nível da lista de colisões. Para o primeiro caso usou-se o *synchronized* do Java e um *Lock* global do tipo *read-write*. Para a Granularidade média foram colocados *Locks* simples e depois do tipo *read-write* ao nível das listas de colisões, identificados pela *hash's*. Para esta granularidade foi adicionado um vector ao *hashmap* do mesmo tamanho da estrutura, onde cada posição mapeava um *lock*, onde a sua posição dada pelo *hash*.

3 Avaliação e conclusão

Para avaliar os resultados foram usadas 2 medidas, o número de operações por segundo e número de repetições de elementos nas tabelas. Cada valor obtido representa a média de 3 testes, cada um com a duração de 5 segundos, com 4 e 8 *threads*. Observou-se um grande aumento no número de operações por segundo entre as diferentes granularidades, mais do dobro entre a implementação do *synchronized* e a implementação do *Lock* global simples. A grande surpresa nos resultados aparece na diferença entre o *synchronized* e o *Lock* global do tipo *read-write*, onde era esperado um aumento no número de operações, acontecendo o oposto, uma grande redução no número de operações. O mesmo comportamento é observado nas implementações de granularidade média, mas menos evidente. Estes resultados podem dever-se ao grande *overhead* necessário para o controlo dos *locks* do tipo *read-write*.
Este trabalhou permitiu fazer uma reflexão entre as dificuldades de implementar diferentes soluções, com diferentes granularidades e estratégias de *locking* e suas eficiências, que nem sempre são o esperado. Face a estes resultados obtidos conclui-se também a importância vital de testar várias soluções antes de se optar por uma.