An overview of Fast Hash Table Lookup Using Extended Bloom Filter

Kristjan Krips

University of Tartu, Estonia

Abstract

Hash tables are a critical part in network processing applications. They are used in flow-state management, monitoring, route lookup, etc. The worst case time for querying an element from the hash table has to be minimized in order to keep the throughput of an application stable. Thus, a hash table that can optimize the worst case performance could fix the bottleneck of the network. For this reason we give an overview of an article that proposed a novel idea for improving the worst case performance. It describes a way to extend and optimize the Bloom Filter. For detailed information, see the original article [1].

Bloom Filter and its improvement

Bloom Filter is a data structure that can be used to check if an element is a member of a set. It is probabilistic and space-efficient but there is a possibility that an incorrect answer is returned, a false positive. However, it always returns a correct answer if an element belongs to a set. For more information about the Bloom Filter, see [2].

This data structure can be used to implement hash tables that perform better than the naïve hash tables by having less collisions and doing a smaller number of queries. The extended Bloom Filter supports exact matching, counting of added items and thus also deleting. It takes advantage of the small, high bandwidth on-chip memory to improve the lookup of an off-chip hash table.

Basic Fast Hash Table (BFHT)

The Basic Fast Hash Table is based on the extended Bloom Filter with counters. An array of counters is maintained on-chip where each counter is associated with a bucket in the hash table. When an item is inserted it is first hashed by k different hash functions and the counters corresponding to the outcome are incremented but each counter is incremented only once per insertion. Querying is similar to insertion, first the value is hashed by k hash functions and then the corresponding counters are checked. Only if the counters are non-zero, the off-chip hash table is accessed to verify the search result. Thus, if the hash table does not contain an element then it will not be accessed. Verification is done on the table entry with the smallest counter value; it is also the main reason why this hash table performs better than the naïve hash table. If an element is not in the table entry then it is a false positive, otherwise we say that the lookup succeeded.

Figure 1 - Insertion of elements into the Basic Fast Hash Table, figure taken from article [1].
Pruned Fast Hash Table
While probing an item, only the table entry with the smallest counter is searched, thus BFHT can be further optimized by deleting the table entries that are never accessed. The resulting table has the same memory requirement as the naïve hash table and is named Pruned Fast Hash Table (PFHT). Pruning does not change the counter values but the smallest counter of an item still contains the queried item. PFHT does not support direct updates to the table, as insertion and deletion can change the counter values and result in an incorrect search result. This is solved by always doing the insertion in the bucket with the smallest counter value. For deletion an off-chip BFHT has to be maintained as the PFHT counters cannot be decremented while guaranteeing the correctness. PFHT can be optimized by rearranging collided items. This is done by increasing the counter value of the bucket which contains several items. It forces the algorithm to check for buckets with smaller counter values. In order to avoid other collisions the results of the move are tested and checked for viability.

Shared-node Fast Hash Table (SFHT)
SFHT allows incremental updates without the use of BFHT. Instead, pointers are used to share the bucket between several items. Therefore, this approach uses less memory than the BFHT but a bit more than PFHT.

Analysis
The analysis showed that the fast hash table algorithms outperform the naïve algorithm by using less time for querying an item. For detailed analysis and information about complexity, see [1].

Conclusion
The Fast Hash Table algorithms describe a possible way to increase the speed of network applications and algorithms. It is important as the network applications and routers are the bottlenecks, which slow up the networks. The algorithm showed that although hash tables have been thoroughly studied it is still possible to create a clever implementation which would show an increase in speed.

References

[2] Bloom Filter 