Increasing the performance of time-limited caches

Antal Tátrai, Balázs Dezső, Dr. István Fekete

January 16, 2007

During the development of a huge telecommunication switch, we wanted faster responds from a DNS server. We implemented a local cache for these data, that have own and individual working mechanism. Our work affects the topic of databases, data structures, and concurrent algorithms.

In the informatics we often use caches, when we want to access faster remote data. If we have too much data, easy to make huge caches quickly. Avoiding this problem, we have to declare restrictions for the upper bound of a cache size, for example, we remove an element which using frequency is small, or its time to live value expired.

A special kind of the caches are the time-limited caches. In this case we assign a time to live value for each key-data pair. When a data expires it have to be removed or updated. Usually this action is performed by a thread, called clearer thread, which find the expired elements in the cache, and make the proper action with them. As usual a data's TTL value depends on the changing frequency of the original data and, of course, these TTL periods give an explicit size bound for the cache. Because more thread can use the database, we must to resolve the mutual exclusion. For this, we can use a read-write lock.

During our work, we examine the linked hash tables, with more than one lock. We have a global lock, for the whole database, and other locks for each buckets of the hash table. Furthermore we introduce an additional data structure, which is finally a priority queue grouped or ungrouped elements by the cleaning period. We make connections between the key-data-experiment time nodes and the node of the priority queue, which contains expire times and pointers to the original data nodes. We analyse more priority queue implementations with different theoretical time performance, for example, binary heap, modified binary search-tree, modified 2-3 tree, and the linked queue. If an insertion occurs to the cache, than we insert its expire time to the additional data structure, and the clearer thread will examine only the additional data structure, and it can enumerate the expired elements of the cache. While the binary heap implementation provides $O(\log(n))$ time complexity for insertion and erasure, the simple linked queue could give $O(1)$ modification time when the TTL values are constants but in worst case each modification costs $O(n)$ time. The modified implementation of the binary search-tree and the 2-3 tree gives beneficial theoretical time performance in input sensitive manner.