Concurrency and Parallelism 2016-17 - Project

Alunos:
João Almeida, n.º 42009
Gonçalo Sousa Mendes, n.º 42082
Grupo 1, Turno p.3

Prof: João Lourenço
Dept. de Informática
FCT/NOVA

1 de Dezembro de 2016

1 Introdução

O projeto visa a simulação de um repositório de artigos científicos. A tarefa proposta baseia-se na criação de métodos que permitem a correta paralelização de operações sobre as estruturas de dados e respectivos métodos de verificação de consistência . Nestas estruturas de dados encontram-se armazenados artigos e os respectivos identificadores, os autores e os seus artigos e as keywords e em que artigos estas são referenciadas. Para armazenar estas informações são utilizadas 3 HashTables e cada operação é reflectida nas 3 tabelas.

2 Abordagem

A solução implementada utiliza como base locks do tipo ReentrantReadWriteLock [1] [2] [3]. Para as operações de inserção e remoção de dados são utilizados os Write Locks desta biblioteca e para a operação de leitura os Read Locks. Desta forma foram criadas três estruturas adicionais à classe de Repository: um Map<Integer, ReentrantReadWriteLock> para alojar os locks relativos aos acessos à tabela de ids de artigos e dois Map<String, ReentrantReadWriteLock> para que se possam bloquear entradas nas tabelas de autores e keywords. O Map trará uma maior simplicidade na pesquisa de forma a que possa ser bloqueado o acesso a um elemento em particular da tabela respectiva. A lógica desta solução passa por bloquear acessos simultâneos a operações que efectuem alterações a entradas das tabelas com a mesma Hash, ou seja, existe um bloqueio à lista de colisões respectiva à entrada. Estes mecanismos foram todos implementados na classe de Repositório de forma a bloquear os acessos às Hash Tables.

Em todas as operações, a lista de colisões relativa ao artigo é bloqueada no início da operação. Este lock encontra-se numa estrutura e referencia o artigo pelo seu identificador, assim duas operações que incidam sobre a mesma lista de colisões não serão executadas em simultâneo. De seguida os recursos de keywords e de autores são iterados e bloqueados individualmente, são efectuadas as operações necessárias à estrutura e o recurso é libertado de seguida. Desta forma, são evitados deadlocks, pois os recursos são requisitados individualmente e são libertados antes de haver um pedido de outro recurso [4]. No final da operação o lock da lista de colisões do artigo é libertado.
Por exemplo, no caso da inserção de um artigo A, que é constituído por 3 autores e 3 keywords. A lista de colisões relativa ao artigo A é bloqueada no início da operação, os autores e keywords são iterados e para cada entrada é efectuado um bloqueio da lista de colisões respectiva, o artigo é adicionado na Hash Table e o recurso será libertado antes da seguinte iteração. No final das iterações o recurso do artigo é libertado podendo Threads

que incidam sobre a mesma lista de colisões operar.
Os métodos de inserção e remoção de artigos utilizam Write Locks na sua implementação enquanto que o método que retorna um artigo utiliza um Read Lock, desta forma, operações de escrita que efectuem alterações sobre a mesma lista de colisões não poderão operar em simultâneo, no entanto, será permitido em operações de leitura.

3 Validação

O método de validação consiste na verificação da consistência entre as três estruturas. Quando se efectua uma operação de inserção ou remoção, o resultado terá de ser reflectido nas três estruturas, logo, a validação passa por verificar se as estruturas se encontram num estado coerente. A validação é feita em três passos distintos, um que verifica a coerência dos artigos, outro para os autores e o ultimo para as keywords.
O primeiro irá verificar se cada artigo é referenciado nas entradas dos seus autores na tabela de autores e se é referenciado nas entradas das suas keywords. O segundo irá iterar todas as entradas da tabela de autores e irá verificar se todos artigos da lista de cada autor existem na tabela de artigos e se todas as keywords existem na tabela de keywords. O terceiro irá iterar as entradas da tabela de keywords e irá verificar se estão são referenciadas na lista de artigos na tabela de artigos e na tabela de autores. Se esta validação tiver sucesso, então as estruturas encontram-se num estado consistente, desta forma, a execução do programa continuará. A chamada ao método de validação é efectuada de 5 em 5 segundos, onde todos os Threads interrompem a sua execução para que se possa confirmar que as estruturas se encontram consistentes, no entanto, este teste poderá ser efectuado em qualquer altura. Se o método executar numa altura em que está a ser executada uma operação sobre a tabela a respectiva entrada estará bloqueada e quando a entrada desbloquear as estruturas estarão num estado consistente.

4 Avaliação

Para testar, foram efectuados vários testes, em três implementações diferentes num computador com quatro processadores lógicos. A primeira, já abordada, a segunda é uma implementação com um Read Write Lock que bloqueia todos os acessos ao repositório e a terceira com o Syncronized [5] [6] nos métodos. Nestes testes foram variados o número de Threads, o tamanho da lista para o get (nfindlist), o tamanho do dicionário a usar (nkeys) e o número de autores e keywords (que variam em conjunto). Os restantes parâmetros foram fixados, com o tempo de execução a 20 segundos, 15% para inserção e remoção e 70% para as operações de leitura. A avaliação foi feita pelo número de operações por segundo. Na Figure 1 estão representados os resultados de um teste utilizando um dicionário pequeno, mas uma nfindlist de tamanho elevado. Conseguimos perceber que as soluções se aproximam bastante em termos de número de operações por segundo, pois existe um elevado número de colisões, fazendo com que seja equiparável o bloqueio do método ao bloqueio da lista de colisões. Na Figure 2 observamos os resultados de um teste com um dicionário e um nfindlist de pequena dimensão, mas um elevado número autores e keywords por artigo. Aqui é claro que a performance da nossa implementação é melhor. Observa-se também que o número de operações por segundo reduz para as três implementações quando são utilizados oito Threads. A Figure 3 representa um teste com um dicionário de grande dimensão, o máximo que possível, e um nfindlist pequeno e poucos autores e keywords por artigo. Aqui observa-se como a performance muda consoante o número de Threads. Para

um número reduzido de Threads, existem muito poucas colisões, no entanto, quando se aumenta o número de Threads, observamos que o overhead causado pelo bloqueio por lista de colisão compensa em termos de performance. Este overhead é causado pela inicialização de um lock de uma lista de colisões específica e os acessos às estruturas onde se encontram os Locks.

5 Conclusão

Consoante os resultados obtidos, podemos observar que efectuar o bloqueio por lista de colisões permite obter resultados melhores em termos de número de operações por segundo. No caso limite, são obtidos valores semelhantes. Este projecto permitiu avaliar soluções com graus diferentes de granularidade e diferentes estratégias na abordagem de problemas de concorrência e paralelismo.

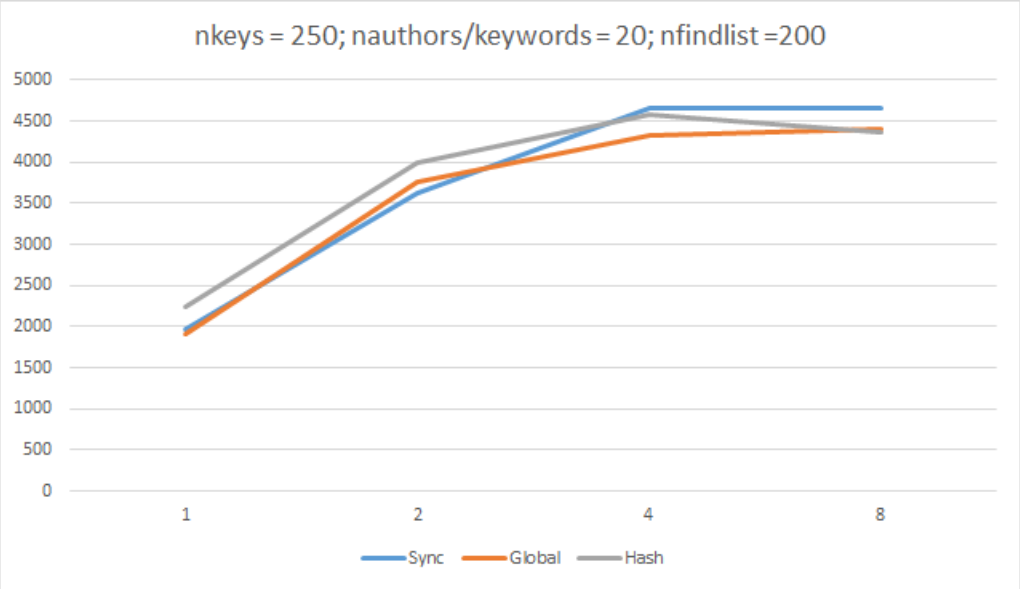

Figura 1: Teste com dicionário pequeno e nfindlist grande

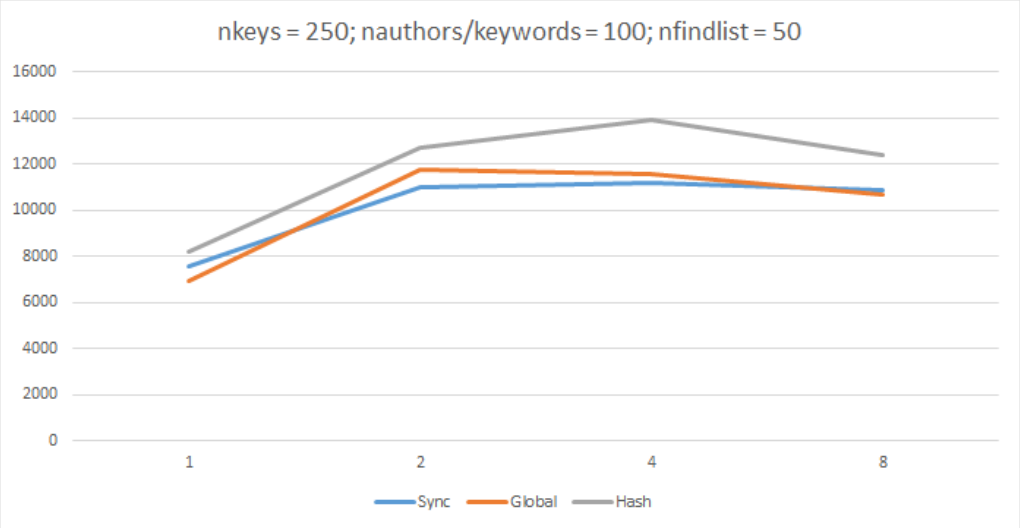

Figura 2: Teste com dicionário e nfindlist pequeno, mas muitos autores/keywords por artigos

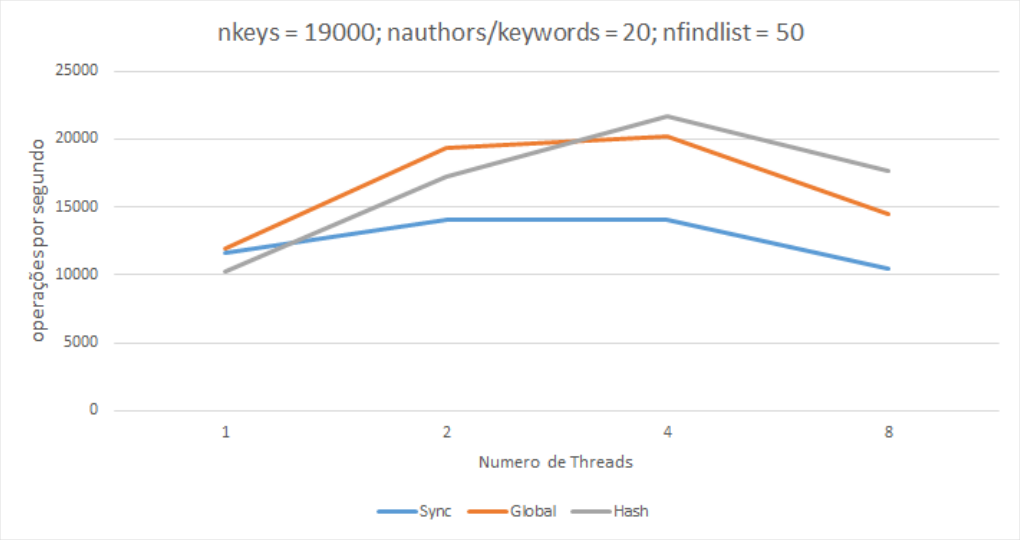

Figura 3: Teste com um dicionário grande e nfindlist pequeno

Referências

[1] https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html

[2] http://tutorials.jenkov.com/java-util-concurrent/lock.html

[3] http://tutorials.jenkov.com/java-util-concurrent/readwritelock.html

[4] Silberschatz, Abraham and Galvin, Peter and Gagne, Greg *"Operating System Concepts, Ninth Edition, Chapter 7* 2012.

[5] https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

[6] https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html